SUMMARY ON BATCH NORMALIZATION

MINI-BATCH GRADIENT DESCENT

This algorithm combines the robustness of stochastic gradient descent (which updates the model for each training example) and the efficiency of batch gradient descent (which updates the model after all training examples have been evaluated). This algorithm proceeds by considering a mini batch X(1)…..(m) of size m.

ADVANTAGES:

- The gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases.
- Computations over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

DISADVANTAGES:

- Inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.
- The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience covariate shift.

INTERNAL COVARIATE SHIFT

To improve the training, we seek to reduce the internal covariate shift. By fixing the distribution of the layer inputs x as the training progresses, we expect to improve the training speed.

We could consider whitening activations at every training step or at some interval, either by modifying the network directly or by changing the parameters of the optimization algorithm to depend on the network activation values. However, if interspersed with the optimization steps, then the gradient descent step may attempt to update the parameters in a way that requires the normalization to be updated, nullifying effect of the gradient step.
This problem can get worse if the normalization not only centers but also scales the activations. We have observed this empirically in initial experiments, where the model blows up when the normalization parameters are computed outside the gradient descent step.

NORMALIZATION

The full whitening of each layer's inputs is costly and not everywhere differentiable. So our other alternative is Normalization.
For a layer with d - dimensional input we will normalize each dimension

$$\hat{x}^k = \frac{x^k - E[x^k]}{\sqrt{Var[x^k]}}$$

The transformation inserted in the network should represent the identity transform. To accomplish this, we introduce parameters γ and β and the normalized input is transformed to:

$$y^k = \gamma^k \hat{x} + \beta^k$$

Since we use mini-batches in stochastic gradient training, each mini-batch produces estimates of the mean and variance of each activation. This way, the statistics used for normalization can fully participate in the gradient backpropagation.
We refer to the transform:

$BN_{\gamma , \beta} : x_{1...m} \rightarrow y_{1...m}$

as the Batch Normalizing Transform

This BN transform is differentiable and ensures that as the model is training, the layers can learn on the input distributions that exhibit less internal covariate shift and can hence accelerate the training. At training time, a subset of activations in specified and BN transform is applied to all of them. During test time, the normalization is done using the population statistics instead of mini-batch statistics to ensure that the output deterministically depends on the input.

BATCH-NORMALIZED CONVOLUTIONAL NETWORKS

By applying BN transform and ignoring the bias b (Since its effect will be canceled by the subsequent mean subtraction), the equation *$z = g(Wu + b)$* transforms to *$z = g(BN(Wu))$*, where the BN transform is applied independently to each dimension of $x = Wu$, with a separate pair of learned parameters γ (k) , β (k) per dimension.
For convolutional layers, to obey the convolutional property, we jointly normalize all the activations in a mini-batch, over all locations. So for a mini-batch of size m and feature maps of size $p \times q$, we use the effective mini-batch of size *$m' = |B| = m \cdot p\, q$*. We learn a pair of parameters *γ (k)* and *β (k)* per feature map, rather than per activation.

Simply adding Batch Normalization to a network does not take full advantage of our method. To do so, we further changed the network and its training parameters, as follows:

- Increase Learning rate
- Remove dropout
- Reduce the L2 weight regularization
- Accelerate the learning rate decay
- Remove Local Response Normalization
- Shuffle training examples more thoroughly
- Reduce the photometric distortions

CONCLUSION

Merely adding Batch Normalization to a state-of-the-art image classification model yields a substantial speedup in training. By further increasing the learning rates, removing Dropout, and applying other modifications afforded by Batch Normalization, we reach the previous state of the art with only a small fraction of training steps – and then beat the state of the art in single-network image classification. Furthermore, by combining multiple models trained with Batch Normalization, we perform better than the best known system on ImageNet, by a significant margin

The goal of Batch Normalization is to achieve a stable distribution of activation values throughout training, and in our experiments we apply it before the nonlinearity since that is where matching the first and second moments is more likely to result in a stable distribution

We plan to investigate whether Batch Normalization can help with domain adaptation, in its traditional sense – i.e. whether the normalization performed by the network would allow it to more easily generalize to new data distributions, perhaps with just a recomputation of the population means and variances