Francis's standard

batch normalization

In a recent paper published on arxiv (http://arxiv.org/abs/1502.03167) by Sergey Ioffe and Christian Szegedy, a technique for accelerating deep neural network learning called batch normalization was introduced. They suggest that a change in the distribution of activations because of parameter updates slows learning. They call this change the *internal covariate shift.*

The paper proposes an efficient method to partially alleviate this phenomenon. During SGD training, each activation of the mini-batch is centered to zero-mean and unit variance. The mean and variance are measured over the whole mini-batch, independently for each activation. A learned offset β and multiplicative factor γ are then applied. This process is called batch normalization. The formal algorithm is shown in the figure below, reproduced from Ioffe and Szegedy's arxiv (http://arxiv.org/abs/1502.03167) paper.

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 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

(https://standardfrancis.files.wordpress.com/2015/04/screenshot-from-2015-04-16-133436.png)Once SGD learning has stopped, a post-training step is applied where the mean and variance for each activation is computed on the whole training dataset (rather than on mini-batches). This new mean and variance replaces the ones computed on mini-batches. The whole training procedure is shown in the figure below, also reproduced from Ioffe and Szegedy's arxiv (http://arxiv.org/abs/1502.03167) paper.

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

$N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network

for $k = 1 \dots K$ **do**

Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)

Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

end for

Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters

for $k = 1 \dots K$ **do**

// For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$

end for

Algorithm 2: Training a Batch-Normalized Network

(https://standardfrancis.files.wordpress.com/2015/04/screenshot-from-2015-04-16-133447.png)

IMPLEMENTATION

Implementing this is quite simple in theano. In liblearn (https://github.com/francisquintallauzon/ift6266h15), I made batch normalization a new type of layer with a "post-learning" operation. That's it!

The following shows the exact architecture of the model tested.

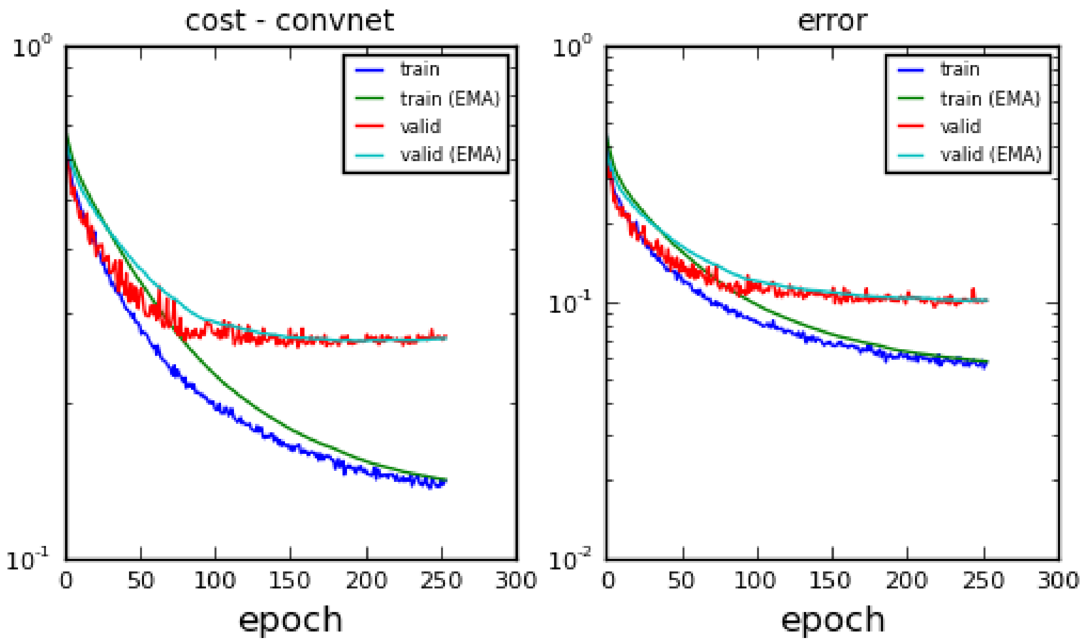
layer 1 : (3, 90, 90) –> (3, 90, 90) on conv_preprocess
layer 2 : (3, 90, 90) –> (16, 88, 88) on conv_vanilla with (16, 3, 3, 3) filters and relu activation
layer 3 : (16, 88, 88) –> (16, 44, 44) on conv_maxpool with (2, 2) downsampling
layer 4 : (16, 44, 44) –> (16, 44, 44) on conv_batchnorm
layer 5 : (16, 44, 44) –> (32, 42, 42) on conv_vanilla with (32, 16, 3, 3) filters and relu activation
layer 6 : (32, 42, 42) –> (32, 42, 42) on conv_batchnorm
layer 7 : (32, 42, 42) –> (32, 40, 40) on conv_vanilla with (32, 32, 3, 3) filters and relu activation
layer 8 : (32, 40, 40) –> (32, 20, 20) on conv_maxpool with (2, 2) downsampling
layer 9 : (32, 20, 20) –> (32, 20, 20) on conv_batchnorm
layer 10 : (32, 20, 20) –> (64, 16, 16) on conv_vanilla with (64, 32, 5, 5) filters and relu activation
layer 11 : (64, 16, 16) –> (64, 8, 8) on conv_maxpool with (2, 2) downsampling
layer 12 : (64, 8, 8) –> (64, 8, 8) on conv_batchnorm
layer 13 : (64, 8, 8) –> (64, 4, 4) on conv_vanilla with (64, 64, 5, 5) filters and relu activation
layer 14 : (64, 4, 4) –> (64, 4, 4) on conv_batchnorm
layer 15 : (64, 4, 4) –> (256,) on hidden with (1024, 256) filters and relu activation
layer 16 : (256,) –> (2,) on logistic with (256, 2) filters

