|  |  |
| --- | --- |
|  | **How to process a json file:** **load configuration parameters from json, etc.**  **How to interact with the file system. And create**  **a working environment with the Module os.**  **Module pprint**  **Collections:**  **OrderedDict, defaultdict, \_\_setitem\_\_, \_\_delitem\_\_**  **Le module collections possède certaines classes concrètes qui dérivent d'ABC**  **abc.ABC**  **abstractmethod**  **Typing:**  **Dict, List, Tuple, Set, Any, Callable, Iterable, Type, BinaryIO, cast, Optional, Tuple, Union, IO**  [**https://docs.python.org/3/glossary.html#term-bytes-like-object**](https://docs.python.org/3/glossary.html#term-bytes-like-object)**:**  **https://docs.python.org/3/library/io.html**  **https://github.com/python/cpython/blob/3.9/Lib/io.py**  **(after autograd)**  **https://github.com/pytorch/pytorch/blob/c6505cc3837eb903f98163e40fad638a1cfeb502/torch/csrc/Module.cpp**  **Types: torch.\_C.\_disabled\_torch\_function\_impl, \_\_deepcopy\_\_**  **https://github.com/pytorch/pytorch/issues/24015 and**  [**https://www.numpy.org/neps/nep-0018-array-function-protocol.html**](https://www.numpy.org/neps/nep-0018-array-function-protocol.html)  **torch/functional.py, test/test\_overrides.py**  **torch.\_C.\_has\_torch\_function, torch.\_C.\_has\_torch\_function\_unary, torch.\_C.\_has\_torch\_function\_variadic, torch.\_C.\_add\_docstr**  **handle\_torch\_function**  **https://pytorch-dev-podcast.simplecast.com/episodes/torch-function-Amez\_iMz**  **overrides some behaviour on torch.tensor**  **\_get\_overloaded\_args**  **TORCH.TENSOR.CLAMP\_, torch.where, torch.norm**  **vars(), locals(), globals**  **Yaml**  **https://pyyaml.org/wiki/PyYAMLDocumentation**  **https://www.cloudbees.com/blog/yaml-tutorial-everything-you-need-get-started**  **Some platforms support YAML s advanced features, including custom datatypes.: https://github.com/yaml/yaml-spec**  [**https://github.com/yaml/www.yaml.org**](https://github.com/yaml/www.yaml.org)  **https://www.json2yaml.com/**  **https://pytorch.org/docs/stable/notes/randomness.html:**  **https://docs.python.org/2/library/weakref.html**  **https://docs.python.org/3/library/weakref.html**  **https://stackoverflow.com/questions/36787603/what-exactly-is-weakref-in-python**  **weakref.proxy**  **https://github.com/python/cpython/blob/2.7/Lib/weakref.py**  **https://github.com/python/cpython/blob/3.9/Lib/weakref.py**  [**https://docs.python.org/3/extending/newtypes.html#weakref-support**](https://docs.python.org/3/extending/newtypes.html#weakref-support)  **Tensor.scatter: Out-of-place version of torch.Tensor.scatter\_()**  **Tensor.scatter\_: Writes all values from the tensor src into self at the indices specified in the index tensor.**  **Tensor.scatter\_add\_: Adds all values from the tensor other into self at the indices specified in the index tensor in a similar fashion as scatter\_().**  **Tensor.scatter\_add: Out-of-place version of torch.Tensor.scatter\_add\_()**  **torch.Tensor.gather**  **TORCH.ARANGE:**  **TORCH.FLIP:**  **TORCH.FULL**  **https://github.com/numpy/numpy/blob/v1.21.0/numpy/core/fromnumeric.py#L2046-L2115**  **https://numpy.org/doc/stable/reference/generated/numpy.clip.html:**  **numpy.clip** |
|  | **Python Imaging Library (or PIL) is an image processing library for the Python programming language:**  **ImageOps,**  **ImageOps.autocontrast**  **ImageOps.invert**  **ImageOps.equalize**  **ImageOps.solarize**  **ImageOps.posterize(img, bits\_to\_keep)**  **ImageEnhance,**  **ImageEnhance.Contrast(img).enhance(factor)**  **ImageEnhance.Color(img).enhance(factor)**  **ImageEnhance.Brightness(img).enhance(factor)**  **ImageEnhance.Sharpness(img).enhance(factor)**  **ImageChops**  **Image**  **PIL.Image.fromarray      #Convert between a PIL Image and a numpy array   #fromarray (obj,  mode)**  **fromarray(obj, mode="'RGB'")**  **mixed = Image.fromarray(mixed.astype(np.uint8))#unsigned long**  **Image.blend(img, mixed, m)**  **Interpolation:  Image.Nearest**  **Image.Bilinear**  **Image.NEAREST**  **Image.BILINEAR**  **Image.BICUBIC**  **Image.LANCZOS**  **Image.HAMMING**  **Image.BOX**  **Requests is a simple and elegant HTTP library for Python, designed for humans. It allows you to send HTTP/1.1 requests extremely easily.**  **There is no need to manually add request strings to your URLs or code your POST data.**  **Keep-alive and pooling of HTTP connections are 100% automatic, thanks to urllib3:**  **https://docs.python-requests.org/en/master/**    **matplotlib.pyplot is a stateful interface to matplotlib. It provides a plotting method similar to MATLAB.**  [**https://matplotlib.org/stable/api/\_as\_gen/matplotlib.pyplot.html**](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)**.**  **Retina display quality for figures:**  **https://matplotlib.org/3.1.1/api/\_as\_gen/matplotlib.pyplot.gca.html:**  **Torch.nn**  **nn.module**  **nn.layers**  **torchvision:**  **https://github.com/pytorch/vision/tree/master/torchvision.**  **The torchvision package consists of popular datasets, model architectures and common image transformations and helper tools for computer vision.**  **torchvision.models: http://man.hubwiz.com/docset/torchvision.docset/Contents/Resources/Documents/models.html**  **torchvision.transforms:**  **T.compose**  **T.interpolate**  **T.normalize**  **torchvision.datasets**  **torchvision.utils**  **torchvision.io**  **torchvision.ops**  [**https://github.com/pytorch/examples/blob/master/mnist/main.py**](https://github.com/pytorch/examples/blob/master/mnist/main.py)  [**https://numpy.org/doc/stable/user/tutorial-svd.html**](https://numpy.org/doc/stable/user/tutorial-svd.html)  **If we want to be able to run the examples in this tutorial, weshould also have**[**matplotlib**](https://matplotlib.org/)**and**[**SciPy**](https://scipy.org/)**installed on our computer.**  **In order to transform our own image into a NumPy array that can be manipulated, we can use the imread function from the**[matplotlib.pyplot](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot)**submodule.**  **Alternatively, we can use the**[imageio.imread](https://imageio.readthedocs.io/en/stable/userapi.html#imageio.imread)**function from the imageio library. Be aware that if you use your own image, you’ll likely need to adapt the steps below.**  **.**  **matplotlib.pyplot.imread(fname, format=None): Read an image from a file into an array.