Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

1 Citation

Ioffe, Sergey, and Christian Szegedy. ”Batch normalization: Accelerating deep network training by reducing internal covariate shift.” arXiv preprint arXiv:1502.03167 (2015).

`https://arxiv.org/pdf/1502.03167.pdf`

2 Abstract

When training a deep net, the input distribution of layer l will change as we change weights in layer $l-1$ - this is called internal covariate shift. Batch normalization is designed to mitigate this and helps models converge faster and regularize better (it can replace dropout sometimes). It also lets us use saturating nonlinearities (i.e. we don't only need to use ReLU). We get state of the art on ImageNet.

3 Introduction

Minibatch Stochastic Gradient Descent is the dominant approach for training deep nets. Larger minibatches decrease variance in the gradient estimate and can be parallelized and computed efficiently. Training a layer of a network is easier if previous layers are fixed, but in practice we do end-to-end training - how do we mitigate this internal covariate shift? Batch normalization fixes this by ensuring that the layer inputs have fixed mean and variance.

4 Towards Reducing Internal Covariate Shift

People usually whiten (i.e. zero-mean unit variance) inputs to neural nets to help convergence. So, let us whiten inputs into layers within the network. The challenge is that, as we update weights, the normalization computation has to change (because the un-normalized input distribution changes). Computing normalization parameters externally does not help because it can cause weights to explode - we need to ensure that the gradient descent step is aware of the normalization. Suppose the input to a layer is $\boldsymbol{x}$ and the training set is $\mathcal{X}$. We seek: $\hat{\boldsymbol{x}} = \text{norm}(\boldsymbol{x}, \mathcal{X})$. For backprop, we need to compute the gradient of $\hat{\boldsymbol{x}}$ w.r.t $\boldsymbol{x}$ and $\mathcal{X}$. The $\mathcal{X}$ makes this way too compute intensive.

5 Normalization via Minibatch Statistics

So, to normalize, we compute statistics on the minibatch $\mathcal{B} = \{x_{1...m}\}$ instead of the entire training set $\mathcal{X}$. Notice that we are looking at one dimension of the feature vector here over all the vectors in the minibatch. We'd like the model to be able to undo this batch norm if it would like, so we also let it learn

γ and β to scale and shift the normed value. This gives the following equations (where ϵ avoids division by zero):

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \tag{2}$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{B}^2 + \epsilon}} \tag{3}$$

$$y_i = \gamma \hat{x}_i + \beta = \mathrm{BN}_{\gamma,\beta}(x_i) \tag{4}$$

We can compute the derivative for each of the equations above, so the whole thing is fully differentiable. At test-time, we can use the population mean and variance of the training set instead of the values estimated from the minibatch (test time may not have minibatches). We can also keep an exponentially weighted mean and variance so the network can adapt its estimates at test time.

We apply batch norm before layers of the form: $z = g(Wu + b)$, which covers fully connected and conv layers (for a conv layer we need to learn one (γ, β) per batch rather than per activation).

Batch norm helps make gradients better behaved. Exactly how this happens requires further study. It also regularizes the model (you can train without dropout or use a lower dropout rate).

6 Experiments

We trained a 3 hidden layer feedforward net on MNIST. Using batch norm improves test error. Visualizing the distribution of an activation in the final hidden layer, we see that, without batch norm, the distribution changes a lot over time. With batch norm, it's pretty stable.

For ImageNet, we use the Inception network as our basis. We add batch norm (in a convolutional way as described earlier), but we also make some other changes. We increase the learning rate, remove dropout, reduce L2 weight regularization, accelerate learning rate decay (our network should train faster, after all), remove local response normalization, shuffle training examples more thoroughly (we don't want the same images to appear together in a minibatch), reduce the photometric distortions (i.e. add less distortion to our training images). For one experiment, we also use sigmoid activation instead of ReLU. With all these changes, except sigmoid activation, we get 74.8% top-1 accuracy with a single model. With ensembling, we can cut top-5 error to 4.82%, which is state of the art.

7 Conclusion

Batch normalization is designed to mitigate internal covariate shift, which should accelerate training by keeping the input distribution to a layer fixed. It only adds two extra parameters per activation. This enables us to increase training rates, not use dropout, and use other (i.e. not ReLU) non-linearities. We set state of the art for ImageNet and it takes us less time to train our model.

Batch normalization does not seem to make activations sparser, however. Future work should focus on applying batch normalization to recurrent neural nets. We should also see if batch norm helps when generalizing to new tasks (can you just compute population mean and variance on new training set)?