HW3.

Mengying Lin SID: 3038737132.

1. Normalization Layers

Recall the pseudocode for a batchnorm layer (with learnable scale and shift) in a neural network:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// standardize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(a) If our input data (1-dimensional) to batchnorm follows roughly the distribution on the left:

Figure 1: Gaussian with mean $\mu = 1.5$, variance $\sigma^2 = 0.16$

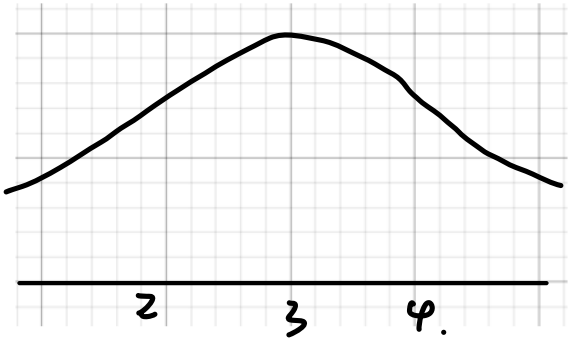


Figure 2: Blank grid for your answer

What does our data distribution look like after batch normalization with $\beta = 3$ and $\gamma = 1$ parameters? Draw your answer on the blank grid above, give a scale to the horizontal axis, and label β. You can assume that the batch-size is very large.

(Note: You do not have to give a scale to the vertical axis.)

(b) Say our input data (now 2-dimensional) to the batchnorm layer follows a Gaussian distribution. The mean and contours (level sets) of points that are 1 and 2 stdev away from the mean are shown below. **On the same graph, draw what the mean, 1-SD, and 2-SD contours would look like after batch-norm without any shifting/scaling (i.e. $\beta = 0$ and $\gamma = 1$).** You can assume that the batch-size is very large.

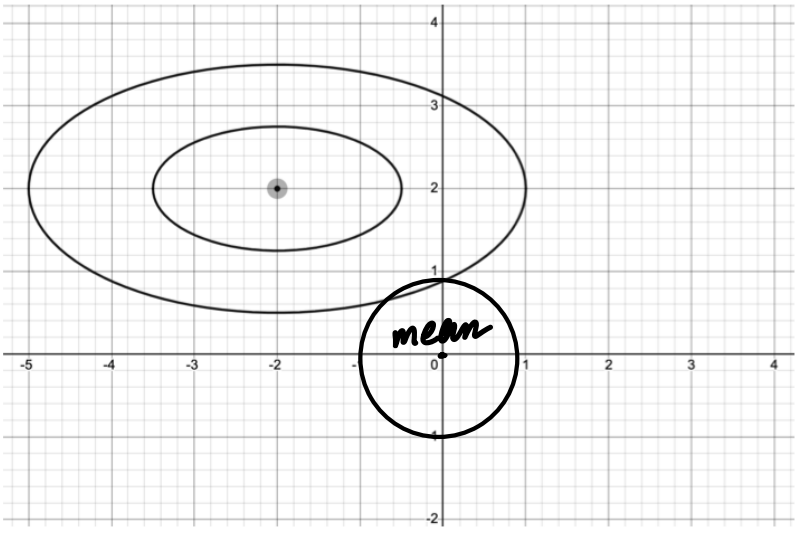


Figure 3: Draw your answer on the grid

2. Understanding Convolution as Finite Impulse Response Filter

For the discrete time signal, the output of linear time invariant system is defined as:

$$y[n] = x[n] * h[n] = \sum_{i=-\infty}^{\infty} x[n-i] \cdot h[i] = \sum_{i=-\infty}^{\infty} x[i] \cdot h[n-i] \tag{1}$$

where x is the input signal, h is impulse response (also referred to as the filter). Please note that the convolution operations is to 'flip and drag'. But for neural networks, we simply implement the convolutional layer without flipping and such operation is called correlation. Interestingly, in CNN those two operations are equivalent because filter weights are initialized and updated. Even though you implement 'true' convolution, you just ended up with getting the flipped kernel. **In this question, we will follow the definition in 1.**

Now let's consider rectangular signal with the length of L (sometimes also called the "rect" for short, or, alternatively, the "boxcar" signal). This signal is defined as:

$$x(n) = \begin{cases} 1 & n = 0, 1, 2, ..., L-1 \\ 0 & \text{otherwise} \end{cases}$$

Here's an example plot for $L = 7$, with time indices shown from -2 to 8 (so some implicit zeros are shown):

(a) The impulse response is define as:

$$h(n) = (\frac{1}{2})^n u(n) = \begin{cases} (\frac{1}{2})^n & n = 0, 1, 2, ... \\ 0 & \text{otherwise} \end{cases}$$

Compute and plot the convolution of $x(n)$ and $h(n)$. For illustrative purposes, your plot should start at -6 and end at +12.