**  [**https://numpy.org/doc/stable/reference/routines.html**](https://numpy.org/doc/stable/reference/routines.html)  **https://numpy.org/doc/stable/reference/generated/numpy.array.html?highlight=array#numpy.array**  **For more information on how images are treated when converted to NumPy arrays, see**[**A crash course on NumPy for images**](https://scikit-image.org/docs/stable/user_guide/numpy_images.html)**from the scikit-image documentation:**  [**https://scikit-image.org/docs/stable/user\_guide/numpy\_images.html**](https://scikit-image.org/docs/stable/user_guide/numpy_images.html)  **Images in scikit-image are represented by NumPy ndarrays. Hence, many common operations can be achieved using standard NumPy methods**  **for manipulating array.**  **https://www.reddit.com/r/scipy/comments/4zdh9x/what\_does\_numpys\_imageshape2\_do/**  [**https://www.bogotobogo.com/python/python\_numpy\_array\_tutorial\_basic\_A.php**](https://www.bogotobogo.com/python/python_numpy_array_tutorial_basic_A.php)**.** |
|  | [**http://man.hubwiz.com/docset/torchvision.docset/Contents/Resources/Documents/transforms.html**](http://man.hubwiz.com/docset/torchvision.docset/Contents/Resources/Documents/transforms.html)   * **Transforms and data augmentation for both image + bbox.: detr/datasets.transforms** * **Transforms are common image transforms. They can be chained together using T.compose** * **T.resize: Resize the input PIL Image to the given size.** * **interpolation (**[int](https://docs.python.org/3/library/functions.html)**, optional) – Desired interpolation. Default is PIL.Image.BILINEAR.** * **T.normalize: Normalize a tensor image with mean and standard deviation** * **T.ToTensor:Convert a PIL Image or numpy.ndarray to tensor.** * **torch.as\_tensor(data, dtype=None, device=None) → TensorConvert the data into a torch.Tensor.** * **Converts a PIL Image or numpy.ndarray (H x W x C) in the range [0, 255] to a torch.FloatTensor of shape (C x H x W) in the range [0.0, 1.0].** * **torch.unbind(input, dim=0) → seq: Removes a tensor dimension.**   **Returns a tuple of all slices along a given dimension, already without it.**   * **torch.stack** * **rescale\_bboxes(out\_bbox)**   + **timm/data/mixup.py** |
|  | **Download the ImageNet dataset and move validation images to labeled subfolders**  **The following script may be helpful: https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.sh**  **mkdir -p n01877812**  **mv ILSVRC2012\_val\_00004844.JPEG n01877812/**  **http://www.adeveloperdiary.com/data-science/computer-vision/how-to-prepare-imagenet-dataset-for-image-classification/**  **Softlink training and validation datasets into the current directory:**  **$ ln -sf /data/imagenet/train-jpeg/ train**  **$ ln -sf /data/imagenet/val-jpeg/ val** |
|  | **Object Detection, Instance and Semantic Segmentation:**  **Detectron2 - https://github.com/facebookresearch/detectron2**  **Segmentation Models (Semantic) - https://github.com/qubvel/segmentation\_models.pytorch**  **EfficientDet (Obj Det, Semantic soon) - https://github.com/rwightman/efficientdet-pytorch**  **Computer Vision / Image Augmentation:**  **Albumentations - https://github.com/albumentations-team/albumentations**  **Kornia - https://github.com/kornia/kornia**  **Knowledge Distillation:**  **RepDistiller - https://github.com/HobbitLong/RepDistiller**  **torchdistill - https://github.com/yoshitomo-matsubara/torchdistill**  **Metric Learning:**  **PyTorch Metric Learning - https://github.com/KevinMusgrave/pytorch-metric-learning** |
|  | **Dataset loading and buildingfrom tensorflow.datsets:**  **tfds.core.DatasetBuilder : https://www.tensorflow.org/datasets/api\_docs/python/tfds/builder?hl=zh-tw**  **DatasetBuilder : https://www.tensorflow.org/datasets/api\_docs/python/tfds/core/DatasetBuilder?hl=zh-tw**  **https://github.com/tensorflow/datasets/blob/v4.4.0/tensorflow\_datasets/core/dataset\_builder.py#L85-L871**  **Typical `DatasetBuilder` usage:**  **```python**  **mnist\_builder = tfds.builder("mnist")**  **mnist\_info = mnist\_builder.info**  **mnist\_builder.download\_and\_prepare()**  **datasets = mnist\_builder.as\_dataset()**  **train\_dataset, test\_dataset = datasets["train"], datasets["test"]**  **assert isinstance(train\_dataset, tf.data.Dataset)**  **# And then the rest of your input pipeline**  **train\_dataset = train\_dataset.repeat().shuffle(1024).batch(128)**  **train\_dataset = train\_dataset.prefetch(2)**  **features = tf.compat.v1.data.make\_one\_shot\_iterator(train\_dataset).get\_next()**  **image, label = features['image'], features['label']**  **"""**  **to prevent excessive drop\_last batch behaviour w/ IterableDatasets**  **# see warnings at** [**https://pytorch.org/docs/stable/data.html#multi-process-data-loading**](https://pytorch.org/docs/stable/data.html#multi-process-data-loading)  **Wraps many (most?) TFDS image-classification datasets**  **from https://github.com/tensorflow/datasets**  **https://www.tensorflow.org/datasets/catalog/overview#image\_classification**  **https://github.com/pytorch/pytorch/issues/33413** |
|  | **Dataset loading and building from torchvision.datsets:**  **timm/data/dataset .py from .dataset import ImageDataset, IterableImageDataset, AugMixDataset**  **IterableImageDataset(root, parser=name, split=split, is\_training=is\_training, batch\_size=batch\_size, \*\*kwargs)**  **IterableImageDataset(data.IterableDataset)**  **ImageDataset(root, parser=name, \*\*kwargs)**  **ImageDataset(data.Dataset) ,**  **AugMixDataset(torch.utils.data.Dataset):**  **"""Dataset wrapper to perform AugMix or other clean/augmentation mixes"""**  **\_search\_split**  **timm/data/ dataset\_factory.py**  **timm.data.dataset\_factory. create\_dataset**  **\_search\_split**  [**timm/data/loader.py**](https://github.com/rwightman/pytorch-image-models/blob/54e90e82a5a6367d468e4f6dd5982715e4e20a72/timm/data/loader.py#L128)  **Dataloader( pytorch/vision, tensorflow/datasets, open-mmlab/mmclassification):**  **https://pytorch.org/vision/stable/datasets.html#torchvision.datasets.ImageNet**  **https://www.tensorflow.org/datasets/catalog/imagenet2012**    [**https://github.com/open-mmlab/mmclassification/blob/master/docs/getting\_started.md**](https://github.com/open-mmlab/mmclassification/blob/master/docs/getting_started.md)  **https://paperswithcode.com/dataset/imagenet:**  **https://image-net.org/challenges/LSVRC/index.php**  [**https://www.kaggle.com/c/imagenet-object-localization-challenge/overview/description**](https://www.kaggle.com/c/imagenet-object-localization-challenge/overview/description)  **Dataset variant accordingy to task nature**  [**https://paperswithcode.com/dataset**](https://paperswithcode.com/dataset)  **Pretrained on more than ImageNet :**  **WSL, SSL, SWSL ResNe(Xt) and the Google Noisy Student EfficientNet models.**  **models pre-trained in weakly-supervised fashion:**  **https://github.com/facebookresearch/WSL-Images**  **Semi-Supervised and Semi-Weakly Supervised ImageNet Models**  **https://github.com/facebookresearch/semi-supervised-ImageNet1K-models** |
