Deep Learning in Hardware — ECE 498/598 (Fall 2020)
Prof. Naresh Shanbhag — Assigned: Oct. 2 - Due: Oct. 9

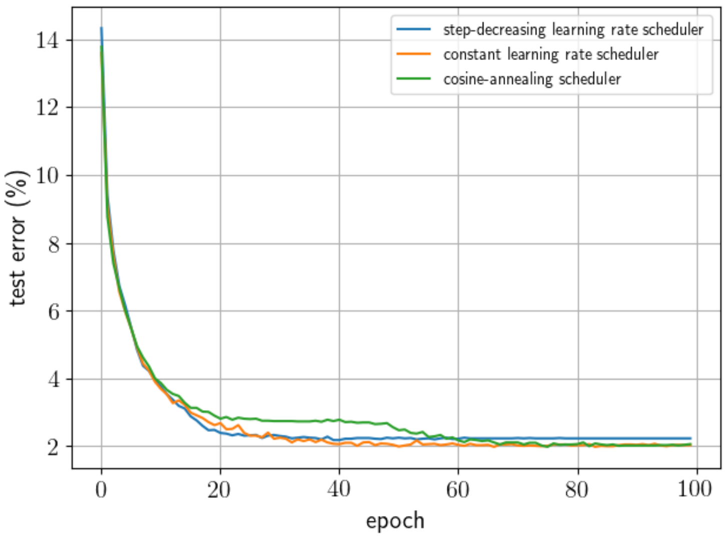
Homework 3

This homework contains 3 problems. There are no star problems in this homework. The topics covered in this homework include: DNN training with quantized gradients, derivation of weight initialization, BatchNorm absorption. This homework is due on Oct. 9 via Gradescope. No extension will be provided, so make sure to get started as soon as possible.

Problem 1: Training Neural Networks using Python with Quantized Gradients

In this problem, you will need the MNIST dataset from `http://yann.lecun.com/exdb/mnist/`. You can initiate the dataset directly using the PyTorch API. Your job is to write a Python code to train a neural network with an MLP architecture of 784-512-256-256-10 that performs the classification task on the MNIST dataset. You are to use the vanilla version of SGD. You are encouraged to read through all questions in this problem before starting to code.

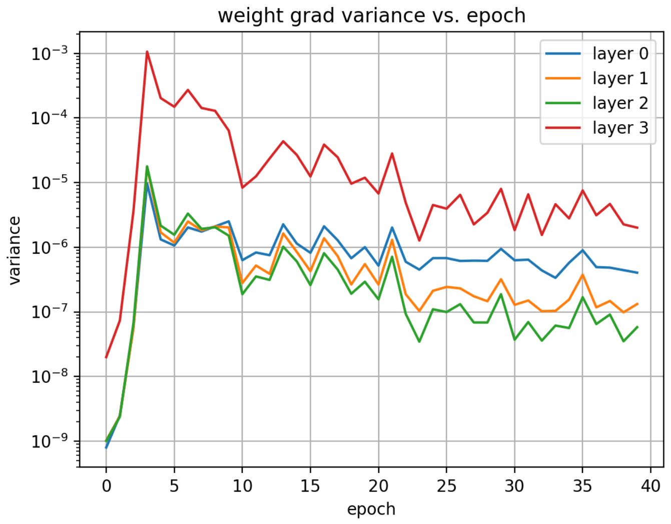
1. You can find tutorials on how to adjust learrning rate in PyTorch at `https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate`. In this problem, train the network with **a**: constant learning rate, **b**: a cosine annealing learning rate, and **c**: a step-decaying learning rate (learning rate decreases every x epochs). Plot the convergence curve of the test error of your network as a function of time (measured in epochs) for each learning rate scheduler.
   Solution: We train the network using SGD using the provided Python script attached with the solution. Below is the convergence curve of the test error.



   The step scheduler decrease the learning rate every 30 epochs. It does not achieves a performance as good as a constant learning rate or a cosine-annealing schedule.

