Automatic Mixed Precision package - torch.amp

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 9)

Unknown directive type "py:module".

```
.. py:module:: torch.cpu
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 10)

Unknown directive type "py:module".

```
.. py:module:: torch.cpu.amp
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 11)

Unknown directive type "py:module".

```
.. py:module:: torch.cuda.amp
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 13)

Unknown directive type "automodule".

```
.. automodule:: torch.amp
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 14)

Unknown directive type "currentmodule".

```
.. currentmodule:: torch.amp
```

:class:`torch.amp` provides convenience methods for mixed precision, where some operations use the `torch.float32` (`float`) datatype and other operations use lower precision floating point datatype (`lower_precision_fp`): `torch.float16` (`half`) or `torch.bfloat16`. Some ops, like linear layers and convolutions, are much faster in `lower_precision_fp`. Other ops, like reductions, often require the dynamic range of `float32`. Mixed precision tries to match each op to its appropriate datatype.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 16); *backlink*

Unknown interpreted text role "class".

Ordinarily, "automatic mixed precision training" with datatype of `torch.float16` uses :class:`torch.autocast` and :class:`torch.cuda.amp.GradScaler` together, as shown in the :ref:`CUDA Automatic Mixed Precision examples<amp-examples>` and CUDA Automatic Mixed Precision recipe. However, :class:`torch.autocast` and :class:`torch.cuda.amp.GradScaler` are modular, and may be used separately if desired.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 22); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 22); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 22); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 22); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 22); *backlink*

Unknown interpreted text role "class".

For CUDA and CPU, APIs are also provided seperately:

- `torch.autocast("cuda", args...)` is equivalent to `torch.cuda.amp.autocast(args...)`.
- `torch.autocast("cpu", args...)` is equivalent to `torch.cpu.amp.autocast(args...)`. For CPU, only lower precision floating point datatype of `torch.bfloat16` is supported for now.

Autocasting

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 38)

Unknown directive type "currentmodule".

```
.. currentmodule:: torch
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 40)

Unknown directive type "autoclass".

```
.. autoclass:: autocast
    :members:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 43)

Unknown directive type "currentmodule".

```
.. currentmodule:: torch.cuda.amp
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 45)

Unknown directive type "autoclass".

```
.. autoclass:: autocast
    :members:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 48)

Unknown directive type "autofunction".

```
.. autofunction::  custom_fwd
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 50)

Unknown directive type "autofunction".

```
.. autofunction::  custom_bwd
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 52)

Unknown directive type "currentmodule".

```
.. currentmodule:: torch.cpu.amp
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 54)

Unknown directive type "autoclass".

```
.. autoclass:: autocast
    :members:
```

Gradient Scaling

If the forward pass for a particular op has `float16` inputs, the backward pass for that op will produce `float16` gradients. Gradient values with small magnitudes may not be representable in `float16`. These values will flush to zero ("underflow"), so the update for the corresponding parameters will be lost.

To prevent underflow, "gradient scaling" multiplies the network's loss(es) by a scale factor and invokes a backward pass on the scaled loss(es). Gradients flowing backward through the network are then scaled by the same factor. In other words, gradient values have a larger magnitude, so they don't flush to zero.

Each parameter's gradient (`.grad` attribute) should be unscaled before the optimizer updates the parameters, so the scale factor does not interfere with the learning rate.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 75)

Unknown directive type "currentmodule".

```
.. currentmodule:: torch.cuda.amp
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 77)

Unknown directive type "autoclass".

```
.. autoclass:: GradScaler
    :members:
```

Autocast Op Reference

Op Eligibility

Ops that run in `float64` or non-floating-point dtypes are not eligible, and will run in these types whether or not autocast is enabled.

Only out-of-place ops and Tensor methods are eligible. In-place variants and calls that explicitly supply an `out=...` Tensor are

allowed in autocast-enabled regions, but won't go through autocasting. For example, in an autocast-enabled region `a.addmm(b, c)` can autocast, but `a.addmm_(b, c)` and `a.addmm(b, c, out=d)` cannot. For best performance and stability, prefer out-of-place ops in autocast-enabled regions.

Ops called with an explicit `dtype=...` argument are not eligible, and will produce output that respects the `dtype` argument.

