LearningRateFinder#

Learning rate range test. The learning rate range test increases the learning rate in a pre-training run between two boundaries in a linear or exponential manner. It provides valuable information on how well the network can be trained over a range of learning rates and what is the optimal learning rate. Example (fastai approach): >>> lr_finder = LearningRateFinder(net, optimizer, criterion) >>> lr_finder.range_test(data_loader, end_lr=100, num_iter=100) >>> lr_finder.get_steepest_gradient() >>> lr_finder.plot() # to inspect the loss-learning rate graph Example (Leslie Smith's approach): >>> lr_finder = LearningRateFinder(net, optimizer, criterion) >>> lr_finder.range_test(train_loader, val_loader=val_loader, end_lr=1, num_iter=100, step_mode="linear") Gradient accumulation is supported; example: >>> train_data = … # prepared dataset >>> desired_bs, real_bs = 32, 4 # batch size >>> accumulation_steps = desired_bs // real_bs # required steps for accumulation >>> data_loader = torch.utils.data.DataLoader(train_data, batch_size=real_bs, shuffle=True) >>> acc_lr_finder = LearningRateFinder(net, optimizer, criterion) >>> acc_lr_finder.range_test(data_loader, end_lr=10, num_iter=100, accumulation_steps=accumulation_steps) By default, image will be extracted from data loader with x["image"] and x[0], depending on whether batch data is a dictionary or not (and similar behaviour for extracting the label). If your data loader returns something other than this, pass a callable function to extract it, e.g.: >>> image_extractor = lambda x: x["input"] >>> label_extractor = lambda x: x[100] >>> lr_finder = LearningRateFinder(net, optimizer, criterion) >>> lr_finder.range_test(train_loader, val_loader, image_extractor, label_extractor) References: Modified from: davidtvs/pytorch-lr-finder. Cyclical Learning Rates for Training Neural Networks: https://arxiv.org/abs/1506.01186 Constructor. model (Module) – wrapped model. optimizer (Optimizer) – wrapped optimizer. criterion (Module) – wrapped loss function. device (Union[str, device, None]) – device on which to test. run a string ("cpu" or "cuda") with an optional ordinal for the device type (e.g. "cuda:X", where is the ordinal). Alternatively, can be an object representing the device on which the computation will take place. Default: None, uses the same device as model. memory_cache (bool) – if this flag is set to True, state_dict of model and optimizer will be cached in memory. Otherwise, they will be saved to files under the cache_dir. cache_dir (Optional[str]) – path for storing temporary files. If no path is specified, system-wide temporary directory is used. Notice that this parameter will be ignored if memory_cache is True. amp (bool) – use Automatic Mixed Precision pickle_module – module used for pickling metadata and objects, default to pickle. this arg is used by torch.save, for more details, please check: https://pytorch.org/docs/stable/generated/torch.save.html#torch.save. pickle_protocol (int) – can be specified to override the default protocol, default to 2. this arg is used by torch.save, for more details, please check: https://pytorch.org/docs/stable/generated/torch.save.html#torch.save. verbose (bool) – verbose output None Get learning rates and their corresponding losses skip_start (int) – number of batches to trim from the start. skip_end (int) – number of batches to trim from the end. Tuple[list, list] Get learning rate which has steepest gradient and its corresponding loss skip_start (int) – number of batches to trim from the start. skip_end (int) – number of batches to trim from the end. Union[Tuple[float, float], Tuple[None, None]] Learning rate which has steepest gradient and its corresponding loss Plots the learning rate range test. skip_start (int) – number of batches to trim from the start. skip_end (int) – number of batches to trim from the start. log_lr (bool) – True to plot the learning rate in a logarithmic scale; otherwise, plotted in a linear scale. ax – the plot is created in the specified matplotlib axes object and the figure is not be shown. If None, then the figure and axes object are created in this method and the figure is

shown. steepest_lr (bool) – plot the learning rate which had the steepest gradient. The matplotlib.axes.Axes object that contains the plot. Returns None if matplotlib is not installed. Performs the learning rate range test. train_loader (DataLoader) – training set data loader. val_loader (Optional[DataLoader]) – validation data loader (if desired). image_extractor (Callable) – callable function to get the image from a batch of data. Default: x["image"] if isinstance(x, dict) else x[0]. label_extractor (Callable) – callable function to get the label from a batch of data. Default: x["label"] if isinstance(x, dict) else x[1]. start_lr (Optional[float]) – the starting learning rate for the range test. The default is the optimizer's learning rate. end_lr (int) – the maximum learning rate to test. The test may stop earlier than this if the result starts diverging. num_iter (int) – the max number of iterations for test. step_mode (str) – schedule for increasing learning rate: (linear or exp). smooth_f (float) – the loss smoothing factor within the [0, 1[interval. Disabled if set to 0, otherwise loss is smoothed using exponential smoothing. diverge_th (int) – test is stopped when loss surpasses threshold: diverge_th * best_loss. accumulation_steps (int) – steps for gradient accumulation. If set to 1, gradients are not accumulated. non_blocking_transfer (bool) – when True, moves data to device asynchronously if possible, e.g., moving CPU Tensors with pinned memory to CUDA devices. auto_reset (bool) – if True, returns model and optimizer to original states at end of test. None None Restores the model and optimizer to their initial states. None

Novograd#

Novograd based on Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks. The code is adapted from the implementations in Jasper for PyTorch, and OpenSeq2Seq. params (Iterable) – iterable of parameters to optimize or dicts defining parameter groups. lr (float) – learning rate. Defaults to 1e-3. betas (Tuple[float, float]) – coefficients used for computing running averages of gradient and its square. Defaults to (0.9, 0.98). eps (float) – term added to the denominator to improve numerical stability. Defaults to 1e-8. weight_decay (float) – weight decay (L2 penalty). Defaults to 0. grad_averaging (bool) – gradient averaging. Defaults to False. amsgrad (bool) – whether to use the AMSGrad variant of this algorithm from the paper On the Convergence of Adam and Beyond. Defaults to False. Performs a single optimization step. closure (Optional[Callable]) – A closure that reevaluates the model and returns the loss. Defaults to None.

Generate parameter groups#

Utility function to generate parameter groups with different LR values for optimizer. The output parameter groups have the same order as layer_match functions. network (Module) – source network to generate parameter groups from. layer_matches (Sequence[Callable]) – a list of callable functions to select or filter out network layer groups, for "select" type, the input will be the network, for "filter" type, the input will be every item of network.named_parameters(). for "select", the parameters will be select_func(network).parameters(). for "filter", the parameters will be (x[1] for x in filter(f, network.named_parameters())) match_types (Sequence[str]) – a list of tags to identify the matching type corresponding to the layer_matches functions, can be "select" or "filter". lr_values (Sequence[float]) – a list of LR values corresponding to the layer_matches functions. include_others (bool) – whether to include the rest layers as the last group, default to True. It's mainly used to set different LR values for different network elements, for example:

ExponentialLR#

Exponentially increases the learning rate between two boundaries over a number of iterations.

LinearLR#

Linearly increases the learning rate between two boundaries over a number of iterations.

WarmupCosineSchedule#

Linear warmup and then cosine decay. Based on https://huggingface.co/ implementation. optimizer (Optimizer) – wrapped optimizer. warmup_steps (int) – number of warmup iterations. t_total (int) – total number of training iterations. cycles (float) – cosine cycles parameter. last_epoch (int) – the index of last epoch. warmup_multiplier (float) – if provided, starts the linear warmup from this fraction of the initial lr. Must be in 0..1 interval. Defaults to 0 None previous Metrics next Data