UVA CS 6316: Machine Learning : 2019 Fall
Course Project: Deep2Reproduce @
https://github.com/qiyanjun/deep2reproduce/tree/master/2019Fall

Norm Matters: Efficient and Accurate Normalization Schemes in Deep Networks

https://arxiv.org/abs/1803.01814

Machine Teachers

12/06/2019

Reproduced by : Akhil Sai, Aniruddha Dave, Hemanth Kumar, Zhe Lou

Background

- Normalization, a pre-processing step, transforms inputs to have zero-mean and unit variance
- Normalization between layers is widely used in Deep neural networks, which speeds up learning and improves accuracy
- Batch Normalization (BN) normalizes each layer to have zero mean and unit variance for each channel across training batch
- Normalization can also be applied to layer parameters instead of outputs. Ex: Weight Normalization, Normalization Propagation

Motivation

Issues with current normalization methods:

- **Interplay With Other Regularization Mechanisms**
  - Unclear how weight decay interacts with BN
  - Is weight decay necessary?BN already constrains the output norms
- **Task-Specific Limitations**
  - BN **assumes** that samples appearing in each batch are independent
- **Computational Cost**
  - BN requires large computation cost and is not easily parallelized
  - Other methods have a much smaller computational cost but typically achieve lower accuracy
- **Numerical Precision**
  - BN is not adaptive to low-precision implementation

Related Work

- Understanding deep learning requires rethinking generalization [1]
  - Explicit regularization (weight decay) may improve generalization performance
  - It is not necessary or sufficient to reduce generalization error
- Recurrent batch normalization [2]
  - Alternatives like weight-normalization and layer-normalization are explicitly devised for BN but have not reached the success and wide adoption of BN

Related Work

- Comparison of batch normalization and weight normalization algorithms for the large-scale image classification [5]
  - BN constitutes up to 24% of the computation time needed for the entire model
- In Advances in neural information processing systems [6]
  - As the use of deep learning continues to evolve, the interest in low-precision training and inference increases

Claim / Target Task

- Devising relation between step-size, weight decay, learning rate and normalization
- Weight Decay affects the training process only indirectly, by modulating the learning rate
- Alternative normalization metrics which reduce computational overhead, retaining accuracy
- By using L^1 normalization, batch normalization can be quantified to half precision with no effect on validation accuracy
- Usage of L^∞ BN or Top (k) relaxation lowers the extent of reduction operation, helping low precision implementations
- Better, improved Weight Normalization technique for large scale implementations

Intuitive Figure Showing WHY Claim

Parameter	**L^2 Norm**	**L^1 & L^∞ Norms**
Computational Cost	HIGH	LOW
Memory Requirement	HIGH	LOW
Run Time	SLOW	FAST
Low Precision Applications	Low Accuracy	High Accuracy

Proposed Solution

- Identify the relation between the step size of the weight direction, learning rate and normalization to withhold the scale invariance of linear and nonlinear functions
- Maintain the accuracy, without using Weight Decay, only by adjusting the learning rate
- Replace the L^2 norm with scale-invariant alternatives (L^1, L^∞) which are more appealing computationally and can cater to low-precision implementations
- Use norm bounding to improve the performance and sustainability of weight normalization in large scale usage

Implementation

Mimicking effective step size with learning rate correction

$$\hat{\eta}_{\text{Correction}} = \eta \frac{\|w\|_2^2}{\|w_{[\text{WD on}]}\|_2^2}$$

Linf batch norm

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu^k}{C_{L_\infty}(n) \cdot \|x^{(k)} - \mu^k\|_\infty},$$

$$\text{Top}(k) = \frac{1}{k}\sum_{n=1}^{k} |s_n|$$

L1 batch norm

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu^k}{C_{L_1} \cdot \|x^{(k)} - \mu^k\|_1 / n}$$

$C_{L_1} = \sqrt{\pi/2}$ is a normalization term

Norm bounded Lp weight normalization

$$w_i = \rho \frac{v_i}{\|v_i\|_p}, \quad \rho = \|V\|_p^{(t=0)} / N^{1/p}$$

Implementation

- **Connection between weight-decay, learning rate and normalization**
  - Several experiments done on CIFAR-10 with adjusted LR to observe this connection
  - Learning rate scheduling replaced by norm scheduling by normalizing the norm of each convolution layer channel to emulate the norm of corresponding channel in training with WD and fixed learning rate.
- **Alternative L^p metrics for batch norm (L^1 batch norm)**
  - Added $C_{L1} = \sqrt{\pi/2}$ as a normalization term which is then implemented on ResNet-18 and ResNet-50 on ImageNet to compare the validation accuracy of L_1 and L_2 batch norms.
  - Verified L_1 layer normalization on the Transformer architecture of the WMT14 dataset.

