PAUL AGGARWAL

Big Transfer

QUESTION 3:

Group Normalization (GN)

Group Normalization (GN) is better alternative to BN. Normalizing along the batch dimension introduces problems — BN's error increases rapidly when the batch size becomes smaller, caused by inaccurate batch statistics estimation. This limits BN's usage for training larger models and transferring features to computer vision tasks including detection, segmentation, and video, which require small batches constrained by memory consumption. BN exhibits drawbacks that are also caused by its distinct behavior of normalizing along the batch dimension. In particular, it is required for BN to work with a sufficiently large batch size (e.g., 32 per worker2). A small batch leads to inaccurate estimation of the batch statistics, and reducing BN's batch size increases the model error dramatically.

With the normalization method, it is well-known that normalizing the input data makes training faster. To normalize hidden features, initialization methods have been derived based on strong assumptions of feature distributions, which can become invalid when training evolves.

GN divides the channels into groups and computes within each group the mean and variance for normalization. GN's computation is independent of batch sizes, and its accuracy is stable in a wide range of batch sizes. On ResNet-50 trained in ImageNet, GN has 10.6% lower error than BN when using a batch size of 2; when using typical batch sizes, GN is comparably good with BN and outperforms other normalization variants. Moreover, GN can be naturally transferred from pre-training to fine-tuning. GN can outperform its BN based counterparts for object detection and segmentation in in ImageNet, COCO, and for video classification in Kinetics, showing that GN can effectively replace BN in a variety of tasks. GN can be easily implemented by a few lines of code in modern libraries.

GN could also be used in place of LN, IN and WN (that also avoid normalizing along the batch dimension and do not have issues by caused by batch normalization) for sequential or generative models. LN, IN and WN are not able to approach same accuracy as BN.

Where BN works well, GN can approach BN's accuracy, with a decent degradation of 0.5% in the validation set. Experiments show that GN has lower training error than BN, indicating that GN is effective for easing optimization. The slightly higher validation error of GN implies that GN loses some regularization ability of BN. This is understandable, because BN's mean and variance computation introduces uncertainty caused by the stochastic batch sampling, which helps regularization. This uncertainty is missing in GN (and LN/IN). But it is possible that GN combined with a suitable regularizer will improve results (adding Weight Standardization to GN).

BN's error becomes considerably higher with small batch sizes. GN's behavior is more stable and insensitive to the batch size.

GN allows to remove the batch size constraint imposed by BN, which can give considerably more memory (e.g., 16× or more). This will make it possible to train higher-capacity models that would be otherwise bottlenecked by memory limitation.

Compared GN with BN on ResNet-101. With a batch size of 32, BN on ResNet-101 has 22.0% validation error, and the GN counterpart has 22.4%, slightly worse by 0.4%. With a batch size of 2, GN ResNet-101's error is 23.0%. This is still a decently stable result considering the very small batch size, and it is 8.9% better than the BN counterpart's 31.9%.

For VGG-16, GN is better than BN by 0.4%. This possibly implies that VGG-16 benefits less from BN's regularization effect, and GN (that leads to lower training error) is superior to BN in this case.

BN's effectiveness to train models in turn prohibits people from exploring higher-capacity models that would be limited by memory. The restriction on batch sizes is more demanding in computer vision tasks including detection, segmentation, video recognition, and other high level systems built on them. For example, the Fast/er and Mask R-CNN frameworks use a batch size of 1 or 2 images because of higher resolution, where BN is "frozen" by transforming to a linear layer; in video classification with 3D convolutions, the presence of spatial-temporal features introduces a trade-off between the temporal length and batch size. The usage of BN often requires these systems to compromise between the model design and batch sizes.

GN does not require group convolutions. GN is a generic layer, as we evaluate in standard ResNets.

GN as a layer that divides channels into groups and normalizes the features within each group, It does not exploit the batch dimension, and its computation is independent of batch sizes. GN behaves very stably over a wide range of batch sizes. With a batch size of 2 samples, GN has 10.6% lower error than its BN counterpart for ResNet-50 in ImageNet. With a regular batch size, GN is comparably good as BN (with a gap of ∼0.5%) and outperforms other normalization variants. Although the batch size may change, GN can naturally transfer from pretraining to fine-tuning.

Batch Normalization performs more global normalization along the batch dimension (and as importantly, it suggests to do this for all layers). Batch-wise normalization is not legitimate at inference time, so the mean and variance are pre-computed from the training set, often by running average; consequently, there is no normalization performed when testing. The pre-computed statistics may also change when the target data distribution changes. These issues lead to inconsistency at training, transferring, and testing time. Also, reducing the batch size can have dramatic impact on the estimated batch statistics. However, group-wise normalization (GN) for deep neural networks seems to avoid the issues stated above.

Batch normalization (BN)

Batch normalization (BN) is a technique to normalize activations in intermediate layers of deep neural networks. BN normalizes the features by the mean and variance computed within a (mini-)batch to ease optimization and enable very deep networks to converge. The stochastic uncertainty of the batch statistics also acts as a regularizer that can benefit generalization.

BN primarily enables training with larger learning rates, which is the cause for faster convergence and better generalization. For networks without BN it demonstrated how large gradient updates can result in diverging loss and activations growing uncontrollably with network depth, which limits possible learning rates. BN avoids this problem by constantly correcting activations to be zero-mean and of unit standard deviation, which enables larger gradient steps, yields faster convergence and may help bypass sharp local minima.