CUDA Op-Specific Behavior

The following lists describe the behavior of eligible ops in autocast-enabled regions. These ops always go through autocasting whether they are invoked as part of a :class:`torch.nn.Module`, as a function, or as a :class:`torch.Tensor` method. If functions are exposed in multiple namespaces, they go through autocasting regardless of the namespace.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 107); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 107); *backlink*

Unknown interpreted text role "class".

Ops not listed below do not go through autocasting. They run in the type defined by their inputs. However, autocasting may still change the type in which unlisted ops run if they're downstream from autocasted ops.

If an op is unlisted, we assume it's numerically stable in `float16`. If you believe an unlisted op is numerically unstable in `float16`, please file an issue.

CUDA Ops that can autocast to `float16`

`__matmul__`, `addbmm`, `addmm`, `addmv`, `addr`, `baddbmm`, `bmm`, `chain_matmul`, `multi_dot`, `conv1d`, `conv2d`, `conv3d`, `conv_transpose1d`, `conv_transpose2d`, `conv_transpose3d`, `GRUCell`, `linear`, `LSTMCell`, `matmul`, `mm`, `mv`, `prelu`, `RNNCell`

CUDA Ops that can autocast to `float32`

`__pow__`, `__rdiv__`, `__rpow__`, `__rtruediv__`, `acos`, `asin`, `binary_cross_entropy_with_logits`, `cosh`, `cosine_embedding_loss`, `cdist`, `cosine_similarity`, `cross_entropy`, `cumprod`, `cumsum`, `dist`, `erfinv`, `exp`, `expm1`, `group_norm`, `hinge_embedding_loss`, `kl_div`, `l1_loss`, `layer_norm`, `log`, `log_softmax`, `log10`, `log1p`, `log2`, `margin_ranking_loss`, `mse_loss`, `multilabel_margin_loss`, `multi_margin_loss`, `nll_loss`, `norm`, `normalize`, `pdist`, `poisson_nll_loss`, `pow`, `prod`, `reciprocal`, `rsqrt`, `sinh`, `smooth_l1_loss`, `soft_margin_loss`, `softmax`, `softmin`, `softplus`, `sum`, `renorm`, `tan`, `triplet_margin_loss`

CUDA Ops that promote to the widest input type

These ops don't require a particular dtype for stability, but take multiple inputs and require that the inputs' dtypes match. If all of the inputs are `float16`, the op runs in `float16`. If any of the inputs is `float32`, autocast casts all inputs to `float32` and runs the op in `float32`.

`addcdiv`, `addcmul`, `atan2`, `bilinear`, `cross`, `dot`, `grid_sample`, `index_put`, `scatter_add`, `tensordot`

Some ops not listed here (e.g., binary ops like `add`) natively promote inputs without autocasting's intervention. If inputs are a mixture of `float16` and `float32`, these ops run in `float32` and produce `float32` output, regardless of whether autocast is enabled.

Prefer `binary_cross_entropy_with_logits` over `binary_cross_entropy`

The backward passes of :func:`torch.nn.functional.binary_cross_entropy` (and :mod:`torch.nn.BCELoss`, which wraps it) can produce gradients that aren't representable in `float16`. In autocast-enabled regions, the forward input may be `float16`, which means the backward gradient must be representable in `float16` (autocasting `float16` forward inputs to `float32` doesn't help, because that cast must be reversed in backward). Therefore, `binary_cross_entropy` and `BCELoss` raise an error in autocast-enabled regions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 227); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 227); *backlink*

Unknown interpreted text role "mod".

Many models use a sigmoid layer right before the binary cross entropy layer. In this case, combine the two layers using :func:`torch.nn.functional.binary_cross_entropy_with_logits` or :mod:`torch.nn.BCEWithLogitsLoss`. `binary_cross_entropy_with_logits` and `BCEWithLogits` are safe to autocast.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 233); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 233); *backlink*

Unknown interpreted text role "mod".

CPU Op-Specific Behavior

The following lists describe the behavior of eligible ops in autocast-enabled regions. These ops always go through autocasting whether they are invoked as part of a :class:`torch.nn.Module`, as a function, or as a :class:`torch.Tensor` method. If functions are exposed in multiple namespaces, they go through autocasting regardless of the namespace.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 242); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]amp.rst, line 242); *backlink*

Unknown interpreted text role "class".