|  | **NVIDIA DDP w/ a single GPU per process, multiple processes with APEX, PyTorch DistributedDataParallel w/ multi-gpu,**  **PyTorch single GPU single process**  **torch.distributed**  **https://pytorch.org/tutorials/beginner/dist\_overview.html**  [**https://pytorch.org/docs/stable/distributed.html**](https://pytorch.org/docs/stable/distributed.html)  **torch.distributed.launch #launch-utility**  [**https://gist.github.com/mcarilli/213a4e698e4a0ae2234ddee56f4f3f95**](https://gist.github.com/mcarilli/213a4e698e4a0ae2234ddee56f4f3f95)    **TensorFloat-32, torch.backends.cuda.matmul.allow\_tf32, torch.backends.cudnn.allow\_tf32**  **https://github.com/pytorch/pytorch/blob/0e3b45eaefbef29c36f0198195022a1e4088b3e0/benchmarks/distributed/ddp/compare/python\_ddp.py**  **torch.backends.cudnn.benchmark**  **torch.cuda**  [**https://pytorch.org/docs/stable/notes/cuda.html#tf32-on-ampere**](https://pytorch.org/docs/stable/notes/cuda.html#tf32-on-ampere)  [**https://pytorch.org/docs/stable/backends.html**](https://pytorch.org/docs/stable/backends.html)  [**https://github.com/rwightman/pytorch-image-models/blob/a6e8598aaf90261402f3e9e9a3f12eac81356e9d/timm/utils/cuda.py**](https://github.com/rwightman/pytorch-image-models/blob/a6e8598aaf90261402f3e9e9a3f12eac81356e9d/timm/utils/cuda.py)  [**https://nvidia.github.io/apex/advanced.html**](https://nvidia.github.io/apex/advanced.html) **/timm/utils/cuda.py #L17**  **loss\_scaler = ApexScaler()**  **loss\_scaler = NativeScaler()**  **""" we can add functions as  did in**  **def state\_dict(self):**  **if 'state\_dict' in amp.\_\_dict\_\_:**  **return amp.state\_dict()**  **def load\_state\_dict(self, state\_dict):**  **if 'load\_state\_dict' in amp.\_\_dict\_\_:**  **amp.load\_state\_dict(state\_dict)**  **"""**  **.**  **automatic mixed precision package training:**  **NVIDIA's APEX Examples : https://github.com/pytorch/examples//tree/master/imagenet**  [**https://github.com/NVIDIA/apex/tree/master/examples/imagenet**](https://github.com/NVIDIA/apex/tree/master/examples/imagenet)**# main\_amp.py**  [**https://github.com/NVIDIA/apex/tree/master/examples/dcgan**](https://github.com/NVIDIA/apex/tree/master/examples/dcgan)  **in timm.Data.loader: Prefetcher and Fast Collate inspired by NVIDIA APEX example at**  **https://github.com/NVIDIA/apex/commit/d5e2bb4bdeedd27b1dfaf5bb2b24d6c000dee9be#diff-cf86c282ff7fba81fad27a559379d5bf**  **Note pay attention old api Amp + the old FP16\_Optimizer and new APi use see   https://nvidia.github.io/apex/amp.html/Transition guide for old API users )**  [**https://pytorch.org/docs/stable/amp.html**](https://pytorch.org/docs/stable/amp.html)  **Automatic Mixed Precision examples : https://pytorch.org/docs/stable/notes/amp\_examples.html#amp-examples**  **https://pytorch.org/docs/stable/notes/amp\_examples.html#amp-multigpu**  **Automatic Mixed Precision recipe:  https://pytorch.org/tutorials/recipes/recipes/amp\_recipe.html**  **torch.cuda.amp**  **torch.cuda.amp.autocast**  **Example:**  **# Creates model and optimizer in default precision**  **model = Net().cuda()**  **optimizer = optim.SGD(model.parameters(), ...)**  **for input, target in data:**  **optimizer.zero\_grad()**  **# Enables autocasting for the forward pass (model + loss)**  **with autocast():**  **output = model(input)**  **loss = loss\_fn(output, target)**  **# Exits the context manager before backward()**  **loss.backward()**  **optimizer.step()**  **apex.amp:**  **https://nvidia.github.io/apex/amp.html**  **apex.amp.handle #** **scale\_loss**  **apex.amp.scale\_loss**  **apex.amp.master\_params(optimizer)**  **Three lines enable Amp:**  **# Added after model and optimizer construction**  **model, optimizer = amp.initialize(model, optimizer, flags...)**  **...**  **# loss.backward() changed to:**  **with amp.scale\_loss(loss, optimizer) as scaled\_loss:**  **scaled\_loss.backward()**  **Example :**  **# Declare model and optimizer as usual, with default (FP32) precision**  **model = torch.nn.Linear(D\_in, D\_out).cuda()**  **optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)**  **# Allow Amp to perform casts as required by the opt\_level**  **model, optimizer = amp.initialize(model, optimizer, opt\_level="O1")**  **...**  **# loss.backward() becomes:**  **with amp.scale\_loss(loss, optimizer) as scaled\_loss:**  **scaled\_loss.backward()**  **# old API**  **with amp\_handle.scale\_loss(loss, optimizer) as scaled\_loss:**  **scaled\_loss.backward()**  **->**  **# new API**  **with amp.scale\_loss(loss, optimizer) as scaled\_loss:**  **scaled\_loss.backward()**  **https://nvidia.github.io/apex/\_modules/apex/parallel.html:**  **apex.parallel.DistributedDataParallel**  **apex.parallel.convert\_syncbn\_model**  **https://nvidia.github.io/apex/parallel.html#apex.parallel.convert\_syncbn\_model**  **has\_native\_amp**  **has\_apex**  **wandb,  has\_wandb** |
|  | [**https://pytorch.org/docs/stable/generated/torch.set\_grad\_enabled.html**](https://pytorch.org/docs/stable/generated/torch.set_grad_enabled.html)  **https://pytorch.org/tutorials/beginner/blitz/autograd\_tutorial.html**  **https://pytorch.org/docs/stable/notes/autograd.html#locally-disable-grad-doc    #good to see**  **#torch.autograd.Function**  [**https://pytorch.org/docs/stable/\_modules/torch/autograd/function.html#Function**](https://pytorch.org/docs/stable/_modules/torch/autograd/function.html#Function)  **github.com/pytorch/pytorch/blob/master/torch/autograd/grad\_mode.py**  [**https://pytorch.org/docs/stable/\_modules/torch/autograd/grad\_mode.html#set\_grad\_enabled**](https://pytorch.org/docs/stable/_modules/torch/autograd/grad_mode.html#set_grad_enabled)  **Numerical gradient checking:**  [**https://pytorch.org/docs/stable/autograd.html#grad-check**](https://pytorch.org/docs/stable/autograd.html#grad-check) **#torch.autograd.gradcheck**  **https://pytorch.org/docs/stable/notes/extending.html:**  **extending torch.nn, torch.autograd, torch, and writing custom C extensions utilizing our C libraries:**  **forward(),backward(),save\_for\_backward(),mark\_dirty(),mark\_non\_differentiable(),set\_materialize\_grads()**  **Helper decorator for forward/backward methods of custom autograd functions (subclasses of torch.autograd.Function):**  **torch.cuda.amp.custom\_fwd(fwd=None, \*\*kwargs)**  **torch.cuda.amp.custom\_bwd(bwd)**  **Gradient Scaling :**  **torch.cuda.amp.grad\_scaler: https://pytorch.org/docs/stable/\_modules/torch/cuda/amp/grad\_scaler.html#GradScaler**  **torch.cuda.amp.GradScaler**  **get\_backoff\_factor(), get\_growth\_factor(),get\_scale(),scale, set\_backoff\_factor, set\_growth\_factor, set\_growth\_interval, \_growth\_tracker**  **If you wish to checkpoint the scaler’s state after a particular iteration, state\_dict() should be called after update().