RESULTS

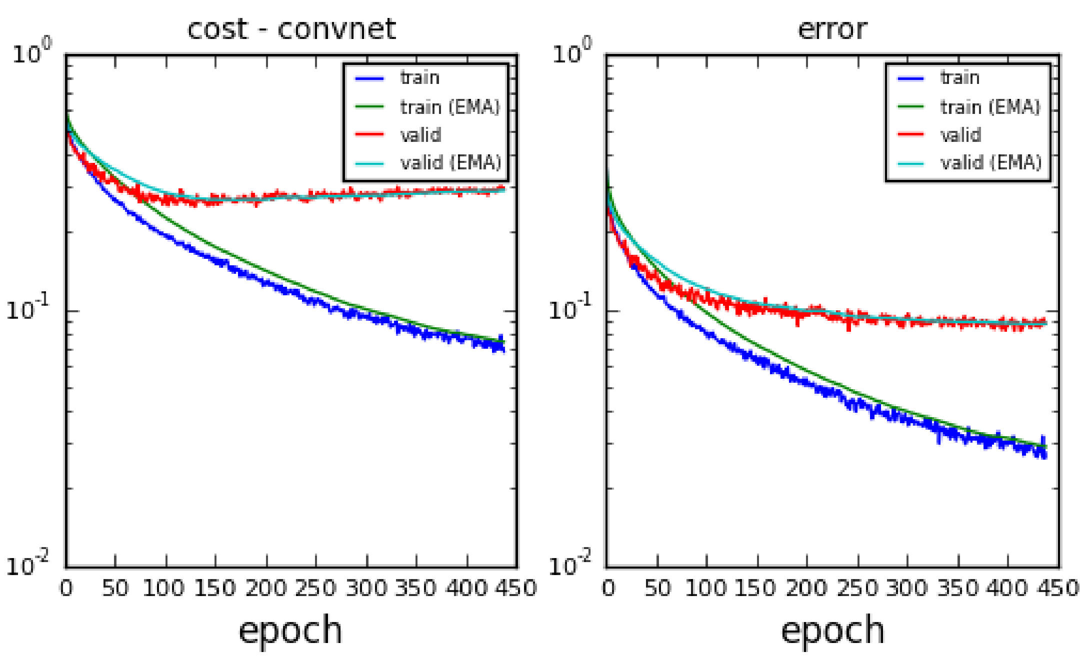
Except for the new batch normalization layers, the model I tested is the exact same as the one of my first post. Therefore, let's compare both results. To do this comparison, I took the best model I trained *with* and *without* batch normalization.

model type	**valid**	**test**
vanilla	8.32%	8.96%
batch normalization	8.36%	7.84%

The main claim of the paper in question suggests that convergence is faster with batch normalization and without. The figure below shows the NLL cost and error as a function of training epoch for the model **without** batch normalization,



(https://standardfrancis.files.wordpress.com/2015/04/convnet.png)and the figure below shows the same evolution for the model **with** batch normalization.



(https://standardfrancis.files.wordpress.com/2015/04/convnet-batch-norm.png)Though it might be a bit confusing because of the two figures have a different X and Y scale, careful inspection shows that it takes approximately 50 epochs to reach a NLL of 0.12 in both models. Based on this observation, I can't conclude that batch normalization yields faster convergence.

I did however get a significantly improved error rate with this model, which suggest it might help. One notable thing I must add is that I did significantly more hyper-parameter exploration on this model than the vanilla convnet I implemented.

NEXT

In class, Kyle suggested that much larger learning rates can be used with this method. This is an obvious next step and I'll report the observation I made in my next post.

- April 16, 2015
- francisquintallauzon