Ops not listed below do not go through autocasting. They run in the type defined by their inputs. However, autocasting may still change the type in which unlisted ops run if they're downstream from autocasted ops.

If an op is unlisted, we assume it's numerically stable in `bfloat16`. If you believe an unlisted op is numerically unstable in `bfloat16`, please file an issue.

CPU Ops that can autocast to `bfloat16`

`conv1d`, `conv2d`, `conv3d`, `bmm`, `mm`, `baddbmm`, `addmm`, `addbmm`, `linear`, `_convolution`

CPU Ops that can autocast to `float32`

`conv_transpose1d`, `conv_transpose2d`, `conv_transpose3d`, `batch_norm`, `dropout`, `avg_pool1d`, `avg_pool2d`, `avg_pool3d`, `gelu`, `upsample_nearest1d`, `_upsample_nearest_exact1d`, `upsample_nearest2d`, `_upsample_nearest_exact2d`, `upsample_nearest3d`, `_upsample_nearest_exact3d`, `upsample_linear1d`, `upsample_bilinear2d`, `upsample_trilinear3d`, `binary_cross_entropy`, `binary_cross_entropy_with_logits`, `instance_norm`, `grid_sampler`, `polar`, `multinomial`, `poisson`, `fmod`, `prod`, `quantile`, `nanquantile`, `stft`, `cdist`, `cross`, `cumprod`, `cumsum`, `diag`, `diagflat`, `histc`, `logcumsumexp`, `searchsorted`, `trace`, `tril`, `triu`, `vander`, `view_as_complex`, `cholesky`, `cholesky_inverse`, `cholesky_solve`, `dot`, `inverse`, `lu_solve`, `matrix_rank`, `orgqr`, `inverse`, `ormqr`, `pinverse`, `vdot`, `im2col`, `col2im`, `max_pool3d`, `max_unpool2d`, `max_unpool3d`, `adaptive_avg_pool3d`, `reflection_pad1d`, `reflection_pad2d`, `replication_pad1d`, `replication_pad2d`, `replication_pad3d`, `elu`, `hardshrink`, `hardsigmoid`, `hardswish`, `log_sigmoid`, `prelu`, `selu`, `celu`, `softplus`, `softshrink`, `group_norm`, `smooth_l1_loss`, `mse_loss`, `ctc_loss`, `kl_div`, `multilabel_margin_loss`, `fft_fft`, `fft_ifft`, `fft_fft2`, `fft_ifft2`, `fft_fftn`, `fft_ifftn`, `fft_rfft`, `fft_irfft`, `fft_rfft2`, `fft_irfft2`, `fft_rfftn`, `fft_irfftn`, `fft_hfft`, `fft_ihfft`, `conv_tbc`, `linalg_matrix_norm`, `linalg_cond`, `linalg_matrix_rank`, `linalg_solve`, `linalg_cholesky`, `linalg_svdvals`, `linalg_eigvals`, `linalg_eigvalsh`, `linalg_inv`, `linalg_householder_product`, `linalg_tensorinv`, `linalg_tensorsolve`, `fake_quantize_per_tensor_affine`, `glu`, `cummax`, `cummin`, `eig`, `geqrf`, `lstsq`, `_lu_with_info`, `lu_unpack`, `qr`, `solve`, `svd`, `symeig`, `triangular_solve`, `fractional_max_pool2d`, `fractional_max_pool3d`, `adaptive_max_pool1d`, `adaptive_max_pool2d`, `adaptive_max_pool3d`, `multilabel_margin_loss_forward`, `linalg_qr`, `linalg_cholesky_ex`, `linalg_svd`, `linalg_eig`, `linalg_eigh`, `linalg_lstsq`, `linalg_inv_ex`

CPU Ops that promote to the widest input type

These ops don't require a particular dtype for stability, but take multiple inputs and require that the inputs' dtypes match. If all of the

inputs are `bfloat16`, the op runs in `bfloat16`. If any of the inputs is `float32`, autocast casts all inputs to `float32` and runs the op in `float32`.

`cat`, `stack`, `index_copy`

Some ops not listed here (e.g., binary ops like `add`) natively promote inputs without autocasting's intervention. If inputs are a mixture of `bfloat16` and `float32`, these ops run in `float32` and produce `float32` output, regardless of whether autocast is enabled.