Supplementary Material for Attentive Normalization

Anonymous ECCV submission

Paper ID 2668

In the supplementary material, we provide the PyTorch source code for implementing our proposed Attentive Normalization. The complete source codes for reproducing all the ImageNet experiments and COCO experiments are provided too. Please follow the readme in each code package. Due to space limit, we can not upload the pre-trained models, which will be made publicly available.

We note that our codes are built on several existing code bases: the vanilla AOGNets [2], different networks implemented in TorchVision, the MMDetection package [1], and the NVIDIA APEX library. We will properly credit those code bases in our public release of the code.

```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

import torch
import torch.nn as nn
import torch.nn.functional as F

_inplace = True
_norm_eps = 1e-5


### hsigmoid
class hsigmoid(nn.Module):
    def forward(self, x):
        out = F.relu6(x + 3, inplace=True) / 6
        return out


### Feature Norm
def FeatureNorm(norm_name, num_channels, num_groups, num_k, attention_mode):
    if norm_name == "BatchNorm2d":
        return nn.BatchNorm2d(num_channels, eps=_norm_eps)
    elif norm_name == "GroupNorm":
        assert num_groups > 1
        if num_channels % num_groups != 0:
            raise ValueError("channels {} not dividable by groups {}".format(
                                num_channels, num_groups))
        return nn.GroupNorm(num_channels, num_groups, eps=_norm_eps)
    elif norm_name == "AttentiveBatchNorm2d":
        assert num_k > 1
        return AttentiveBatchNorm2d(num_channels, num_k, attention_mode)
    elif norm_name == "AttentiveGroupNorm":
        assert num_groups > 1 and num_k > 1
        if num_channels % num_groups != 0:
            raise ValueError("channels {} not dividable by groups {}".format(
                                num_channels, num_groups))
        return AttentiveGroupNorm(num_channels, num_groups, num_k,
                                  attention_mode)
    else:
        raise NotImplementedError("Unknown feature norm name")

```

```python

### Attention weights for attentive norm
class AttentionWeights(nn.Module):
    expansion = 2
    def __init__(self, attention_mode, num_channels, k,
                norm_name=None, norm_groups=0):
        super(AttentionWeights, self).__init__()
        self.k = k
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        layers = []
        if attention_mode == 0:
            layers = [ nn.Conv2d(num_channels, k, 1),
                        nn.Sigmoid() ]
        elif attention_mode == 4:
            layers = [ nn.Conv2d(num_channels, k, 1),
                        hsigmoid() ]
        elif attention_mode == 1:
            layers = [ nn.Conv2d(num_channels, k*self.expansion, 1),
                        nn.ReLU(inplace=True),
                        nn.Conv2d(k*self.expansion, k, 1),
                        nn.Sigmoid() ]
        elif attention_mode == 2: # best, others for ablation study
            assert norm_name is not None
            layers = [ nn.Conv2d(num_channels, k, 1, bias=False),
                        FeatureNorm(norm_name, k, norm_groups, 0, 0),
                        hsigmoid() ]
        elif attention_mode == 5:
            assert norm_name is not None
            layers = [ nn.Conv2d(num_channels, k, 1, bias=False),
                        FeatureNorm(norm_name, k, norm_groups, 0, 0),
                        nn.Sigmoid() ]
        elif attention_mode == 6:
            assert norm_name is not None
            layers = [ nn.Conv2d(num_channels, k, 1, bias=False),
                        FeatureNorm(norm_name, k, norm_groups, 0, 0),
                        nn.Softmax(dim=1) ]
        elif attention_mode == 3:
            assert norm_name is not None
            layers = [ nn.Conv2d(num_channels, k*self.expansion, 1, bias=False),
                       FeatureNorm(norm_name, k*self.expansion, norm_groups, 0,
                                    0),
                       nn.ReLU(inplace=True),
                       nn.Conv2d(k*self.expansion, k, 1, bias=False),
                       FeatureNorm(norm_name, k, norm_groups, 0, 0),
                      hsigmoid() ]
        else:
            raise NotImplementedError("Unknow attention weight type")
        self.attention = nn.Sequential(*layers)

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avgpool(x)
        var = torch.var(x, dim=(2, 3)).view(b, c, 1, 1)
        y *= (var + 1e-3).rsqrt() # RSD
        #y = torch.cat((y, var), dim=1) # for ablation study
        return self.attention(y).view(b, self.k)


### AN w/ BN
class AttentiveBatchNorm2d(nn.BatchNorm2d):
    def __init__(self, num_channels, k, attention_mode,
                 eps=_norm_eps, momentum=0.1, track_running_stats=True):
        super(AttentiveBatchNorm2d, self).__init__(num_channels, eps=eps,
            momentum=momentum, affine=False,
            track_running_stats=track_running_stats)
        self.k = k
```

```python
        self.weight_ = nn.Parameter(torch.Tensor(k, num_channels))
        self.bias_ = nn.Parameter(torch.Tensor(k, num_channels))

        self.attention_weights = AttentionWeights(attention_mode, num_channels,
                                                  k, norm_name='BatchNorm2d')

        self._init_params()

    def _init_params(self):
        nn.init.normal_(self.weight_, 1, 0.1)
        nn.init.normal_(self.bias_, 0, 0.1)

    def forward(self, x):
        output = super(AttentiveBatchNorm2d, self).forward(x)
        size = output.size()
        y = self.attention_weights(x) # bxk

        weight = y @ self.weight_ # bxc
        bias = y @ self.bias_ # bxc
        weight = weight.unsqueeze(-1).unsqueeze(-1).expand(size)
        bias = bias.unsqueeze(-1).unsqueeze(-1).expand(size)

        return weight * output + bias


# AN w/ GN
class AttentiveGroupNorm(nn.Module):
    __constants__ = ['num_groups', 'num_channels', 'k', 'eps', 'weight',
                     'bias']

    def __init__(self, num_channels, num_groups, k, attention_mode,
                 eps=_norm_eps):
        super(AttentiveGroupNorm, self).__init__()
        self.num_groups = num_groups
        self.num_channels = num_channels
        self.k = k
        self.eps = eps
        self.affine = True
        self.weight_ = nn.Parameter(torch.Tensor(k, num_channels))
        self.bias_ = nn.Parameter(torch.Tensor(k, num_channels))
        self.register_parameter('weight', None)
        self.register_parameter('bias', None)

        self.attention_weights = AttentionWeights(attention_mode, num_channels,
                                                  k,
                                                  norm_name='GroupNorm', norm_groups=1)

        self.reset_parameters()

    def reset_parameters(self):
        nn.init.normal_(self.weight_, 1, 0.1)
        nn.init.normal_(self.bias_, 0, 0.1)

    def forward(self, x):
        output = F.group_norm(
            x, self.num_groups, self.weight, self.bias, self.eps)
        size = output.size()

        y = self.attention_weights(x)

        weight = y @ self.weight_
        bias = y @ self.bias_

        weight = weight.unsqueeze(-1).unsqueeze(-1).expand(size)
        bias = bias.unsqueeze(-1).unsqueeze(-1).expand(size)
```

```
        return weight * output + bias

    def extra_repr(self):
        return '{num_groups}, {num_channels}, eps={eps}, ' \
            'affine={affine}'.format(**self.__dict__)
```

References

1. Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C.C., Lin, D.: MMDetection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:1906.07155 (2019)
2. Li, X., Song, X., Wu, T.: Aognets: Compositional grammatical architectures for deep learning. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. pp. 6220–6230 (2019)