IST 597: Foundations of Deep Learning **(Due: 03/16/22)**

Assignment #01000

Instructor: **Dr. Lee Giles** *TA:* **Ankur Mali** *Name: , PSUID:*

Created by Ankur Mali
Source : Thanks and credits to materials from MIT, Google, NYU

Course Policy: Carefully read all the instructions, before you start working on the assignment

- Please typeset your submissions in any LaTeX or word template, give maximum explanation for each sub-problems. Please include your name and PSUID with submission along with date and time on the first page.
- Assignments are due before class at 02:29:59 pm {Please check the due date on the **Official course** webportal} Portal.
- Please avoid single line answers, submissions without any explanations would receive **0** points.
- Late assignments will suffer 50 percent points reductions per day, formula is $x/2$, where x=number of days and counter will start at 02:30:00 pm.
- All source materials must be cited. The University Academic Code of Conduct will be strictly enforced.
- We will be creating Canvas submission page for this. Submit all files on Canvas.
- All queries related to Assignment should have a subject line IST597:Assignment01000 Queries

Problem 1. Build your own Batch Norm Layer. (10 points)

To motivate batch normalization, let us review a few practical challenges that arise when training machine learning models and neural networks in particular.

- choices regarding data preprocessing often make an enormous difference in the final results. Recall our prior 2 assignments, simply dividing by 255 works, but is not always a best choice
- Better approach is to standardize our input feature space by taking mean (ideally zero) and variance (ideally close to 1), this approach often plays nicely with stochastic optimizers because it adds kind of a priori to model parameters and puts them at a similar scale and helps in faster convergence.

Second, for a typical MLP or CNN, as we train, the variables (e.g., affine transformation outputs or output after pre-activations in feedforward system (MLP,CNN)) in intermediate layers or hidden layers, could take values with varying magnitudes both along input and output layers, across neurons in same layers and over time as training or model updates are computed. I would advise to read materials on BN, to better understand theory behind BN and how it helps in optimization [3, 1, 2, 4]. Furthermore original papers postulates that such drift in the distribution leads to sub-optimal performance and in some scenario poor convergence and generalization. Finally BN acts as regularizer, adding it always helps when dealing with deeper layers.

In practice batch normalization (BN) is applied to individual layers (optionally, to all of them) and works as follows: In each training iteration, we first normalize the inputs (of batch normalization) by subtracting their mean and dividing by their standard deviation, where both are estimated based on the statistics of the current minibatch. Next, we apply a scale coefficient and a scale offset. It is precisely due to this *normalization* based on *batch* statistics that *batch normalization* derives its name.

One important point to remember is that when applying batch normalization, the choice of batch size may be even more significant than without batch normalization. One can apply BN after pre-activation, as g(BN(f(x)) where f(x) = X.W + b and g is any non-linear activation function or after post-activation as BN(g(f(x))

Formally, denoting by $\mathbf{x} \in \mathcal{B}$ an input to batch normalization (BN) that is from a minibatch $\mathcal{B}$, batch normalization transforms $\mathbf{x}$ according to the following expression:

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

In equation x, $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ is the sample mean and $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$ is the sample standard deviation of the minibatch $\mathcal{B}$. After applying standardization, the resulting minibatch has zero mean and unit variance. Because the choice of unit variance (vs. some other magic number) is an arbitrary choice, we commonly include elementwise *scale parameter* $\boldsymbol{\gamma}$ and *shift parameter* $\boldsymbol{\beta}$ that have the same shape as $\mathbf{x}$. Note that $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are parameters that need to be learned jointly with the other model parameters.

Consequently, the variable magnitudes for intermediate layers cannot diverge during training because batch normalization actively centers and rescales them back to a given mean and size (via $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ and $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$). One piece of practitioner's intuition or wisdom is that batch normalization seems to allow for more aggressive learning rates.

Formally, we calculate $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ and $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$ as follows:

$$\hat{\boldsymbol{\mu}}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x},$$
$$\hat{\boldsymbol{\sigma}}^2_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} (\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}})^2 + \epsilon.$$

Note that we add a small constant $\epsilon > 0$ to the variance estimate to ensure that we never attempt division by zero, even in cases where the empirical variance estimate might vanish. The estimates $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ and $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$ counteract the scaling issue by using noisy estimates of mean and variance. You might think that this noisiness should be a problem. As it turns out, this is actually beneficial.