Normalizing the input data of neural networks to zero-mean and constant standard deviation has been known for decades to be beneficial to neural network training. With the rise of deep networks, Batch Normalization (BN) naturally extends this idea across the intermediate layers within a deep network, although for speed reasons the normalization is performed across mini-batches and not the entire training set. BN accelerates training, enables higher learning rates, and improves generalization accuracy.

Activations and gradients in deep neural networks without BN tend to "explode" during an early on-set of divergence. The typical practice to avoid such divergence is to set the learning rate to be sufficiently small such that no steep gradient direction can lead to divergence. However, small learning rates yield little progress along flat directions of the optimization landscape and may be more prone to convergence to sharp local minima with possibly worse generalization performance. BN avoids activation explosion by repeatedly correcting all activations to be zero-mean and of unit standard deviation. With this "safety precaution", it is possible to train networks with large learning rates, as activations cannot grow in-controllably since their means and variances are normalized. SGD with large learning rates yields faster convergence along the flat directions of the optimization landscape and is less likely to get stuck in sharp minima. Also, batch normalized networks that do not use high learning rates performs no better than an unnormalized network. It is the higher learning rate that is allowed by using BN to improve regularization, accuracy, faster convergence, yields faster training (lower epochs) and minimize overfitting.

In conclusion, larger learning rate increases the implicit regularization of SGD, which improves generalization. Our Unnormalized networks can result in activations whose magnitudes grow dramatically with depth, which limits large learning rates. Unnormalized networks have large and ill-behaved outputs, and that this results in gradients that are input independent. Batch normalization are primarily caused by higher learning rates and focus on the gradients at initialization. For unnormalized networks only small gradient steps lead to reductions in loss, whereas networks with BN can use a far broader range of learning rates. Training a Resnet that uses one batch normalization layer only at the very last layer of the network, normalizing the output of the last residual block allows for learning rates up to 0.03 and yields higher final test accuracy. Normalizing the final layer of a deep network is important contribution of BN.

Weight Standardization (WS)

Batch Normalization (BN) effectiveness is limited for micro-batch training, i.e., each GPU typically has only 1-2 images for training, which is inevitable for many computer vision tasks, e.g., object detection and semantic segmentation, constrained by memory consumption. Weight Standardization (WS) can be used to address this issue.

WS standardizes the weights in convolutional layers, i.e., making the weights have zero mean and unit variance. WS reduces the Lipschitz constants of the loss and the gradients and improves training.

Experiments show that on tasks where large-batches are available (e.g. ImageNet), GN + WS with batch size 1 is able to match or outperform the performances of BN with large batch sizes.

Many results show that training smooth functions using gradient descent algorithms is faster than training non-smooth functions. Intuitively, gradient descent based training algorithms can be unstable due to exploding or vanishing gradients. As a result, they are sensitive to the selections of the learning rate and initialization if the loss landscape is not smooth. Using an algorithm (e.g. WS) that smooths the loss landscape will make the gradients more reliable and predictive; larger steps can be taken and the training will be accelerated.

Deep neural networks are hard to train partly due to the singularities caused by the non-identifiability of the model. Elimination singularities, which correspond to the points on the training trajectory where neurons in the model become constantly deactivated. So, we consider elimination singularities for networks that use ReLU as their activation functions. When ReLU is used, ωc = 0 is no longer necessary for a neuron to be eliminatable. This is because ReLU sets any values below 0 to 0; thus a neuron is constantly deactivated if its maximum value after the normalization layer is below 0. Their gradients will also be 0 because of ReLU, making them hard to revive; hence, a singularity is created.

GN prevents the models from getting too close to singularities. Without the help of batch information, GN alleviates this issue by assigning channels to more than one group to encourage more activated neurons, and WS adds constraints to pull the channels to be not so statistically different.

When the batch size is limited to 1, GN+WS is able to achieve performances comparable to BN with large batch size. GN+WS should be used for micro-batch training because GN shows the best results among all the normalization methods that can be trained with 1 image per GPU

There will be improvements not only in the final model accuracy, but also in the training speed and a big drop of training error rates at each epoch. This demonstrates that the model is now farther from elimination singularities, resulting in an easier and faster learning.

BN reduces the Lipschitz constants of the loss function, and makes the gradients more Lipschitz, too, i.e., the loss will have a better β-smoothness. We notice that BN considers the Lipschitz constants with respect to activations, not the weights that the optimizer is directly optimizing. However, WS will standardize the weights in the convolutional layers to further smooth the landscape. By doing so, we do not have to worry about transferring smoothing effects from activations to weights; moreover, the smoothing effects on activations and weights are also additive.

WS standardizes the weights in a differentiable way which aims to normalize gradients during back-propagation. When WS normalizes the gradients, it lowers the Lipschitz constants of the loss and the gradients which makes the loss landscape smoother by optimizing L on W. The optimizer take longer steps without worrying about sudden changes of the loss landscape and gradients. Therefore, WS accelerates training.

Weight Standardization (WS) bring the success factors of Batch Normalization (BN) into micro-batch training,) the smoothing effects on the loss landscape and 2) the ability to avoid harmful elimination singularities along the training trajectory. WS standardizes the weights in convolutional layers. Similar to BN, WS controls the first and second moments of the weights of each output channel individually in convolutional layers. Having a normalization technique such as WS will ease the training without worrying about the memory and batch size issues.

In conclusion, WS reduces the Lipschitz constants of the loss and the gradients, and thus it smooths the loss landscape. WS is able to keep far distances from elimination singularities, caused by lack of batch knowledge. WS improves micro-batch training significantly, and WS+GN with batch size 1 is even able to match or outperform the performances of BN with large batch sizes. GN+WS outperforms both BN and GN.