**  **...**  **scaler.scale(loss).backward()**  **scaler.unscale\_(optimizer)**  **torch.nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm)**  **scaler.step(optimizer)**  **scaler.update()** |
|  | **torch.optim:**  **https://pytorch.org/docs/stable/optim.html**  **PyTorch RMSProp**  **Create an optimizer:**  **https://github.com/rwightman/pytorch-image-models/blob/54e90e82a5a6367d468e4f6dd5982715e4e20a72/timm/optim/optim\_factory.py**  **optim.SGD**  **fused<name> optimizers by name with NVIDIA Apex installed**  **apex.optimizers:**  **FusedNovoGrad, FusedAdam, FusedLAMB, FusedSGD**  **Adabelief.py, adafactor.py, adahessian.py, adamp.py, lamb.py, lars.py, lookahead.py, madgrad.py, nadam.py, nvnovograd.py, radam.py, rmsprop\_tf.py,**  **sgdp.py**   |  | | --- | |  | |  | |  | |  | |  | |  | |  | |  |   **add\_weight\_decay**  **optimizer\_kwargs(cfg/argparse to kwargs helper)**  **Adaptive Gradient Clipping(Nfnets) :** [**https://github.com/rwightman/pytorch-image-models/blob/a6e8598aaf90261402f3e9e9a3f12eac81356e9d/timm/utils/agc.py**](https://github.com/rwightman/pytorch-image-models/blob/a6e8598aaf90261402f3e9e9a3f12eac81356e9d/timm/utils/agc.py)  **\* Official JAX impl (paper authors): https://github.com/deepmind/deepmind-research/tree/master/nfnets**  **\* Phil Wang's PyTorch gist: https://gist.github.com/lucidrains/0d6560077edac419ab5d3aa29e674d5c**  **https://github.com/rwightman/pytorch-image-models/blob/a6e8598aaf90261402f3e9e9a3f12eac81356e9d/timm/utils/clip\_grad.py**  **adaptive\_clip\_grad, torch.nn.utils.clip\_grad\_norm\_, torch.nn.utils.clip\_grad\_value\_**    **optimizer (torch.optim.Optimizer)**  **step(optimizer, \*args, \*\*kwargs)**     |  | | --- | |  | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |
|  | **create\_scheduler :**  **Learning rate schedulers:**  **AllenNLP schedulers**  **FAIRseq lr\_scheduler**  **SGDR: Stochastic Gradient Descent with Warm Restarts (https://arxiv.org/abs/1608.03983)**  **Schedulers include step, cosine w/ restarts, tanh w/ restarts, plateau**  **https://github.com/rwightman/pytorch-image-models/blob/3d9c23af879283e80c2c208786d5613351ca040b/timm/scheduler/scheduler\_factory.py#L12**  **from .cosine\_lr import CosineLRScheduler**  **from .multistep\_lr import MultiStepLRScheduler**  **from .plateau\_lr import PlateauLRScheduler**  **from .poly\_lr import PolyLRScheduler**  **from .step\_lr import StepLRScheduler**  **from .tanh\_lr import TanhLRScheduler** |
|  | **Timm:**  [**https://fastai.github.io/timmdocs/**](https://fastai.github.io/timmdocs/)  [**https://github.com/topics/pytorch**](https://github.com/topics/pytorch)  [**https://rwightman.github.io/pytorch-image-models**](https://rwightman.github.io/pytorch-image-models)  [**https://github.com/rwightman/pytorch-image-models**](https://github.com/rwightman/pytorch-image-models)  **https://github.com/rwightman/pytorch-image-models/releases**  [**https://rwightman.github.io/pytorch-image-models/results/**](https://rwightman.github.io/pytorch-image-models/results/)  **https://rwightman.github.io/pytorch-image-models/results/**  **https://github.com/rwightman/pytorch-image-models/blob/master/results/README.md**  **training hparams for some train examples that produce SOTA ImageNet results.:**  **https://rwightman.github.io/pytorch-image-models/training\_hparam\_examples/**  [**https://rwightman.github.io/pytorch-image-models\models\”ModelName**](https://rwightman.github.io/pytorch-image-models\models\)**:**   * **adapt configuration parameters to the model and the image transformation functions** * **load, read and preprocess an image :** [**https://github.com/rwightman/pytorch-image-models/timm/data**](https://github.com/rwightman/pytorch-image-models/timm/data)     **from .auto\_augment import RandAugment, AutoAugment, rand\_augment\_ops, auto\_augment\_policy,\**  **rand\_augment\_transform, auto\_augment\_transform**  **from .config import resolve\_data\_config**  **from .constants import \***  **from .dataset import ImageDataset, IterableImageDataset, AugMixDataset**  **from .dataset\_factory import create\_dataset**  **torch.utils.data**  **torch.utils.data.get\_worker\_info()**  **torch.utils.data.Dataset**  **data.IterableDataset:**  **With PyTorch IterableDatasets, each worker in each replica operates in isolation, the final batch from each worker could be a different size.**  **This is similar to common handling in DistributedSampler for normal Datasets but a bit worse since there are up**  **to N \* J extra samples with IterableDatasets.**  **from .loader import create\_loader**  **fast\_collate, PrefetchLoader, MultiEpochsDataLoader, \_RepeatSampler**  **from .mixup import Mixup, FastCollateMixup**  **from .parsers import**  **create\_parser**  **parser\_tfds, ParserTfds(Parser)**  **class\_map.py**  **load\_class\_map**  **constants.py: #IMG\_EXTENSIONS = ('.png', '.jpg', '.jpeg')**  **parser\_image\_folder , ParserImageTar, parser\_image\_in\_tar, ParserImageInTar**  **from .real\_labels import RealLabelsImagenet**  **from .transforms import \***  **from .transforms\_factory import create\_transform**  **import math**  **import torch**  **from torchvision import transforms**  **transforms.Resize     transforms.CenterCrop, transforms.Normalize, mean=torch.tensor(mean), transforms.Compose,**  **transforms.RandomHorizontalFlip, transforms.RandomVerticalFlip, transforms.ColorJitter**  **from timm.data.constants import IMAGENET\_DEFAULT\_MEAN, IMAGENET\_DEFAULT\_STD, DEFAULT\_CROP\_PCT**  **from timm.data.auto\_augment import rand\_augment\_transform, augment\_and\_mix\_transform, auto\_augment\_transform  #####**  **from timm.data.transforms import \_pil\_interp, RandomResizedCropAndInterpolation, ToNumpy, ToTensor**  **from timm.data.random\_erasing import**         * **list the pretrained models.