Implementation

- **L^∞ batch norm**
  - Defined Top(k) to replace maximum absolute deviation with the mean of ten largest deviations for robustness to outliers.
  - Top(k) generalizes L^1 and L^∞ metrics
  - L^∞ is Top(1); L^1 is to Top(n)

- **Norm bounded weight-normalization**
  - If the norm is fixed, the weight's norm can be made completely disjoint from its values
  - Introduce 'ρ' - a fixed scalar for each layer, that is determined by its size (number of input and output channels)
  - Compute results on Imagenet using ResNet50

Data Summary

- **CIFAR-10**

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

- **WMT14 de-en**

WMT14 is a German-English dataset primarily used for the machine translational task with data taken from version 7 of Europarl corpus.

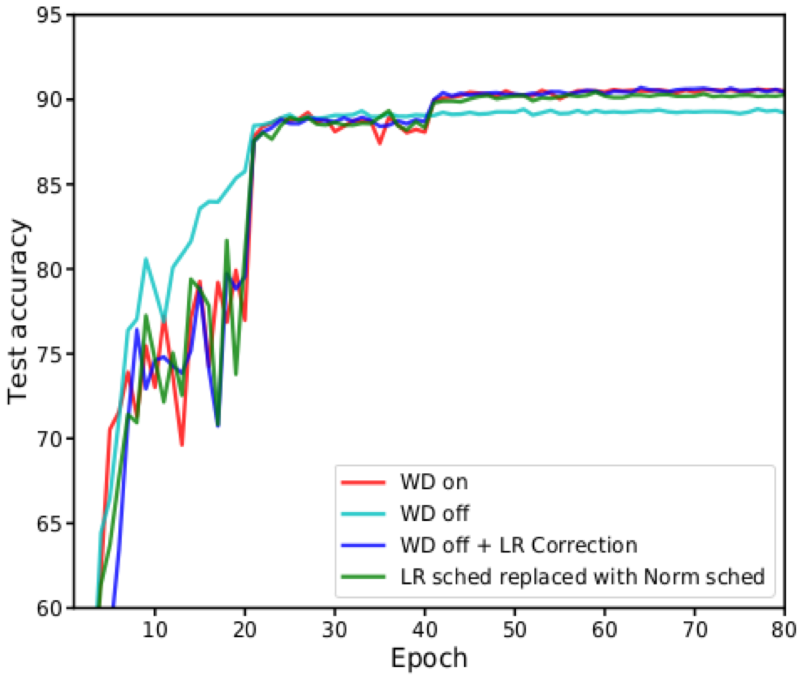
- **ImageNet**

ImageNet is a large visual database for visual object recognition with more than 14 million images hand-annotated and at least 1 million with bounding boxes.

Author's Experimental Results

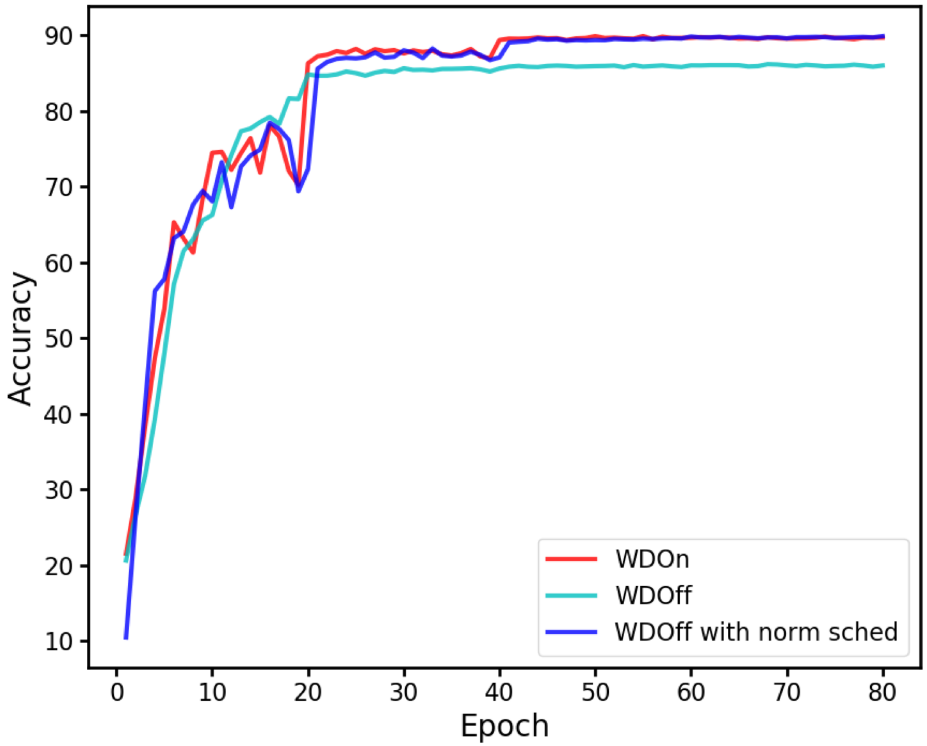
Connection between weight-decay, learning rate and normalization

