# Homework 7.

## Due on Gradescope by 5 PM, Saturday, May 29

**Submission format.** Please submit the programming problem through **google colab notebook** as you have done in the previous homeworks. Please submit your notebook in **.ipynb** format instead of PDF on Gradescope. Students are recommended to organize their code into modular cells with proper comments for ease of readability and follow the same steps for submitting the colab notebook as done in previous homeworks. You will see a separate assignment on gradescope for Programming of this homework – be sure to submit each problem to the right assignment on gradescope.

## Theoretical Part

1. **RMSprop Algorithm:** You are given $\boldsymbol{x_i}$ as the input vector, $y_i$ as the corresponding class label for $i = 1, 2$ and $\boldsymbol{w}$ as the weight vector:

$$\boldsymbol{w} = [w_0, w_1, w_2]^T. \tag{1}$$

Here,

$$\boldsymbol{x_1} = [1, 2, 0]^T, \quad y_1 = -1$$
$$\boldsymbol{x_2} = [1, 4, 4]^T, \quad y_2 = 1$$

We want to train a simple perceptron using a hinge loss to discriminate between the two classes. Use **RMSprop gradient descent** (one sample at a time) algorithm to update your weights $\hat{\boldsymbol{w}}$ and show weights after each step for 2 update steps. You should update your weights 2 times, once for each sample above.

Initialize weight vector as $\boldsymbol{w} = [w_0, w_1, w_2]^T = [0, 0, 0]$ and set the learning rate to 1. For a desired target output $y \in \pm 1$, your prediction will be $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$ and hinge loss is defined as $L = max(0, 1 - y\hat{y})$

After each of the 2 training steps, provide the predictions $\hat{\boldsymbol{y}}$, loss value, and updated weights in $\boldsymbol{w}$.

**RMSprop Algorithm:**

$$\begin{aligned} S_{dw_i} &= 0.9 S_{dw_i} + 0.1 dw_i^2 \\ dw_i &\leftarrow dw_i - \frac{\alpha}{\sqrt{S_{dw_i} + \epsilon}} dw_i \end{aligned} \tag{2}$$

Where, $dw_i = \frac{\delta L}{\delta w_i}$ is the gradient of the loss $L$ with respect to $w_i$ (which is the $i^{th}$ element of weight vector), $S_{dw_i}$ is the accumulated average value of $dw_i^2$, $\alpha = 1.0$ is the learning rate, and $\epsilon = 0.1$.

## Programming Part

1. **Effectiveness of residual connections:** In the previous homework, you were introduced to the challenges of training a deep network where the convergence rate decreases as the network gets deeper. You have observed the effectiveness of batch-norm for addressing this issue. In this problem, let us now use residual connections to train the deep network shown in Fig 1 (see this link for more details about residual connections and their advantage: https://arxiv.org/pdf/1512.03385.pdf). Choose $N = 10$ and replace the blocks shown in the the network in Fig. 1, with residual blocks (the implementation for the residual block can be found in Fig. 3). As with the previous homework, use the CIFAR10 dataset for training and testing.

   (a) Plot Train and Test accuracy curve versus the number of iterations.

   (b) Compare the result of training this network with residual connections to that of training with the network *without* these connections (but with batch norm). You did this experiment in the previous homework with $N = 10$ and a *block* in the network was replaced with a convolutional block with batch norm (this block is shown in Fig. 2). Report the observed differences in accuracy as well as convergence speed.

**Note for the programming part**:

1. Normalize your input data to be 0-mean, unit variance (hint: you can use **torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))** in the list of transforms to be applied during data loading). Remember, you input data should be in range of 0 to 1 to apply this transform.

2. Unless otherwise specified, the bias parameter is absent from a given layer.

3. Other training details: loss = Cross Entropy loss, Optimizer = SGD, learning rate = 0.01, number of iterations = 10000, batch size = 128.