** * **prepare or load a classes file.** * **load a pretrained model and create our own model:**   **https://github.com/ timm.models.factory.py, https://github.com/pytorch/vision/tree/master/torchvision**  **https://github.com/rwightman/pytorch-image-models/blob/master/train.py**  **timm.models..registry**  **timm.models.helpers**  **timm.models.layers**  **split\_batchnorm.py**  **nn. BatchNorm2d, nn.SplitBatchNorm2d, convert\_splitbn\_model, torch.nn.ModuleList**  **torch.nn.modules.batchnorm.\_BatchNorm**  **torch.nn.modules.batchnorm.\_InstanceNorm, \_verify\_batch\_size, \_verify\_spatial\_size**  **instance\_norm, group\_norm, local\_response\_norm,convert\_sync\_batchnorm,**  **torch.nn.SyncBatchNorm, convert\_syncbn\_model**  **create\_syncbn\_process\_group**  **apex.parallel.convert\_syncbn\_model**  **To create a model, we’ll need Module. To create Module, we’ll need Parameter:**  **torch/nn/parameter.py**  **torch.\_utils.\_rebuild\_parameter: https://fossies.org/linux/pytorch/torch/\_utils.py**  **https://github.com/pytorch/pytorch/tree/c6505cc3837eb903f98163e40fad638a1cfeb502/torch/utils/data/\_utils**  **torch/overrides.py**  **timm.models.hub**  **torch.hub.list**  **torch.hub.help**  **create a hubconfig file:** **pytorch/vision.hubconf.py**  **torch.hub.load**  **torch.hub.download\_url\_to\_file**  **torch.hub.load\_state\_dict\_from\_url**  **torch.hub.HASH\_REGEX, torch.hub.urlparse,**  **torch.hub.get\_dir**  **torch.hub.\_get\_torch\_home**  **Torch.hub.py,** [**https://pytorch.org/docs/stable/\_modules/torch/hub.htm**](https://pytorch.org/docs/stable/_modules/torch/hub.htm)**.**  [**https://github.com/huggingface/huggingface\_hub**](https://github.com/huggingface/huggingface_hub)**;  https://huggingface.co/docs**  **huggingface\_hub/constants.py.**  **huggingface\_hub/file\_download.py**  **hf\_hub\_url, has\_hf\_hub, hf\_split**  **cached\_download, \_download\_from\_hf, load\_model\_config\_from\_hf**  **load\_state\_dict\_from\_hf**  **get\_cache\_dir, urllib.parse**  **download\_cached\_file**  **adapt\_input\_conv**  **load\_pretrained, load\_checkpoint, load\_state\_dict**  **set\_layer, extract\_layer, layer.reset\_parameters()**  **A new testing technique Mean-Max Pooling which can further improve the performance of a well trained CNN**  **in the testing phase without the need of any training/fine-tuining process.**  **A 'Test Time Pool' wrapper: https://github.com/cypw/DPNs dual path networks:**  **All models have a common default configuration interface and API for**  **accessing/changing the classifier - get\_classifier and reset\_classifier**  **doing a forward pass on just the features - forward\_features (see documentation)**  **these makes it easy to write consistent network wrappers that work with any of the models**  **load\_custom\_pretrained**  **https://www.tensorflow.org/api\_docs/python/tf/train/ExponentialMovingAverage**  **Exponential Moving Average** **of model updates: /timm/utils/model\_ema.py:**  **ModelEma, \_load\_checkpoint, update,** **ModelEmaV2**    **setattr , adapt\_model\_from\_string, adapt\_model\_from\_file, default\_cfg\_for\_features,**  **overlay\_external\_default\_cfg, set\_default\_kwargs, filter\_kwargs, update\_default\_cfg\_and\_kwargs**  **build\_model\_with\_cfg, model\_parameters, named\_apply, named\_modules, \_module\_to\_models**  **\_model\_to\_module, \_model\_entrypoints, \_model\_has\_pretrained,\_model\_default\_cfgs, split\_model\_name**  **safe\_model\_name**  **torch.nn.modules.module.py#**  **register\_buffer #** **BatchNorm’s running\_mean**  **register\_parameter**  **register\_backward\_hook, nn.Module.register\_full\_backward\_hook(),register\_forward\_hook**  **PyTorch Feature Extraction Helpers:https://github.com/rwightman/pytorch-image-models/blob/ timm/models/features.py:**  **FeatureListNet, FeatureHookNet, FeatureDictNet**  **torchvision IntermediateLayerGetter**  **https://github.com/pytorch/vision/blob/d88d8961ae51507d0cb680329d985b1488b1b76b/torchvision/models/\_utils.py**  **fnmatch.filter, fnmatch.fnmatch**  **output\_stride, num\_classes : timm/models/inception\_resnet\_v2.py**  **out\_indices, only features:** **timm/models/helpers.py, timm/models/features.py**  **timm/models/BiT.py**  **Example:**  **# For checkpoint saved in local github repo, e.g. <RELATIVE\_PATH\_TO\_CHECKPOINT>=weights/save.pth**  **dirname = os.path.dirname(\_\_file\_\_)**  **checkpoint = os.path.join(dirname, <RELATIVE\_PATH\_TO\_CHECKPOINT>)#find the path joining these 2 paths**  **state\_dict = torch.load(checkpoint)**  **model.load\_state\_dict(state\_dict)**  **# For checkpoint saved elsewhere**  **checkpoint = 'https://download.pytorch.org/models/resnet18-5c106cde.pth'**  **model.load\_state\_dict(torch.hub.load\_state\_dict\_from\_url(checkpoint, progress=False))**  **#torch.hub.load(repo\_or\_dir, model, \*args, \*\*kwargs):**  **load a model from a github repo or a local directory.**  **Note: Loading a model is the typical use case, but this can also be used to for loading other objects such as tokenizers, loss functions, etc.**  **#torch.hub.download\_url\_to\_file('https://s3.amazonaws.com/pytorch/models/resnet18-5c106cde.pth', '/tmp/temporary\_file')**  **# torch.hub.load\_state\_dict\_from\_url(url, model\_dir=None, map\_location=None, progress=True, check\_hash=False, file\_name=None)**  **Loads the Torch serialized object at the given URL.**  **#** **dir(model) to see all available methods of the model.**  **#** **model.load\_state\_dict**  **#model.parameters**  **#help(model.foo) to check what arguments model.foo takes to run**  **# torch.hub.get\_dir():**  **Get the Torch Hub cache directory used for storing downloaded models & weights.**  **If set\_dir() is not called, default path is $TORCH\_HOME/hub where environment variable $TORCH\_HOME defaults**  **to $XDG\_CACHE\_HOME/torch. $XDG\_CACHE\_HOME follows the X Design Group specification of the Linux filesystem layout,**  **with a default value ~/.cache if the environment variable is not set.**  **# torch.hub.set\_dir(d):**  **Optionally set the Torch Hub directory used to save downloaded models & weights.