Batch Normalization

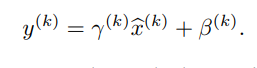
Training Deep Neural Networks is complicated by the fact that the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. This phenomenon is called internal covariate shift.

It can be reduced by normalizing layer inputs and making normalization a part of the model architecture and performing the normalization for each training mini-batch.

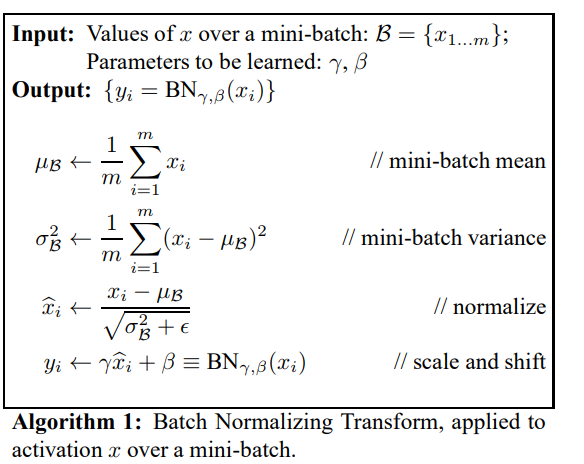
Batch Normalization allows us to use much higher learning rates and be less careful about initialization.

**Algorithm:**

Normalize each scalar feature independently, by making it have the mean of zero and the variance of 1. Since normalizing each input of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. To prevent this, for each activation x (k) ,a pair of parameters γ (k) , β(k) is used which scale and shift the normalized value .



These parameters are learned along with the original model parameters



References: <https://arxiv.org/pdf/1502.03167v3.pdf>

Tied Weights

It means parameter sharing. Since the encoding and decoding layers mirror each other in structure, you parameters can be shared between them so only one set of weights can be learned. The decode weights are the transpose of the encode weights.

This is often preferred over learning separate weights for both phases because

(a) The number of parameters is reduced and can train faster

(b) It is seen as a form of regularization that leads to better performance in practice.

Denoising AutoEncoder

The idea behind denoising autoencoders is simple. In order to force the hidden layer to discover more robust features and prevent it from simply learning the identity, we train the autoencoder to reconstruct the input from a corrupted version of it.

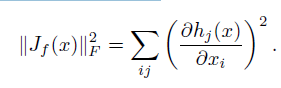
Intuitively, a denoising auto-encoder does two things: try to encode the input (preserve the information about the input), and try to undo the effect of a corruption process stochastically applied to the input of the auto-encoder.

So the Loss function is calculated by comparing the output values with the original input, not with the corrupted input.

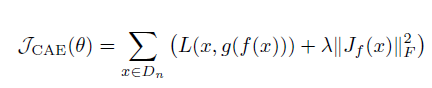
Contractive Autoencoder

A contractive autoencoder makes this encoding less sensitive to small variations in its training dataset. This is accomplished by adding a penalty term, to whatever cost function the algorithm is trying to minimize. The end result is to reduce the learned representation’s sensitivity towards the training input.

This sensitivity penalization term is the sum of squares of all partial derivatives of the extracted features with respect to input dimensions:



The total cost function becomes:



Where ƛ is a hyper parameter to tune.

**References:**

<http://www.icml-2011.org/papers/455_icmlpaper.pdf>

<https://arxiv.org/ftp/arxiv/papers/1305/1305.4076.pdf>

Sparse AutoEncoder

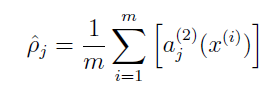
Sparse autoencoders offer us an alternative method for introducing an information bottleneck without requiring a reduction in the number of nodes at our hidden layers. Rather, we'll construct our loss function such that we penalize activations within a layer. For any given observation, we'll encourage our network to learn an encoding and decoding which only relies on activating a small number of neurons.

Whereas an undercomplete autoencoder will use the entire network for every observation, a sparse autoencoder will be forced to selectively activate regions of the network depending on the input data. As a result, we've limited the network's capacity to memorize the input data without limiting the networks capability to extract features from the data.

To impose this sparsity constraint we measure the hidden layer activations for each training batch and add some term to the loss function in order to penalize excessive activations.



Where :



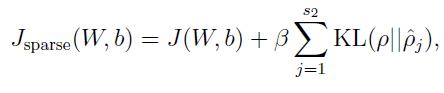
average activation of hidden unit j (averaged over the training set).

And ƿ is a sparsity parameter, typically a small value close to zero .

This penalty function has the property that



The total cost function will be



Where beta controls the weight of the sparesity penalty term.

**References:**

<http://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf?fbclid=IwAR1eQ75MZi98fzxyKkRdrhq9DC-dpRGsvRjxk1ckbobRNymLFpVpB-Slf_A>