$$\hat{\eta}_{\text{Correction}} = \eta \frac{\|w\|_2^2}{\|w_{[\text{WD on}]}\|_2^2}$$



- Applied correction on step-size
- Replace learning rate scheduling with norm scheduling
- Accuracy for both is similar to training with WD

Our Results
Effect of Weight Decay



- Results for three experiments as observed in the paper
- Accuracy with Norm scheduling and without Weight Decay is similar to the accuracy with Weight Decay
- WDoff with LR correction results not available as loss turns out to be ‘nan’

Our Results
Effect of Weight Decay: Code

```python
# [WD on] Regular WD on
def sgd_wd0_0005_lr0_1_momentum0_9(model, **kwargs):
    all_params = [{'params': params, 'name': name} for l, (name, params) in enumerate(model.named_parameters())]
    return optim.SGD(all_params, momentum=0.9, lr=0.1, weight_decay=5e-4)
```

```python
# [WD off] - Last layer wd on, rest off
def sgd_lastlayerwd0_0005_otherlayerswd0_lr0_1_momentum0_9(model, **kwargs):
    lastlayer_params = [{'params': params, 'name': name, 'weight_decay': 5e-4}
                        for l, (name, params) in enumerate(model.named_parameters()) if "lastlayer" in name]
    notlastlayer_params = [{'params': params, 'name': name, 'weight_decay': 0}
                           for l, (name, params) in enumerate(model.named_parameters()) if not ("lastlayer" in name)]
    all_params = lastlayer_params + notlastlayer_params
    return optim.SGD(all_params, momentum=0.9, lr=0.1, weight_decay=0.0)
```

Our Results
Effect of Weight Decay: Code

```python
# [WD off + correction] - Step size correction
def sgd_lastlayerwd0_0005_otherlayerswd0_lr0_1_with_correction(model, **kwargs):
    conv_layers_indices = get_model(model).get_conv_indices_set()
    lastlayer_params = [{'params': params, 'name': name, 'weight_decay': 5e-4}
                        for l, (name, params) in enumerate(model.named_parameters()) if "lastlayer" in name]
    notconv_notlastlayer_params = [{'params': params, 'name': name, 'weight_decay': 0}
                                   for l, (name, params) in enumerate(model.named_parameters())
                                   if (not ("lastlayer" in name)) and l not in conv_layers_indices]
    convlayer_params = [{'params': params, 'name': name, 'weight_decay': 0, 'l_idx': l, 'wd_norms': None}
                        for l, (name, params) in enumerate(model.named_parameters())
                        if (not ("lastlayer" in name)) and l in conv_layers_indices]

    all_params = lastlayer_params + notconv_notlastlayer_params + convlayer_params
    return SGDWDMimic(all_params, momentum=0.9, lr=0.1, weight_decay=0.0)
```

```python
# WD mimic+norm scheduling instead of lr scheduling
def sgd_lastlayerwd0_0005_otherlayerswd0_lr0_1_norm_sched_instead(model, **kwargs):
    conv_layers_indices = get_model(model).get_conv_indices_set()
    lastlayer_params = [{'params': params, 'name': name, 'weight_decay': 5e-4}
                        for l, (name, params) in enumerate(model.named_parameters()) if "lastlayer" in name]
    notconv_notlastlayer_params = [{'params': params, 'name': name, 'weight_decay': 0}
                                   for l, (name, params) in enumerate(model.named_parameters())
                                   if (not ("lastlayer" in name)) and l not in conv_layers_indices]
    convlayer_params = [{'params': params, 'name': name, 'weight_decay': 0, 'l_idx': l, 'wd_norms': None}
                        for l, (name, params) in enumerate(model.named_parameters())
                        if (not ("lastlayer" in name)) and l in conv_layers_indices]

    all_params = lastlayer_params + notconv_notlastlayer_params + convlayer_params
    return SGDWDMimicNormSchedInsteadLR(all_params, momentum=0.9, lr=0.1, weight_decay=0.0)
```

