Ghost Batch Normalization for MNIST dataset training

Sitao Min

Batch Normalization

Batch Normalization is first proposed by Sergey Ioffe and Christian Szegedy in 2015.【1】 It is an algorithm to overcome the bad effects of the *internal covariate shift* which is the phenomenon that the training processes will be slowed down and too hard to be trained because of change of the distribution of each layer's inputs during training. As we all know, the training result of deep neural network is sensitive to the distribution of initial parameter so it requires careful parameter initialization. However, the activation of each layers output can be regarded as the input of the next layer so the distribution of each layer inputs can also affect the performance of deep neural network. Considering this factor, the core idea of batch normalization is to fix the distribution of the layer inputs during the training processes to remove the ill effects of the internal covariate shift and promote the performance of deep learning. The common method to fix the distribution of each layer's inputs is to use whitening -- linearly transforming inputs to have zero means and unit variances, and make them decorrelation. 【2】However, full whitening of each layer's inputs is costly and not everywhere differentiable. So in Batch Normalization algorithm, they made two necessary simplifications. First is that instead of whitening the features in layer inputs and outputs jointly, they normalize each scalar feature independently, by making it have the mean of zero and the variance of 1. For a layer with d-dimensional input $\mathrm{x} = (\mathrm{x}(1) \ldots \mathrm{x}(\mathrm{d}))$, normalization of each dimension is $\hat{\mathrm{x}}(\mathrm{k}) = \frac{\mathrm{x}(\mathrm{k}) - \mathrm{E}[\mathrm{x}(\mathrm{k})]}{\sqrt{Var[x[k]]}}$【1】Second is that they introduce, for each activation x(k), a pair of parameters $\gamma(\mathrm{k}), \beta(\mathrm{k})$, and $\mathrm{y}(\mathrm{k}) = \gamma(\mathrm{k})\hat{\mathrm{x}}(\mathrm{k}) + \beta(\mathrm{k})$,which scale and shift the normalized value to avoid the problem that simply normalizing each input of a layer may change what the layer can represent.【1】Full details of batch normalization algorithm are showed below.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm: Batch Normalizing Transform

Ghost Batch Normalization Algorithm

Ghost Batch Normalization is an algorithm designed by Elad Hoffer【3】to solve the 'generalization gap' problem. The 'generalization gap' phenomena is when using large batch sizes to train deep neural network, there is a persistent degradation in generalization performance proposed by

Keskar【4】. The core idea of Ghost Batch Normalization is to use small batch information to help large batch promote its train performance. The details of algorithm shows below.

Algorithm 1: Ghost Batch Normalization, applied to activation x over a large batch B_L with virtual mini-batch B_S. Where $B_S < B_L$.

Require: Values of x over a large-batch: $B_L = \{x_{1\dots m}\}$ size of virtual batch $|B_S|$; Parameters to be learned: γ, β, momentum η

Training Phase:

Scatter B_L to $\{X^1, X^2, \dots X^{|B_L|/|B_S|}\} = \{x_{1\dots|B_S|}, x_{|B_S|+1\dots 2|B_S|} \dots x_{|B_L|-|B_S|\dots m}\}$

$\mu_B^l \leftarrow \frac{1}{|B_S|}\sum_{i=1}^{|B_S|} X_i^l \quad \text{for } l = 1, 2, 3\dots$ {calculate ghost mini-batches means}

$\sigma_B^l \leftarrow \sqrt{\frac{1}{|B_S|}\sum_{i=1}^{|B_S|}(X_i^l - \mu_B)^2 + \epsilon} \quad \text{for } l = 1, 2, 3\dots$ {calculate ghost mini-batches std}

$\mu_{run} = (1-\eta)^{|B_S|}\mu_{run} + \sum_{i=1}^{|B_L|/|B_S|}(1-\eta)^i \cdot \eta \cdot \mu_B^l$

$\sigma_{run} = (1-\eta)^{|B_S|}\sigma_{run} + \sum_{i=1}^{|B_L|/|B_S|}(1-\eta)^i \cdot \eta \cdot \sigma_B^l$

return $\gamma\frac{X^l - \mu_B^l}{\sigma_B^l} + \beta$

Test Phase:

return $\gamma\frac{X - \mu_{run}^l}{\sigma_{run}} + \beta$ {scale and shift}

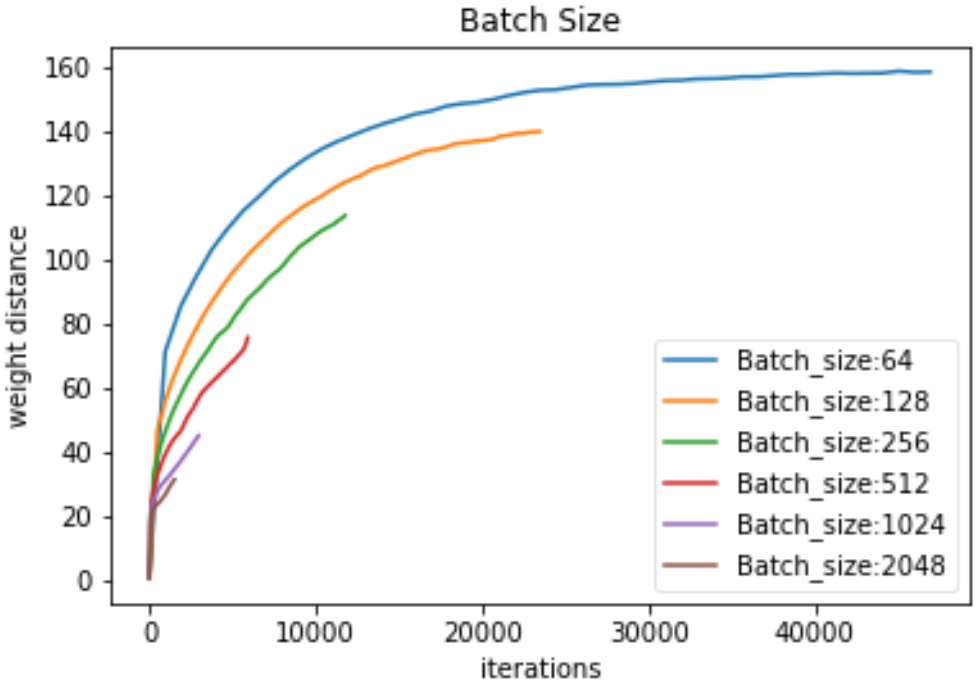
MNIST Datasets Training Result

We perform deep neural network with ghost batch normalization and adjusted learning rate to MNIST datasets. The network structure is 5-layer neural network. Each layer contains 512 nodes. We choose stochastic gradient descent optimizer with hyper parameters (learning rate: 1e-1, weight decay: 1e-4, momentum: 0.9). For different batch size, we adjust their learning rate with following formula:

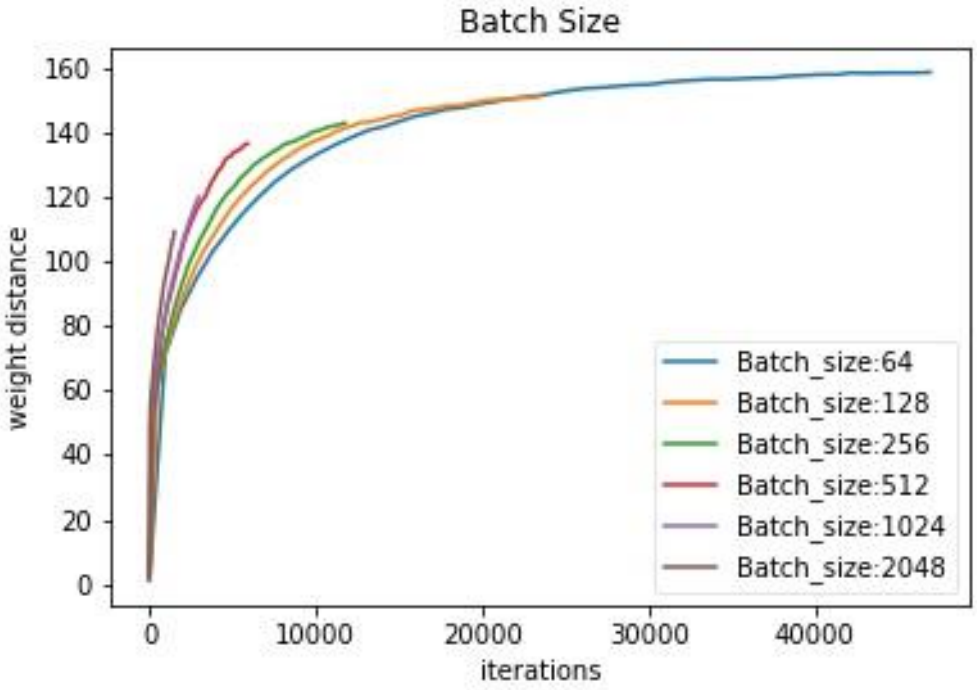
$$\text{Learning rate } = \text{Learning rate}0(=1\text{e}-1) \times \sqrt{\frac{Large\ Batch\ Size}{Mini\ Batch\ Size(=64)}}$$

And we separate large batch to many mini-batch to perform Ghost Batch Normalization algorithm. The size of mini-batch is 64.

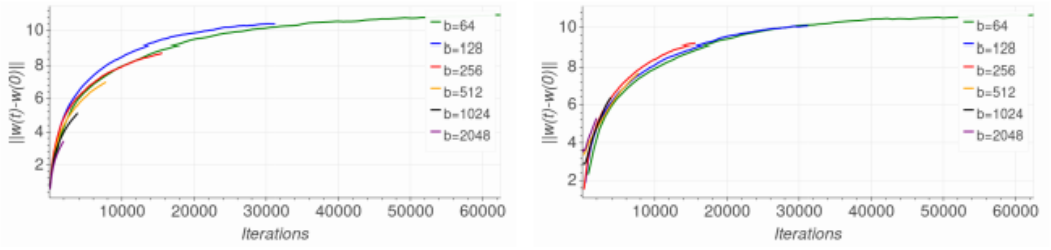
The training result is as below. Before performing GBN and adjusting learning rate, the weight distance from initialization point increases logarithmically with the number of iterations (training time). However, with different batch size, the relation curve has different slope (diffusion rate), which may cause loss function fall into different minima and this may cause neural network has different performance (generalization gap). After performing GBN and adjusting learning rate, the weight distance from initialization point increases logarithmically with the number of iterations (training time) and the slope of relation curve is closer. So, loss function is more likely to fall into same minima and deep neural network can have similar performance when training with different batch size. So we can see that GBN is likely to reduce generalization gap for different batch size.



Before adjusting learning rate and gbn



After adjusting learning rate and gbn



(a) Before learning rate adjustment and GBN (b) After learning rate adjustment and GBN

Figure 2: Euclidean distance of weight vector from initialization

Original figure in reference 【3】

References

【1】S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015

【2】Wiesler, Simon and Ney, Hermann. A convergence analysis of log-linear training. In Shawe-Taylor, J., Zemel, R.S., Bartlett, P., Pereira, F.C.N., and Weinberger, K.Q. (eds.), Advances in Neural Information Processing Systems 24, pp. 657–665, Granada, Spain, December 2011

【3】Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. arXiv preprint arXiv:1705.08741, 2017.

【4】Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In ICLR, 2017.