**  **Parameters d (string) – path to a local folder to save downloaded models & weights.**  **#Caching logic:**  **By default, we don’t clean up files after loading it. Hub uses the cache by default if it already exists in the directory returned by get\_dir().**  **Users can force a reload by calling hub.load(..., force\_reload=True). This will delete the existing github folder and downloaded weights,**  **reinitialize a fresh download. This is useful when updates are published to the same branch, users can keep up with the latest release.**  **#Known limitations:**  **Torch hub works by importing the package as if it was installed. There’re some side effects introduced by importing in Python.**  **For example, you can see new items in Pytho caches sys.modules and sys.path\_importer\_cache which is normal Python behavior.**  **A known limitation that worth mentioning here is user CANNOT load two different branches of the same repo in the same python process.**  **It’s just like installing two packages with the same name in Python, which is not good. Cache might join the party and give you**  **surprises if you actually try that. Of course it’s totally fine to load them in separate processes.**   * **get the channel and resolution reduction information of the model.** * **get the model predictions and print shape of probabilities.** * **get the top 5, top 1 predictions class-names.** * **how to extract features from the backbone, the task head or the penultimate layer.** * **multi-scale backbone feature extraction for use in downstream tasks.** * **fine-tune the model by changing the classifier layer or adapt the timm s’ training script to use our dataset.(github.com/pytorch-image-models/train.py). follow also** [**https://rwightman.github.io/pytorch-image-models/scripts/**](https://rwightman.github.io/pytorch-image-models/scripts/) **and huggingface/transformers.** * **adapt the validate.py and inference.py to use our dataset.** * **save the new pretrained model.**   [**https://pytorch.org/docs/stable/jit.html**](https://pytorch.org/docs/stable/jit.html)  **https://pytorch.org/tutorials/beginner/Intro\_to\_TorchScript\_tutorial.html**  **Check out the NeurIPS demo for converting machine translation models using TorchScript:**  [**https://colab.research.google.com/drive/1HiICg6jRkBnr5hvK2-VnMi88Vi9pUzEJ**](https://colab.research.google.com/drive/1HiICg6jRkBnr5hvK2-VnMi88Vi9pUzEJ)  **https://docs.aws.amazon.com/**  **https://docs.aws.amazon.com/dlami/latest/devguide/tutorial-pytorch.html(Deep Learning AMI /amazon )**  **https://pytorch.org/tutorials/advanced/cpp\_export.html              #c++=~ cpp**  **torch.jit.ScriptModule:   https://pytorch.org/docs/stable/generated/torch.jit.ScriptModule.html**  **https://pytorch.org/docs/stable/\_modules/torch/jit/\_script.html#ScriptModule**  **torch.jit.script: https://github.com/pytorch/pytorch/blob/6cac7ca98054feb299c2d68994809b547f3a3c2e/torch/jit/\_script.py#L459**  **NLPGraph** |
|  | **Python transformers Module: https://www.programcreek.com/python/index/11447/transformers.**  **transformers.PreTrainedModel(for pytorch): https://github.com/plkmo/NLP\_Toolkit**  **transformers.TFPreTrainedModel (for TensorFlow)**  **from ..modeling\_tf\_utils import TFPreTrainedModel**  **from ..modeling\_utils import**  **transformers.PreTrainedFeatureExtractor**  **transformers.PreTrainedTokenizerFast**  **HuggingFace Transformers library :** [**huggingface**](https://github.com/huggingface)**/**[**transformer**](https://github.com/huggingface/transformers)**,** [**huggingface**](https://github.com/huggingface)**/ huggingface\_hub,** [**huggingface/datasets**](https://github.com/huggingface)  [**https://github.com/huggingface/transformers**](https://github.com/huggingface/transformers)**,** [**https://github.com/huggingface/datasets**](https://github.com/huggingface/datasets)**, https://github.com/huggingface/tokenizers, https://github.com/huggingface/huggingface\_hub.**  **https://huggingface.co/:**  **https://huggingface.co/transformers/pretrained\_models.html**  **https://huggingface.co/inference-api**  **Create our new repository on website : https://huggingface.co/oumeima/InfraRedTransF/new/main/?filename=README.md**  [**https://huggingface.co/pricing**](https://huggingface.co/pricing)  **a single API through which any Transformer model can be loaded, trained, and saved.**  **An end-to-end example where we use a model and a tokenizer together to replicate the pipeline API.**  **-the model API**  **-the tokenizer API: okenizers take care of the first and last processing steps, handling the conversion from text to numerical inputs for the neural network,**  **and the conversion back to text when it is needed.**  **-how to handle sending multiple sentences through a model in a prepared batch, then wrap it all up with a closer look at the high-level tokenizer function.**  **tokenizer : Splitting the input into words, subwords, or symbols (like punctuation) that are called tokens**  **Mapping each token to an integer**  **Adding additional inputs that may be useful to the model**  **from transformers import pipeline**  **https://github.com/huggingface/transformers/tree/master/src/transformers/pipelines**  **https://huggingface.co/transformers/main\_classes/pipelines.html**  **transformers.pipeline-→ transformers.pipelines.base.Pipeline**  **TextClassificationPipeline**  **classifier = pipeline("sentiment-analysis")**  **from .text\_classification import TextClassificationPipeline: src/transformers/pipelines/text\_classification.py**  **classifier = pipeline("text-classification", model=model\_name)**  **AutoModelForSequenceClassification, AutoModelForSequenceClassification.from\_pretrained**  **Tokenizer=AutoTokenizer, AutoTokenizer.from\_pretrained(checkpoint)**  **tokenizer.convert\_ids\_to\_tokens**  **choose other datset to make experiments with: https://huggingface.co/datasets**  **Docs » Loading a Dataset: https://huggingface.co/docs/datasets/loading\_datasets.html#from-the-huggingface-hub**  **tokenize\_function for input batch.**  **data\_collator =DataCollatorWithPadding(tokenizer)**  **metric = load\_metric("dataset\_name") examples Training Loss, Validation Loss, Accuracy, F1, ect.