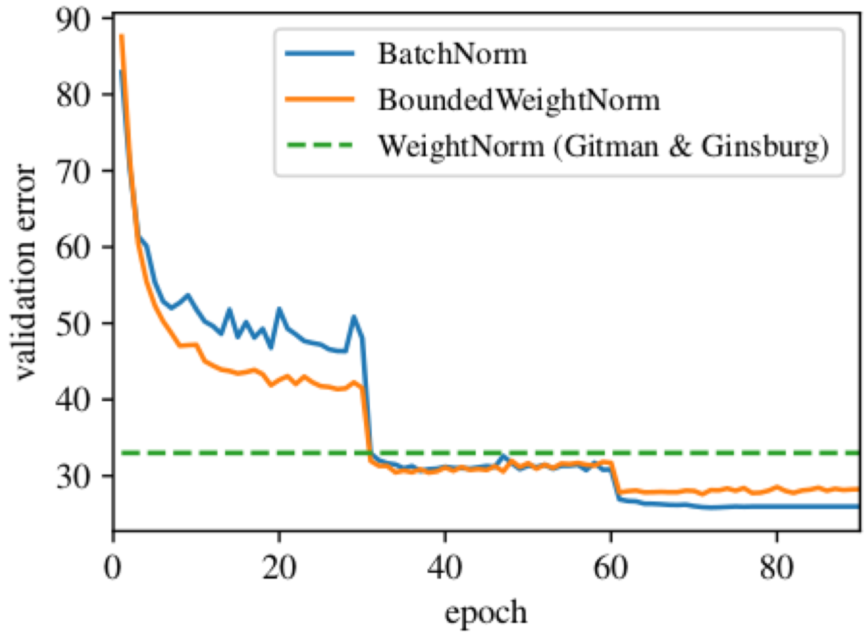
Author's Experimental Results

Results comparing baseline, L^2 based norm with weight norm and bounded weight norm

Network	Batch/Layer norm	WN	BWN
ResNet56 (Cifar10)	93.03%	92.5%	92.88%
ResNet50 (ImageNet)	75.3%	67% [11]	73.8%
Transformer (WMT14)	27.3 BLEU	-	26.5 BLEU
2-layer LSTM (WMT14)	21.5 BLEU	-	21.2 BLEU

