```
O PyTorch
                                                                                                   Mobile
                                                        Get Started
                                                                              Ecosystem
                                                                                                                   Blog
                                                                                                                                Tutorials
                                                                                                                                                  Docs 🗸
                                                                                                                                                                     Resources ~
                                                                                                                                                                                              GitHub
                                                                                                                                                                                                >_
                                                         Docs > torch.optim > Adam
1.11.0 ▼
                                                         ADAM
  Q Search Docs
                                                          CLASS torch.optim.Adam(params, 1r=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0,
Notes [+]
                                                                   amsgrad=False, *, maximize=False) [SOURCE]
Language Bindings [+]
                                                                  Implements Adam algorithm.
Python API [-]
torch
                                                                              input: \gamma (lr), \beta_1, \beta_2 (betas), \theta_0 (params), f(\theta) (objective)
torch.nn
                                                                                          \lambda (weight decay), amsgrad, maximize
torch.nn.functional
                                                                              initialize: m_0 \leftarrow 0 (first moment), v_0 \leftarrow 0 (second moment), \widehat{v_0}^{max} \leftarrow 0
torch.Tensor
Tensor Attributes
                                                                              for t = 1 to ... do
Tensor Views
                                                                                   if maximize:
torch.autograd
                                                                                       g_t \leftarrow -
abla_{	heta} f_t(	heta_{t-1})
torch.cuda
torch.cuda.amp
                                                                                   else
torch.backends
                                                                                       g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})
torch.distributed
                                                                                   if \lambda \neq 0
torch.distributed.algorithms.join
                                                                                       g_t \leftarrow g_t + \lambda 	heta_{t-1}
torch.distributed.elastic
                                                                                   m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)g_t
torch.distributed.fsdp
                                                                                   v_t \leftarrow eta_2 v_{t-1} + (1-eta_2)g_t^2
torch.distributed.optim
                                                                                   \widehat{m_t} \leftarrow m_t/ig(1-eta_1^tig)
torch.distributions
                                                                                   \widehat{v_t} \leftarrow v_t / (1 - \beta_2^t)
torch.fft
torch.futures
                                                                                   {f if}\, amsgrad
torch.fx
                                                                                       \widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})
torch.hub
                                                                                       	heta_t \leftarrow 	heta_{t-1} - \gamma \widehat{m_t} / ig(\sqrt{\widehat{v_t}^{max}} + \epsilonig)
torch.jit
                                                                                   else
torch.linalg
                                                                                       	heta_t \leftarrow 	heta_{t-1} - \gamma \widehat{m_t} / ig(\sqrt{\widehat{v_t}} + \epsilonig)
torch.monitor
torch.special
                                                                              \mathbf{return} \ \theta_{\mathbf{t}}
torch.overrides
torch.package
torch.profiler
                                                                   For further details regarding the algorithm we refer to Adam: A Method for Stochastic Optimization.
                                                                             Parameters
                                                                              • params (iterable) – iterable of parameters to optimize or dicts defining parameter groups
                                                                              • Ir (float, optional) – learning rate (default: 1e-3)
                                                                              • betas (Tuple[float, float], optional) – coefficients used for computing running averages of gradient and its
                                                                                 square (default: (0.9, 0.999))
                                                                              • eps (float, optional) – term added to the denominator to improve numerical stability (default: 1e-8)
                                                                              • weight_decay (float, optional) – weight decay (L2 penalty) (default: 0)
                                                                              • amsgrad (boolean, optional) – whether to use the AMSGrad variant of this algorithm from the paper On
                                                                                 the Convergence of Adam and Beyond (default: False)
                                                                              • maximize (bool, optional) – maximize the params based on the objective, instead of minimizing (default:
                                                                                 False)
                                                                    add_param_group(param_group)
                                                                            Add a param group to the Optimizer s param_groups.
                                                                            This can be useful when fine tuning a pre-trained network as frozen layers can be made trainable and added to
                                                                            the Optimizer as training progresses.
                                                                              Parameters
                                                                                      param_group (dict) – Specifies what Tensors should be optimized along with group specific
                                                                                      optimization options.
                                                                    load_state_dict(state_dict)
                                                                            Loads the optimizer state.
                                                                              Parameters
                                                                                      state_dict (dict) – optimizer state. Should be an object returned from a call to state_dict().
                                                                    state_dict()
                                                                            Returns the state of the optimizer as a dict.
                                                                            It contains two entries:
                                                                                   state - a dict holding current optimization state. Its content
                                                                                           differs between optimizer classes.
                                                                                   param_groups - a list containing all parameter groups where each
                                                                                           parameter group is a dict
                                                                    step(closure=None) [SOURCE]
                                                                            Performs a single optimization step.
                                                                              Parameters
                                                                                      closure (callable, optional) - A closure that reevaluates the model and returns the loss.
                                                                    zero_grad(set_to_none=False)
                                                                            Sets the gradients of all optimized torch. Tensor s to zero.
                                                                              Parameters
                                                                                      set_to_none (bool) - instead of setting to zero, set the grads to None. This will in general have lower
                                                                                      memory footprint, and can modestly improve performance. However, it changes certain behaviors. For
                                                                                      example: 1. When the user tries to access a gradient and perform manual ops on it, a None attribute or
                                                                                      a Tensor full of 0s will behave differently. 2. If the user requests zero_grad(set_to_none=True)
                                                                                      followed by a backward pass, .grad s are guaranteed to be None for params that did not receive a
                                                                                      gradient. 3. torch.optim optimizers have a different behavior if the gradient is 0 or None (in one case
                                                                                      it does the step with a gradient of 0 and in the other it skips the step altogether).
                                                         Previous
                                                                                                                                                                                             Next >
```

Docs
Tutorials

Access comprehensive developer documentation for PyTorch

View Docs >

Tutorials

Get in-depth tutorials for beginners and advanced development resources and get your questions answered

View Tutorials >

View Tutorials >

View Resources

© Copyright 2019, Torch Contributors.

Built with Sphinx using a theme provided by Read the Docs.

PyTorch Resources

Get Started Tutorials

Features Docs

Ecosystem Discuss

Blog Github Issues

Brand Guidelines

 \rightarrow

Stay Connected

Email Address