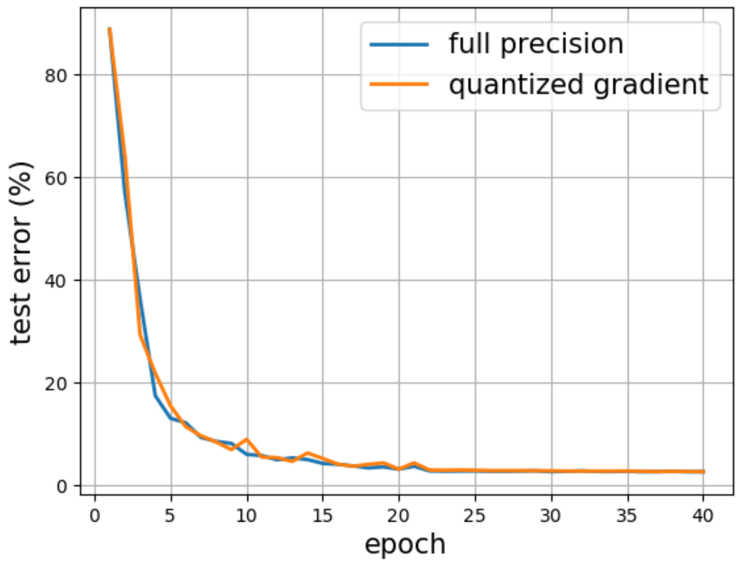
2. For the model trained using scheduler **a**, plot the per-tensor variance of each weight

gradient as a function of time (measured in epochs), and record the maximum and minimum values of recorded variances during training.
Solution: The per-tensor evolution of the gradient variances is shown below.



The maximum variances recorded per layer are [9.74E-06 1.76E-05 1.79E-05 1.06E-03]. And the minimum ones are [7.96E-10 1.02E-09 1.01E-09 2.00E-08].

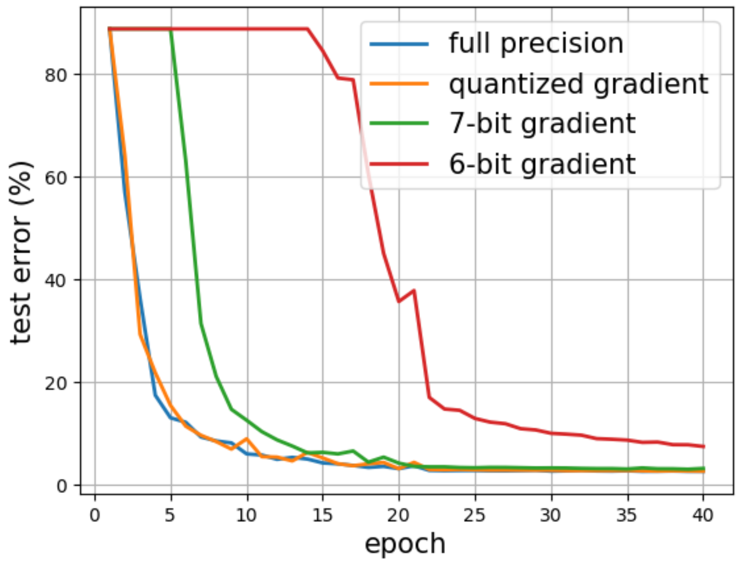
3. You are asked to retrain this network with scheduler **a**, but using close-to-minimal (CTM) quantized weight gradients. The challenge is to determine a suitable clipping level and quantization step-size (Hint: use your answers above). Show convergence curves of training with CTM quantized gradients and report the number of bits used for each tensor.
Solution: As per the discussion in class on fixed-point training, we set the clipping level using Criterion 2: $r = 5\sigma_{\max}$. We obtain the following clipping levels: [0.015625 0.03125 0.03125 0.25]. The quantization step is set using Criterion 3: $\delta = 0.25\sigma_{\min}$. We obtain the following quantization steps: [3.8147E-06 7.62939E-06 7.62939E-06 3.05176E-05]. The number of bits at each layer is therefore $1 + \log_2 \frac{r}{\delta} =:$ [13,13,13,14]. The convergence curve of the test error when using CTM quantized gradients is shown below.



As can be seen, the model with CTM quantized gradients converges with close fidelity to the full precision baseline.