Author’s Experimental Results

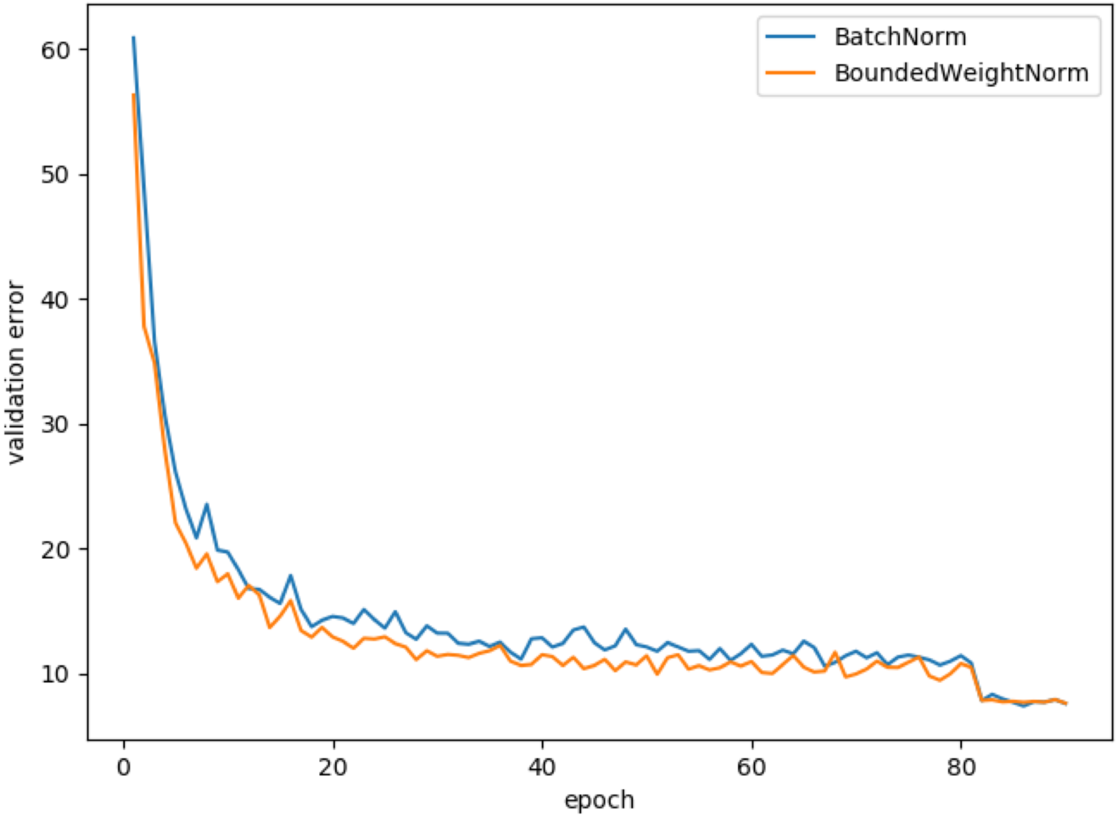
Norm Bounded Weight Normalization



- Final Accuracy:
  - BN: 75.3%
  - WN: 67%
  - BWN: 73.8%

*For ImageNet. WN did not converge. Similar issues were reported by [5]

Our Results
Norm bounded Weight Normalization



Our Results
L2 BN and BWN

Norm	Val. accuracy**	Val. loss
L2 Batch Norm	92.40	0.409
Bounded Weight Norm	92.37	0.428