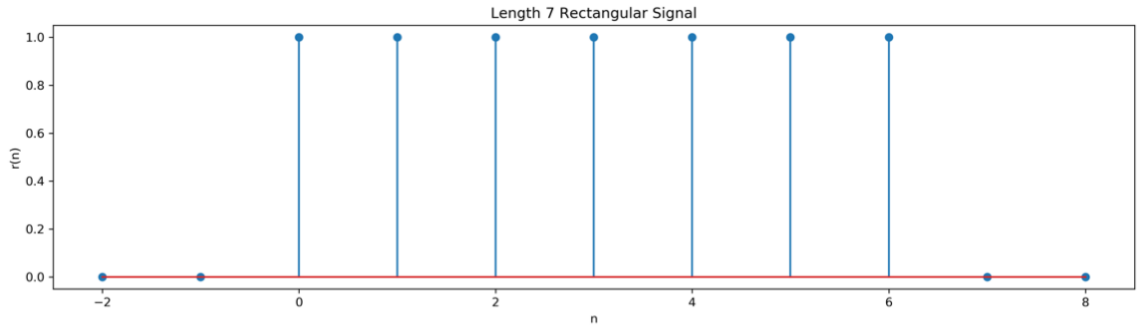


Figure 4: The rectangular signal with the length of 7

$$x(n) * h(n) = \sum_{i=-\infty}^{+\infty} x(n-i)\,h(i)$$

When $n \geq L$, $R.H.S. = \sum_{i=0}^{L-1} x(L-i)\,h(i) = \sum_{i=0}^{L-1} \left(\frac{1}{2}\right)^i = \frac{1-\left(\frac{1}{2}\right)^L}{1-\frac{1}{2}}$

$$= 2-\left(\frac{1}{2}\right)^{L-1}.$$

When $0 \leq n < L$, $R.H.S. = \sum_{i=0}^{n} x(n-i)\,h(i) = 2-\left(\frac{1}{2}\right)^n$

When $n < 0$. $R.H.S = 0$.

Hence $x(n) * h(n) = \begin{cases} 2-\left(\frac{1}{2}\right)^{L-1} & (n \geq L). \\ 2-\left(\frac{1}{2}\right)^n & (0 \leq n < L) \\ 0 & (n < 0). \end{cases}$

(b) Now let's shift $x(n)$ by N, i.e. $x_2(n) = x(n-N)$. Let's put $N = 5$ **Then, compute** $y_2(n) = h(n) * x_2(n)$**. Which property of the convolution can you find?**

Now, let's extend 1D to 2D. The example of 2D signal is the image. The operation of 2D convolution is defined as follows:

$$y[m,n] = x[m,n] * h[m,n] = \sum_{i,j=-\infty}^{\infty} x[m-i, n-j] \cdot h[i,j] = \sum_{i,j=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j] \tag{2}$$

, where x is input signal, h is FIR filter and y is the output signal.

$y_2(n) = x_2(n) * h(n) = \sum_{i=-\infty}^{+\infty} x_2(n-i)\,h(i) = \sum_{i=-\infty}^{+\infty} x(n-i-5)\,h(i)$.

① $n-5 \geq 6$. $\quad y_2(n) = \sum_{i=5}^{6+4} h(i) = \dfrac{(\frac{1}{2})^5 - (\frac{1}{2})^{6+4}}{1-\frac{1}{2}} = (\frac{1}{2})^4 - (\frac{1}{2})^{6+4}$.

$0 \leq n-5 < 6$. $\quad y_2(n) = \sum_{i=5}^{n} h(i) = (\frac{1}{2})^4 - (\frac{1}{2})^n$.

$n < 0$. $\quad y_2(n) = 0$.

Hence $x(n) * h(n) =$ $\begin{cases} (\frac{1}{2})^4 - (\frac{1}{2})^{6+4} & (n \geq 6+5) \\ (\frac{1}{2})^4 - (\frac{1}{2})^n & (5 \leq n < 6+5) \\ 0 & (n < 0) \end{cases}$

The result is the original one shifted by N, which means convolution has the invariance of shifting.

(c) 2D matrices, x and h are given like below:

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \tag{3}$$

$$h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4}$$

Then, evaluate y**. Assume that there is no pad and stride is 1.**

$$y = x * h = \begin{bmatrix} -40 & -40 & -40 \\ -40 & -40 & -40 \\ -40 & -40 & -40 \end{bmatrix}$$

Then, evaluate y. Assume that there is no pad and stride is 1.