4. Compare the test error convergence curve obtained using CTM quantized weight gradients in Part 2, with those obtained by quantizing the weight gradients to: a) 6-b, and b)7-b, for all layers.

Solution: With the same dynamic ranges as found above, we can try to quantize all gradients to 7 bits and observe weak convergence with a significant accuracy drop. We can also try 6 bits and observe an unstable behavior and large accuracy drop. These results are included below.



Problem 2: Deriving Weight Initialization Formulas

In this problem, your task is to derive the two key equations utilized in the He initialization scheme (`https://arxiv.org/abs/1502.01852`). In what follows, we assume weights are zero-mean and independent. Further, activations are independent but non-zero mean (because of their rectifying nature).

1. Show that during for the forward propagation we have:

$$Var(y_L) = Var(y_1)\left(\prod_{l=2}^{L} \frac{1}{2} n_l Var(w_l)\right)$$

   where y_l, w_l, and n_l are the pre-activations, weights, and forward dot-product length at layer l and, L is the number of layers.
   Solution: The first part to explain is how do we get to eq. (6) in the paper, namely:

$$Var(y_l) = n_l Var(w_l x_l)$$

   This is simply due to the independence assumption of the individual product terms which allows us to obtain the variance of the sum as sum of variances. The identical distribution assumption gives us the factor of n_l.

   Next we need to explain how to get to eq. (8), namely:

$$Var(y_l) = \frac{1}{2} n_l Var(w_l) Var(y_{l-1})$$

   This is simply due to the independence of w_l and x_l and the fact that w_l is zero-mean. Furthermore, we have $E[x_l^2] = \frac{1}{2}Var(y_l)$ because of the rectifier operation of the ReLU. Assuming y_{l-1} has a symmetric distribution around zero, ReLU rectifies out half of the signal power.

   Finally, the desired equation is obtain by recursing over eq. (8) across the L layers.

2. Show that during the backward propagation we have:

$$Var(\Delta x_2) = Var(\Delta x_{L+1})\left(\prod_{l=2}^{L} \frac{1}{2} \hat{n}_l Var(w_l)\right)$$

   where Δx_{l+1} and $\hat{n}_l$ are the activation gradients and backward dot-product length at layer l.
   Solution: The procedure is exactly similar tot he first question but when considering the back propagation of gradients instead of the forward propagation of activations.

3. In a convolutional layer, how are the forward dot-product length n_l and backward dot-product length $\hat{n}_l$ related?
   Solution: We have $n_l = k_l^2 c_l$ and $\hat{n}_l = k_l^2 d_l$ where k_l is the filter dimension, c_l is the number of input channels, and d_l is the number of output channels. Thus n_l and $\hat{n}_l$ are related but are not the same.

4. Explain how the He initialization prevents the vanishing or explosion of activations in the forward propagation.
   Solution: The He initialization is engineered so that each term $\frac{1}{2}n_l Var(w_l)$ is equal to 1. As these get multiplied together in order to compute variances of activations, there is no risk of explosion or vanishing.

5. Explain how the He initialization prevents the vanishing or explosion of gradients in the backward propagation (HINT: the answer to this question is not the same as that of the previous one).
   Solution: While it is not the case that $\frac{1}{2}\hat{n}_l Var(w_l) = 1$. It can be shown that when $\frac{1}{2}n_l Var(w_l) = 1$, then $Var(\Delta x_{L+1})\left(\prod_{l=2}^{L}\frac{1}{2}\hat{n}_l Var(w_l)\right) = c_2/d_L$. This is due to the correpondance between n_l and $\hat{n}_l$ discussed in question 3 of this problem. Thus the He initialization also prevent vanishing/explosion of gradients.

Problem 3: BatchNorm Absorption

Recall in Batch Normalization, the output $\langle \mathbf{w}, \mathbf{x}\rangle$ of a layer with activation or feature map $\mathbf{x}$ with a weight or filter $\mathbf{w}$ is transformed as:

$$\gamma\frac{\langle \mathbf{w}, \mathbf{x}\rangle - \mu}{\sqrt{\sigma^2+\epsilon}} + \beta$$

where γ, β, μ, and σ are the BatchNorm (BN) parameters, ϵ is a numerical stability constant, and $< \cdot, \cdot >$ denotes the dot product or convolution depending on the context.