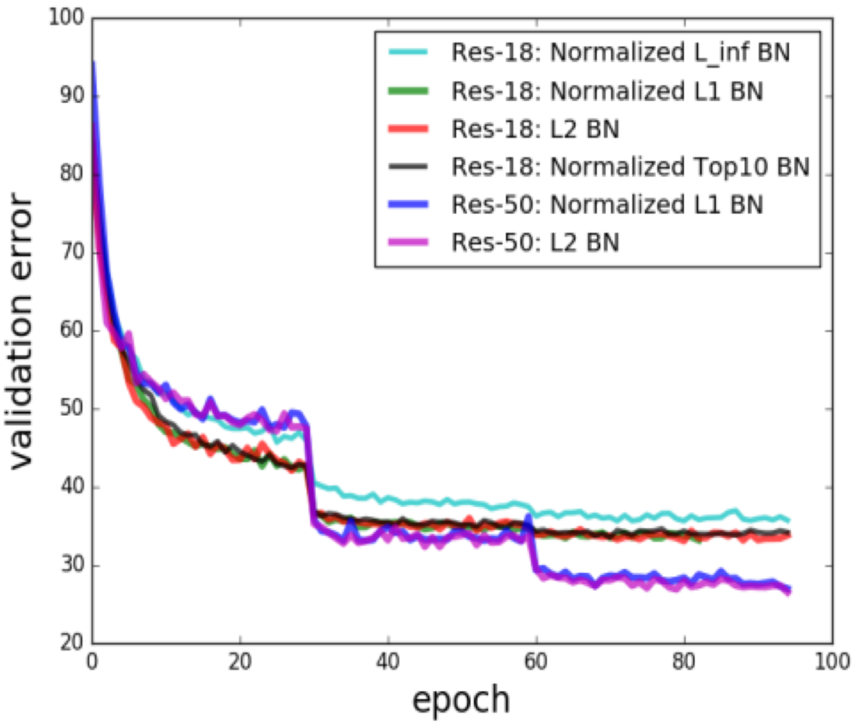
Code

Bounded Weight Norm

```
class BoundedWeighNorm(object):

    def __init__(self, name, dim, p):
        self.name = name
        self.dim = dim
        self.p = p

    def compute_weight(self, module):
        g = getattr(module, self.name + '_g')
        v = getattr(module, self.name + '_v')
        pre_norm = getattr(module, self.name + '_g_prenorm')
        norm = g.norm()
        g = (Variable(pre_norm) / norm) * g
        return v * (g / _norm(v, self.dim, p=self.p))

    @staticmethod
    def apply(module, name, dim, p):
        fn = BoundedWeighNorm(name, dim, p)

        weight = getattr(module, name)

        # remove w from parameter list
        del module._parameters[name]

        # add g and v as new parameters and express w as g/||v|| * v
        module.register_parameter(
            name + '_g', Parameter(_norm(weight, dim, p=p).data))

        g = getattr(module, name + '_g')
        module.register_buffer(
            name + '_g_prenorm', torch.Tensor([g.data.norm()]))
        pre_norm = getattr(module, name + '_g_prenorm')
        print(pre_norm)
        module.register_parameter(name + '_v', Parameter(weight.data))
        setattr(module, name, fn.compute_weight(module))

        # recompute weight before every forward()
        module.register_forward_pre_hook(fn)

        def gather_normed_params(self, memo=None, param_func=lambda s: fn.compute_weight(s)):
            return gather_params(self, memo, param_func)
        module.gather_params = gather_normed_params
        return fn
```

Author's Experimental Results

Alternative L^p metrics for BN



- Baseline: L2 normalization
- Results for ResNet18 and ResNet50
- L1 normalization reached a similar final accuracy
- L^∞ had a slightly lower accuracy

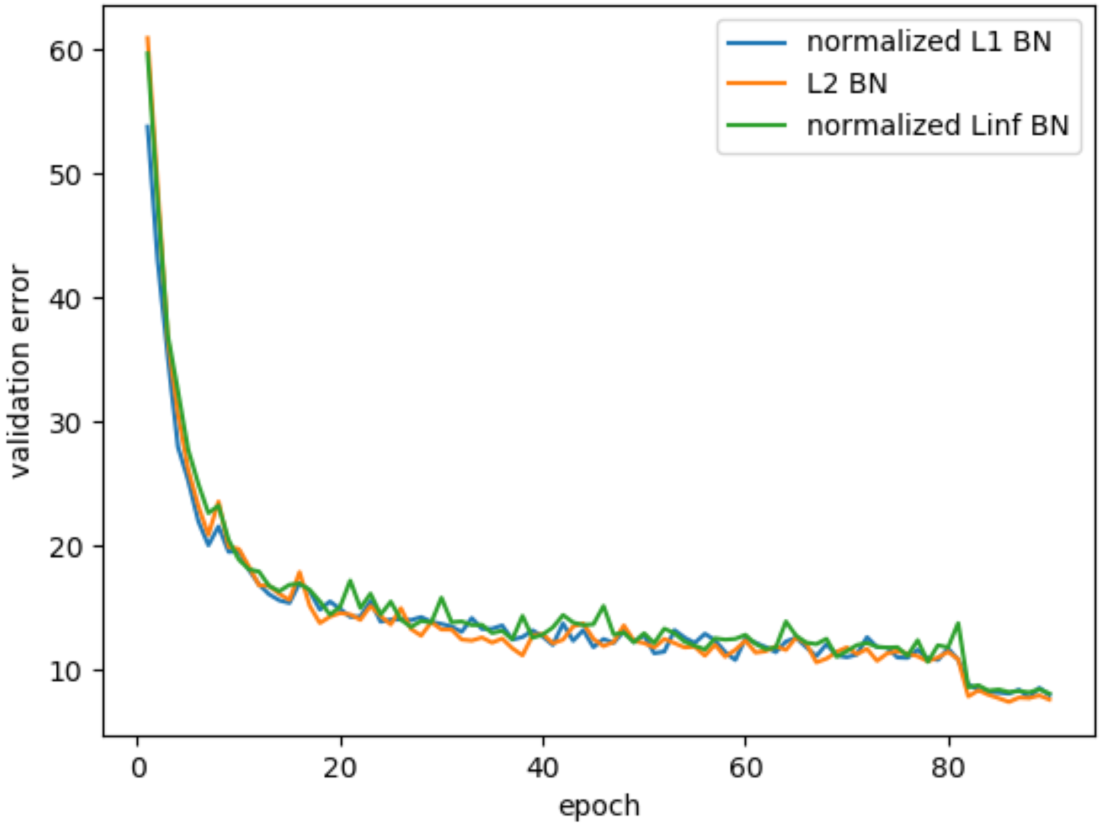
Author's Experimental Results

Results comparing baseline and L^1 norm results

Network	L^2 Batch/Layer norm	L^1 Batch/Layer norm
ResNet56 (Cifar10)	93.03%	93.07%
ResNet18 (ImageNet)	69.8%	69.74%
ResNet50 (ImageNet)	75.3%	75.32%
Transformer (WMT14)	5.1 ppl	5.2 ppl