**  **def compute\_metrics(eval\_preds):**  **logits, labels = eval\_preds**  **predictions = np.argmax(logits, axis=-1)**  **return metric.compute(predictions=predictions, references=labels)**  **TrainingArguments,**  **“Model\_id”**  **per\_device\_train\_batch\_size=,**  **per\_device\_eval\_batch\_size=6,**  **learning\_rate=,**  **weight\_decay=0,**  **evaluation\_strategy="epoch",**  **logging\_strategy="epoch",**  **log\_level="error",**  **push\_to\_hub=True,**  **push\_to\_hub\_model\_id=**  **# push\_to\_hub\_organization="huggingface",**  **push\_to\_hub\_token=**  **trainer = Trainer( model,**  **training\_args,**  **train\_dataset=tokenized\_datasets["train"],**  **eval\_dataset=tokenized\_datasets["validation"],**  **data\_collator=data\_collator,**  **tokenizer=tokenizer,**  **compute\_metrics=compute\_metrics,)**  **trainer.train()**  **trainer.push\_to\_hub()**  **model.push\_to\_hub("finetuned-bert-mrpc")**  **tokenizer.push\_to\_hub**  **model.config.push\_to\_hub.**  **adapt a sentencePiece, a tokenizer and a pipeline for our framework into the huggingface repository and define its model card.**  **load\_dataset, load\_metric**  **The models in the Hub are not limited to 🤗 Transformers or even NLP.**  **There are  models from timm: https://github.com/rwightman/pytorch-image-models(vision)**  **from https://github.com/flairNLP/flair(for NLP)**  **from https://github.com/allenai/allennlp(for NLP)**  **from  Asteroid  The PyTorch-based audio source separation toolkit for researchers.(for speech)**  **from  pyannote (for speech)         https://github.com/pyannote/pyannote-audio: pyannote.audio is an open-source toolkit written in Python for speaker diarization.**  **Based on PyTorch machine learning framework, it provides a set of trainable end-to-end neural building blocks that can be combined and jointly optimized to build speaker diarization pipelines:**    **to name a few.**  **see also:  modification is occuring by the authors for new frameworks:  https://github.com/huggingface/transformers/tree/master/examples/research\_projects**  **https://huggingface.co/transformers/bertology.html:  growing field of study concerned with investigating the inner working of large-scale transformers like BERT (that some call “BERTology”).**  **full training in Colab(course)** |
|  | [**https://github.com/fastai**](https://github.com/fastai)  [**https://course.fast.ai/start\_colab**](https://course.fast.ai/start_colab)  **https://course.fast.ai/datasets**  [**https://docs.fast.ai/vision**](https://docs.fast.ai/vision)**.**  **#list of jyputer notebooks, modeules(.py), custoon url for the website, url for pushing document in git.#examples:**  **https://github.com/fastai/fastai/blob/4661ea1b7603578c5faaf18032449e0c57a8fa63/fastai/\_nbdev.py**  [**https://github.com/fastai/fastai/tree/master/fastai/callback**](https://github.com/fastai/fastai/tree/master/fastai/callback)  [**https://github.com/fastai/fastai2/tree/master/nbs**](https://github.com/fastai/fastai2/tree/master/nbs)**:**  [**00\_torch\_core.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/00_torch_core.ipynb)  [**01\_layers.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/01_layers.ipynb)  [**02\_data.load.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/02_data.load.ipynb)  [**03\_data.core.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/03_data.core.ipynb)  [**04\_data.external.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/04_data.external.ipynb)  [**05\_data.transforms.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/05_data.transforms.ipynb)  [**06\_data.block.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/06_data.block.ipynb)  [**07\_vision.core.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/07_vision.core.ipynb)  [**09\_vision.augment.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/09_vision.augment.ipynb)  [**12\_optimizer.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/12_optimizer.ipynb)  [**13b\_metrics.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/13b_metrics.ipynb)  [**14\_callback.schedule.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/14_callback.schedule.ipynb)  [**15\_callback.hook.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/15_callback.hook.ipynb)  [**15a\_vision.models.unet.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/15a_vision.models.unet.ipynb)  [**17\_callback.tracker.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/17_callback.tracker.ipynb)**: make decisions depending on how a monitored metric/loss behaves.**  [**19\_callback.mixup.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/19_callback.mixup.ipynb)  [**70\_callback.wandb.ipynb**](https://github.com/oumeimaghnimi/Huggingface_hub_learning/blob/master/nbs/70_callback.wandb.ipynb)  [**74\_callback.cutmix.ipynb**](https://github.com/fastai/fastai2/blob/master/nbs/74_callback.cutmix.ipynb)  **Etc.**  **Track model training at scale, Build better models more efficiently with Weights & Biases experiment tracking.**  **https://github.com/wandb:**  [**https://docs.wandb.ai/**](https://docs.wandb.ai/)**:**  [**https://github.com/fastai/fastai/blob/4661ea1b7603578c5faaf18032449e0c57a8fa63/fastai/l**](https://github.com/fastai/fastai/blob/4661ea1b7603578c5faaf18032449e0c57a8fa63/fastai/learner.py)  **/learner.py**  **Tracker.py**  **wandb.py**  **1. learner.py**  **To  test these   Callbacks, we deploy synthetic learner in:**  **https://github.com/fastai/fastai/blob/master/fastai/test\_utils.py.....**  **.................delegates of learner: https://github.com/fastai/fastai/blob/4661ea1b7603578c5faaf18032449e0c57a8fa63/fastai/learner.py#L83......**  **@delegates(Learner.\_\_init\_\_)**  **def synth\_learner(n\_trn=10, n\_val=2, cuda=False, lr=1e-3, data=None, model=None, \*\*kwargs):**  **if data is None: data=synth\_dbunch(n\_train=n\_trn,n\_valid=n\_val, cuda=cuda)**  **if model is None: model=RegModel()**  **return Learner(data, model, lr=lr, loss\_func=MSELossFlat(),**  **opt\_func=partial(SGD, mom=0.9), \*\*kwargs)**  **2.wandb.py:**  **Integration of fastai with weights&biases wandb(W&B):**  **https://docs.fast.ai/callback.wandb.html:**  [**https://github.com/fastai/fastai/blob/master/fastai/callback/wandb.py**](https://github.com/fastai/fastai/blob/master/fastai/callback/wandb.py)  **https://docs.wandb.ai/guides/integrations/fastai:**  **Start Logging with W&B**  **WandbCallback Arguments**  **Data Visualization with W&B Tables**  **Examples**  **https://wandb.ai/borisd13/demo\_config/reports/Visualize-Track-Compare-Fastai-Models-Vmlldzo4MzAyNA:**  **Visualize, compare, and iterate on fastai models with Weights & Biases**  **Guides**  **Experiment Tracking: Visualize experiments in real time**  **Integrations: PyTorch, Keras, Hugging Face, and more**  **Hyperparameter Tuning: Optimize models quickly**  **Data + Model Versioning: Version datasets and models**  **Data Visualization: Visualize predictions across model versions.