4. Use **torch.manual_seed(4)** and **numpy.random.seed(4)** for both models to set the same seed for random generators.

5. Use **torch.nn.BatchNorm2d(channel_dim)** for batch normalization.

6. Be sure to set hardware accelerator to GPU so your training completes in reasonable amount of time.

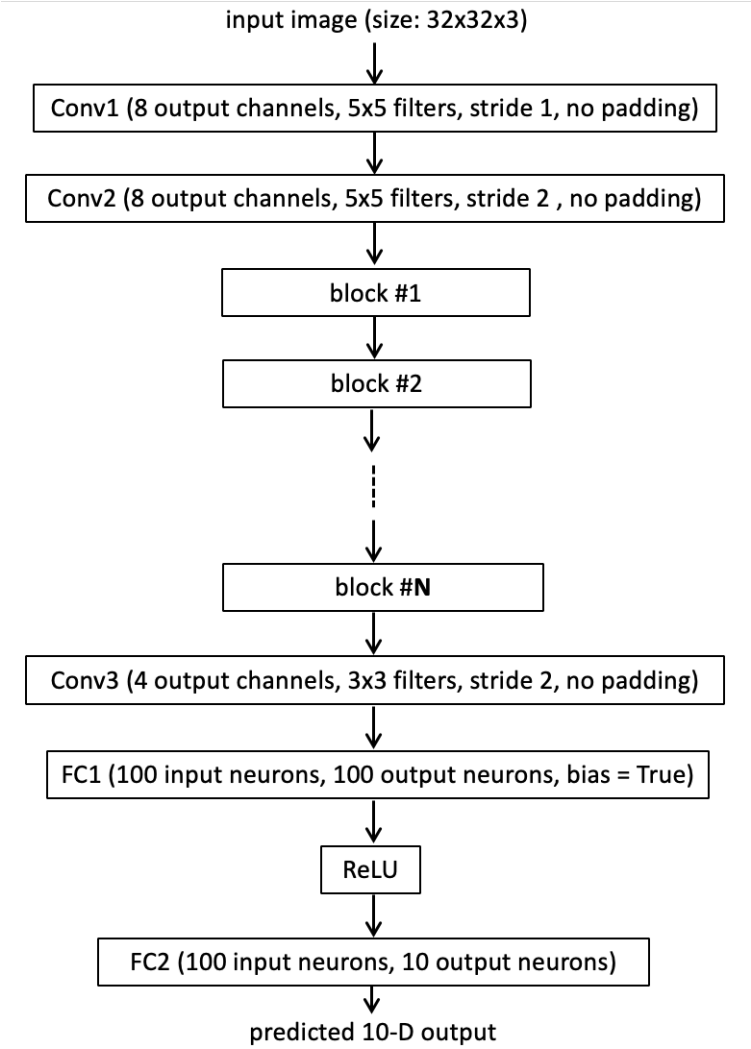


Figure 1: Overall network architecture: the *block* is either a pair of convolutional layers with batchnorm or a residual block. There are a total of 10 such blocks. FC = fully connected layer (linear layer

.

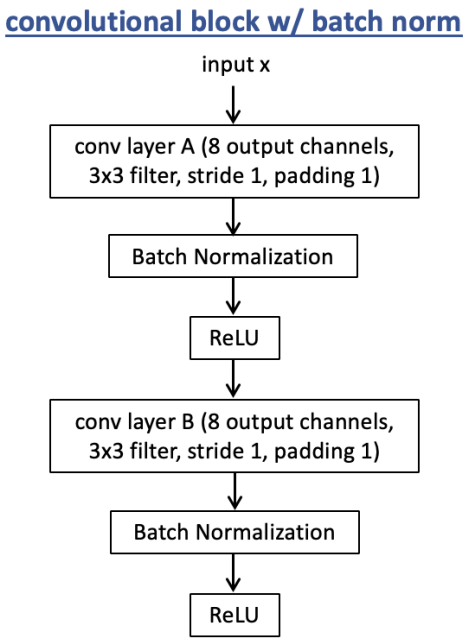


Figure 2: A block of conv layers with batch norm.

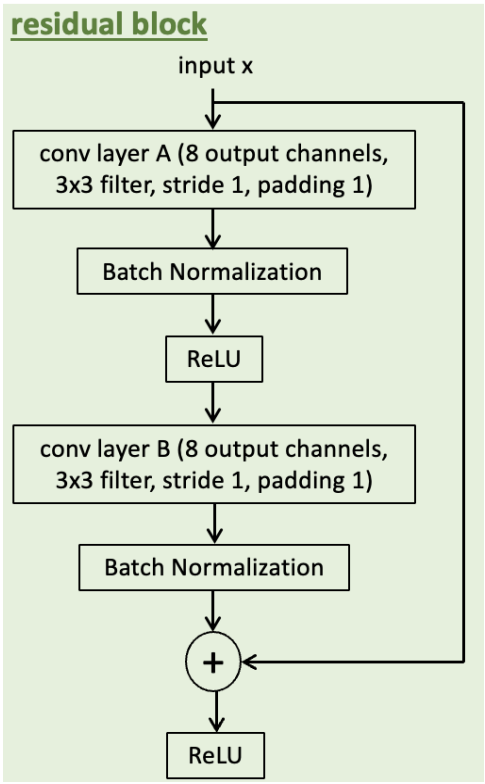


Figure 3: A residual block.