Our Results

Alternative Lp metrics for BN

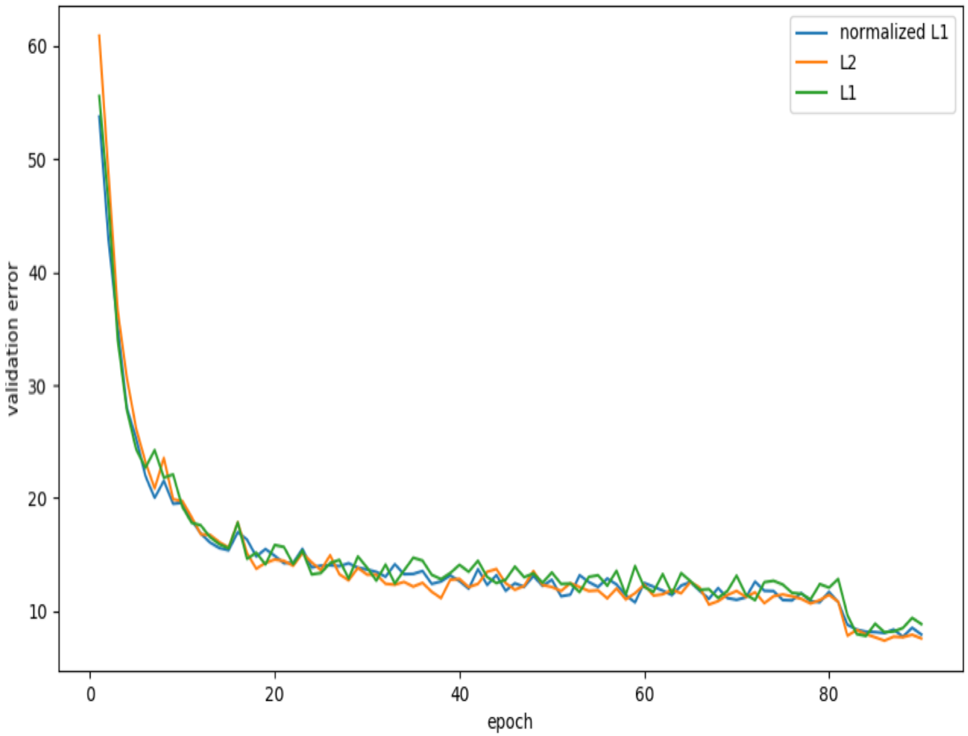


Our Results
Comparison between Norms

Norm	Execution Time*	Val. results**	Val. loss
Normalized L_{inf} Norm	85 mins	91.91	1.758
Normalized L1 Norm	83 mins	92.05	0.912
L2 Batch Norm	93 mins	92.40	0.409

* - On GPU at CS SLURM Nodes
** - For CIFAR-10 on ResNet-56

Our Results
Importance of Normalization Constants



The importance of normalization term C_{L1} while training ResNet-56 on CIFAR-10. Without the use of C_{L1} the network reaches a higher final validation error.

Code

L1 Batch Norm

```
def __init__(self, num_features, dim=1, momentum=0.1, bias=True, normalized=True, eps=1e-5, noise=False):
    super(L1BatchNorm2d, self).__init__()
    self.register_buffer('running_mean', torch.zeros(num_features))
    self.register_buffer('running_var', torch.zeros(num_features))
    self.momentum = momentum
    self.dim = dim
    self.noise = noise
    self.mean = Parameter(torch.Tensor(num_features))
    self.scale = Parameter(torch.Tensor(num_features))
    self.eps = eps
    if normalized:
        self.scale_fix = (np.pi / 2) ** 0.5
    else:
        self.scale_fix = 1

def forward(self, x):
    p = 1
    if self.training:
        mean = x.view(x.size(0), x.size(self.dim), -1).mean(-1).mean(0)
        y = x.transpose(0, 1)
        z = y.contiguous()
        t = z.view(z.size(0), -1)
        Var = (torch.abs((t.transpose(1, 0) - mean))).mean(0)
        scale = (Var * self.scale_fix + self.eps) ** (-1)
        self.running_mean.mul_(self.momentum).add_(
            mean.data * (1 - self.momentum))
```