**  **Collaborative Reports: Describe and share findings.**  **Self-Hosting: Private cloud and local hosting of the W&B app**  **Once your have defined your Learner, before you call to fit or fit\_one\_cycle, you need to initialize wandb:**  **import wandb**  **wandb.init()**    **Then you add the callback to your learner or call to fit methods, potentially with SaveModelCallback if you want to save the best model:**  **from fastai.callback.wandb import \***  **# To log only during one training phase# fitting the model while logging the callback**  **learn.fit(..., cbs=WandbCallback())**  **# To log continuously for all training phases# fitting the model while logging the callback**  **learn = learner(..., cbs=WandbCallback())**  **Datasets and models can be tracked through the callback or directly through log\_model and log\_dataset functions.**  **3.Tracker.py**  **Callbacks that make decisions depending how a monitored metric/loss behaves Tracking callbacks**  **class TerminateOnNaNCallback**  **class TrackerCallback**  **class EarlyStoppingCallback**  **class SaveModelCallback**  **ReduceLROnPlateau**  **class ReduceLROnPlateau**  **https://fastai.github.io/timmdocs/    & Training / Frameworks: fastai -** [**https://github.com/fastai/fastai**](https://github.com/fastai/fastai)  **https://www.bookstack.cn/read/th-fastai-book/# see all good for building a model**  **https://www.bookstack.cn/read/th-fastai-book/spilt.2.5443c76c2b161687.md** |
|  | **https://www.w3resource.com/pandas/dataframe/dataframe-mul.php** |
|  | **colab-env-testbed**  **A simply jupyter notebook for verifying that the colab-env module works as intended in the real world!** |
|  | **Prepare for training(huggingface)**  **The dataloaders we will use to iterate over batches.**  **Set the format of the datasets so they return PyTorch tensors instead of lists.**    **o quickly check there is no mistake in the data processing, we can inspect a batch like this:**  **for batch in train\_dataloader:**  **break**  **{k: v.shape for k, v in batch.items()}**  **the learning rate scheduler used by default is just a linear decay from the maximum value (5e-5) to 0.**  **To properly define it, we need to know the number of training steps we will take, which is the number of epochs we want to run multiplied by the number of training batches (which is the length of our training dataloader).**  **The Trainer uses three epochs by default, so we will follow that:**  **One last thing: we will want to use the GPU if we have access to one (on a CPU, training might take several hours instead of a couple of minutes).**  **To do this, we define a device we will put our model and our batches on:**  **import torch**  **device = torch.device("cuda") if torch.cuda.is\_available() else torch.device("cpu")**  **model.to(device)**  **device**  **output : device(type='cuda')**  **get some sense of when training will be finished, we add a progress bar over our number of training steps, using the tqdm library:**  **from tqdm.auto import tqdm**  **progress\_bar = tqdm(range(num\_training\_steps))**  **The evaluation loop:metric.compute, add\_batch.**    **Supercharge your training loop with 🤗 Accelerate:  Accelerate library,**  **The first line to add is the import line. The second line instantiates an Accelerator object that will look at the environment and initialize the proper distributed setup.**  **🤗 Accelerate handles the device placement for you, so you can remove the lines that put the model on the device (or, if you prefer, change them to use accelerator.device instead of device).**  **sends the dataloaders, the model, and the optimizer to accelerator.prepare**  **This will wrap those objects in the proper container to make sure your distributed training works as intended. The remaining changes to make are removing the line that puts the batch on the device (again, if you want to keep this you can just change it to use accelerator.device) and replacing loss.backward() with accelerator.backward(loss).**  **the complete training loop looks like with 🤗 Accelerate:**  **from accelerate import Accelerator**  **from transformers import AdamW, AutoModelForSequenceClassification, get\_scheduler**  **accelerator = Accelerator()**  **model = AutoModelForSequenceClassification.from\_pretrained(checkpoint, num\_labels=2)#when you instantiate one of the AutoModelForXxx classes with a pretrained language model (such as bert-base-uncased) that corresponds to a different task than the one for which it was trained? The head of the pretrained model is discarded and a new head suitable for the task is inserted instead.**  **optimizer = AdamW(model.parameters(), lr=3e-5)**  **train\_dl, eval\_dl, model, optimizer = accelerator.prepare(**  **train\_dataloader, eval\_dataloader, model, optimizer)**  **num\_epochs = 3**  **num\_training\_steps = num\_epochs \* len(train\_dl)**  **lr\_scheduler = get\_scheduler(**  **"linear",**  **optimizer=optimizer,**  **num\_warmup\_steps=0,**  **num\_training\_steps=num\_training\_steps)**  **progress\_bar = tqdm(range(num\_training\_steps))**  **model.train()**  **for epoch in range(num\_epochs):**  **for batch in train\_dl:**  **outputs = model(\*\*batch)**  **loss = outputs.loss**  **accelerator.backward(loss)**    **optimizer.step()**  **lr\_scheduler.step()**  **optimizer.zero\_grad()**  **progress\_bar.update(1)**    **Putting this in a train.py script will make that script runnable on any kind of distributed setup.**  **To try it out in your distributed setup, run the command:**  **accelerate config #in cmd prompt**  **accelerate launch train.py #in cmd**    **which will prompt you to answer a few questions and dump your answers in a configuration file used by this command:**  **accelerate launch train.py #in cmd**  **which will launch the distributed training.**  **If you want to try this in a Notebook (for instance, to test it with TPUs on Colab), just paste the code in a training\_function and run a last cell with:**  **In notebook: for configuration set up of distribution(GPU, CPU, etc)**  **from accelerate import notebook\_launcher**  **notebook\_launcher(training\_function)**  **full training in Colab;(course)** |