This turns out to be a recurring theme in deep learning. For reasons that are not yet well-characterized theoretically, various sources of noise in optimization often lead to faster training and less overfitting: this variation appears to act as a form of regularization. In particular, this sheds some light on the puzzle of why batch normalization works best for moderate minibatches sizes in the $50 \sim 100$ range.

Fixing a trained model, you might think that we would prefer using the entire dataset to estimate the mean and variance. Once training is complete, why would we want the same image to be classified differently, depending on the batch in which it happens to reside? During training, such exact calculation is infeasible because the intermediate variables for all data examples change every time we update our model. However, once the model is trained, we can calculate the means and variances of each layer's variables based on the entire dataset. Indeed this is standard practice for models employing batch normalization and thus batch normalization layers function differently in *training mode* (normalizing by minibatch statistics) and in *prediction mode* (normalizing by dataset statistics).

Let's see how this works in practice

Batch Normalization Layers

Batch normalization implementations for fully-connected layers and convolutional layers are slightly different. We discuss both cases below. Recall that one key differences between batch normalization and other layers is that because batch normalization operates on a full minibatch at a time, we cannot just ignore the batch dimension as we did before when introducing other layers.

Fully-Connected Layers

When applying batch normalization to fully-connected layers, the original paper inserts batch normalization after the affine transformation or pre-activation and before the nonlinear activation or post activation function (later applications may insert batch normalization right after activation functions). Denoting the input to the fully-connected layer by $\mathbf{x}$, the affine transformation (f(x)) by $\mathbf{W}\mathbf{x} + \mathbf{b}$ (with the weight parameter $\mathbf{W}$ and the bias parameter $\mathbf{b}$), and the activation function by ϕ, we can express the computation of a batch-normalization-enabled, fully-connected layer output $\mathbf{h}$ as follows:

$$\mathbf{h} = \phi(\mathrm{BN}(\mathbf{W}\mathbf{x} + \mathbf{b})).$$

or

$$\mathbf{h} = \mathrm{BN}\phi((\mathbf{W}\mathbf{x} + \mathbf{b})).$$

Recall that mean and variance are computed on the same minibatch on which the transformation is applied.

Convolutional Layers

Similarly, with convolutional layers, we can apply batch normalization after the convolution and before the nonlinear activation function. When the convolution has multiple output channels, we need to carry out batch normalization for each of the outputs of these channels, and each channel has its own scale and shift

parameters, both of which are scalars. Assume that our minibatches contain m examples and that for each channel, the output of the convolution has height p and width q. For convolutional layers, we carry out each batch normalization over the $m \cdot p \cdot q$ elements per output channel simultaneously. Thus, we collect the values over all spatial locations when computing the mean and variance and consequently apply the same mean and variance within a given channel to normalize the value at each spatial location.

Batch Normalization During Prediction

As we mentioned earlier, batch normalization typically behaves differently in training mode and prediction mode. First, the noise in the sample mean and the sample variance arising from estimating each on minibatches are no longer desirable once we have trained the model. Second, we might not have the luxury of computing per-batch normalization statistics. For example, we might need to apply our model to make one prediction at a time.

Typically, after training, we use the entire dataset to compute stable estimates of the variable statistics and then fix them at prediction time. Consequently, batch normalization behaves differently during training and at test time. Recall that dropout also exhibits this characteristic.

Things to do in this Assignment:

- We have provided a codebase (starter code) for CNN and MLP trained on Fmnist.
- Implement custom Batch Norm from scratch by modifying these files.
- Given BN works differently while training and testing. Remember to have various BN function for train and test(validation and test both).
- You will report performance of models when BN (MLP and CNN) is applied after pre-activation (g(BN(f(x)) when trained on Fashion MNIST (In this phase you will have 2 models).
- You will also report performance of your models when BN (MLP and CNN) is applied after post-activation (BN(g(f(x)) when trained on Fashion MNIST (In this phase you will have 2 models).
- Four all 4 models, report performance for minimum 3 trials.

References

[1] Bjorck, J., Gomes, C., Selman, B., and Weinberger, K. Q. Understanding batch normalization, 2018.

[2] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[3] Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization?, 2019.

[4] Yang, G., Pennington, J., Rao, V., Sohl-Dickstein, J., and Schoenholz, S. S. A mean field theory of batch normalization, 2019.