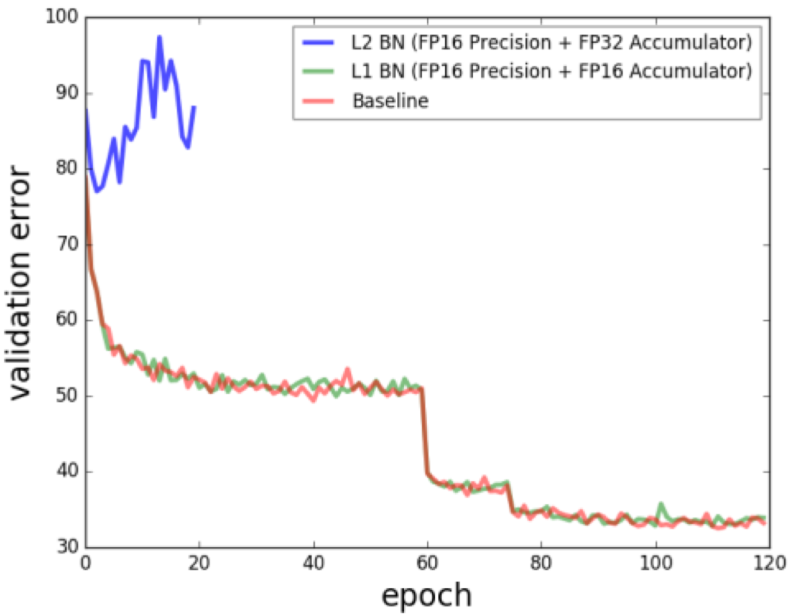
Code

Linf Batch Norm

```
mean = x.view(x.size(0), x.size(self.dim), -1).mean(-1).mean(0)
y = x.transpose(0, 1)
z = y.contiguous()
t = z.view(z.size(0), -1)
A = torch.abs(t.transpose(1, 0) - mean)

const = 0.5 * (1 + (np.pi * np.log(4)) ** 0.5) / \
    ((2 * np.log(A.size(0))) ** 0.5)

MeanTOPK = (torch.topk(A, self.k, dim=0)[0].mean(0)) * const
scale = 1 / (MeanTOPK + self.eps)

self.running_mean.mul_(self.momentum).add_(
    mean.data * (1 - self.momentum))

self.running_var.mul_(self.momentum).add_(
    scale.data * (1 - self.momentum))
```

Author’s Experimental Results

BN at Half Precision



- Results for ResNet18 on ImageNet
- L^1 BN is more robust to quantization compared to L^2 BN
- Half precision run on L^2 BN is clearly diverging and was stopped early

Experimental Analysis

- Weight Decay (WD) affects training process only indirectly, by modulating the learning rate
- Introduction of the normalization term C_{L1} helps network reach a lower final validation error at a faster rate
- L^1 norm improves both running time and memory consumption
- Using L^2 in low precision mode leads to overflow and significant quantization noise
- Using L^1, BN can be quantized to half precision with no effect on validation accuracy

Conclusion and Future Work

- L^1 and L^∞ - based normalization provides similar results to standard BN (allowing low-precision computation)
- C_{L1} normalization constant is critical for achieving same performance as L^2
- This can be used for easy mobile deployment of the networks
- Bounded weight normalization achieves improved results on large-scale tasks and is comparable to BN
- Bounded weight normalization enables improved learning in tasks like reinforcement learning and temporal modeling
- Strong connection between hyper-parameters exists and this can be leveraged to ease design and training by fixing some of the hyper-parameters

References

[1] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530, 2016.

[2] Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. Recurrent batch normalization. arXiv preprint arXiv:1603.09025, 2016

[3] Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Advances in Neural Information Processing Systems, pp. 901–909, 2016.

[4] Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

References

[5] Gitman, I. and Ginsburg, B. Comparison of batch normalization and weight normalization algorithms for the large-scale image classification. CoRR, abs/1709.08145, 2017.

[6] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In Advances in neural information processing systems, pp. 4107–4115, 2016.

[7] http://www.statmt.org/wmt14/translation-task.html

[8] https://www.mathworks.com/help/deeplearning/ref/resnet18.html

Work Distribution

- Aniruddha- Connection between Weight Decay, learning rate and normalization
- Akhil Sai - L_P Norms
- Zhe - L_P Norms & Norm Bounded Weight Normalization : Results and visualization
- Hemanth - Norm Bounded Weight Normalization