If only inference needs to be performed, it is possible to eliminate the extra computations required by BN by absorbing its parameters into the weights. This can be rewriting the above equation as:

$$\langle \hat{\mathbf{w}}, \mathbf{x}\rangle + b$$

where $\hat{\mathbf{w}}$ and b are a new weight/filter and a bias, respectively.

1. Derive expressions for $\hat{\mathbf{w}}$ and b as a function of $\mathbf{w}$, γ, β, μ, σ, and ϵ?
   Solution: By inspection, we have

$$\hat{\mathbf{w}} = \frac{\gamma \mathbf{w}}{\sqrt{\sigma^2+\epsilon}}$$

   and

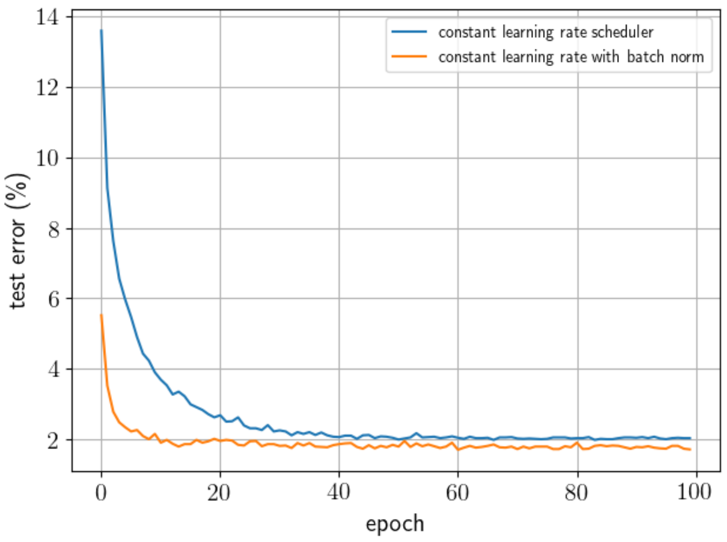
$$b = \beta - \frac{\gamma\mu}{\sqrt{\sigma^2+\epsilon}}$$

2. For the case of convolutions, write a Python script that returns the BN 4D tensor $\hat{\mathbf{w}}$ and 1D vector b for standard 2D convolution given a original weight tensor $\mathbf{w}$ and BN parameters γ, β, μ, σ, and ϵ. You only need to submit a script for this question (Hint: the solution is not trivial and requires *broadcasting*).
   Solution: The Python script to realize the operation is as follows

```
def fuse_conv_and_bn(conv, bn): #nn.Conv2d, nn.BatchNorm2d
   #
   # init
   fusedconv = torch.nn.Conv2d(
       conv.in_channels,
       conv.out_channels,
       kernel_size=conv.kernel_size,
       stride=conv.stride,
       padding=conv.padding,
       bias=True
   )
   #
   # prepare filters
   w_conv = conv.weight.clone().view(conv.out_channels, -1)
   w_bn = torch.diag(bn.weight.div(torch.sqrt(bn.eps+bn.running_var)))
   fusedconv.weight.copy_( torch.mm(w_bn, w_conv).view(fusedconv.weight.size()) )
   #
   # prepare spatial bias
   if conv.bias is not None:
       b_conv = conv.bias
   else:
       b_conv = torch.zeros( conv.weight.size(0) )
   b_bn = bn.bias - bn.weight.mul(bn.running_mean).div(torch.sqrt(bn.running_var + bn.eps))
   fusedconv.bias.copy_( b_conv + b_bn )
   #
   # we're done
   return fusedconv
```

3. For the MLP model in Problem 1, add a BN layer after each of the first three fully connected layers and retrain the network from scratch. Plot the convergence of test error for the MLP before and after adding the BN layer. Pay attention to the order between the BN layer and the ReLU unit. How many parameters does this new MLP have compared to the old one?

   Solution:



The benefits of BN are evident in the faster convergence and lower test error rate.