(d) Now let's consider striding and padding. Evaluate y for following cases:

i. stride, pad = 1, 1

ii. stride, pad = 2, 1

i. $y = \left[\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 & 9 & 10 & 0 \\ 0 & 11 & 12 & 13 & 14 & 15 & 0 \\ 0 & 16 & 17 & 18 & 19 & 20 & 0 \\ 0 & 21 & 22 & 23 & 24 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \right]$ stride = 1.

$$= \begin{pmatrix} -19 & -28 & -32 & -36 & -29 \\ -30 & -40 & -40 & -40 & -30 \\ -30 & -40 & -40 & -40 & -30 \\ -30 & -40 & -40 & -40 & -30 \\ 49 & 68 & 72 & 76 & 59 \end{pmatrix}$$

ii. $y = \left[\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 & 9 & 10 & 0 \\ 0 & 11 & 12 & 13 & 14 & 15 & 0 \\ 0 & 16 & 17 & 18 & 19 & 20 & 0 \\ 0 & 21 & 22 & 23 & 24 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \right]$ stride = 2.

$$= \begin{pmatrix} -19 & -32 & -29 \\ -30 & -40 & -30 \\ 49 & 72 & 59 \end{pmatrix}$$

3. Feature Dimensions of Convolutional Neural Network

In this problem, we compute output feature shape of convolutional layers and pooling layers, which are building blocks of CNN. Let's assume that input feature shape is $W \times H \times C$, where W is the width, H is the height and C is the number of channels of input feature.

(a) A convolutional layer has 4 architectural hyperparameters: the filter size(K), the padding size (P), the stride step size (S) and the number of filters (F). **How many weights and biases are in this convolutional layer? And what is the shape of output feature that this convolutional layer produces?**

The bias matrix size is $\lfloor \frac{W+2P-K}{S}+1 \rfloor \times \lfloor \frac{H+2P-K}{S}+1 \rfloor \times F$.

Hence we have $\lfloor \frac{W+2P-K}{S}+1 \rfloor \times \lfloor \frac{H+2P-K}{S}+1 \rfloor \times F$ biases.

The filter matrix size is $K \times K \times C \times F$.

Hence we have $K \times K \times C \times F$ weights.

(c) Let's assume that we have the CNN model which consists of L successive convolutional layers and the filter size is K and the stride step size is 1 for every convolutional layer. Then **what is the receptive field size of the last output**?

Denote the receptive field of ith layer as RFi.

$RFi = (RFi+1 - 1) \times \text{stride} + K$.

Therefore one pixel in L th layer corresponds to $(K-1)(L-1)+1$ pixels in 1st layer.

(d) Consider a downsampling layer (e.g. pooling layer and strided convolution layer). In this problem, we investigate pros and cons of downsampling layer. This layer reduces the output feature resolution and this implies that the output features loose the certain amount of spatial information. Therefore when we design CNN, we usually increase the channel length to compensate this loss. For example, if we apply the max pooling layer with kernel size of 2 and stride size of 2, we increase the output feature size by a factor of 2. **If we apply this max pooling layer, how much does the receptive field increases? Explain the advantage of decreasing the output feature resolution with the perspective of reducing the amount of computation.**

The receptive field doubles.

Reducing the output feature resolution can decrease the total number of operations.

(e) Let's take a real example. We are going to describe a convolutional neural net using the following pieces:

- CONV3-F denotes a convolutional layer with F different filters, each of size $3 \times 3 \times C$, where C is the depth (i.e. number of channels) of the activations from the previous layer. Padding is 1, and stride is 1.
- POOL2 denotes a 2×2 max-pooling layer with stride 2 (pad 0)
- FLATTEN just turns whatever shape input tensor into a one-dimensional array with the same values in it.
- FC-K denotes a fully-connected layer with K output neurons.

Note: All CONV3-F and FC-K layers have biases as well as weights. **Do not forget the biases when counting parameters.**

Now, we are going to use this network to do inference on a single input. **Fill in the missing entries in this table of the size of the activations at each layer, and the number of parameters at each layer. You can/should write your answer as a computation (e.g.** $128 \times 128 \times 3$**) in the style of the already filled-in entries of the table.**

Layer	Number of Parameters	Dimension of Activations
Input	0	$28 \times 28 \times 1$
CONV3-10	$3 \times 3 \times 1 \times 10 + 10$	$28 \times 28 \times 10$
POOL2	0	$14 \times 14 \times 10$
CONV3-10	$3 \times 3 \times 10 \times 10 + 10$	$14 \times 14 \times 10$
POOL2	0	$7 \times 7 \times 10$
FLATTEN	0	490
FC-3	$7 \times 7 \times 10 \times 3 + 3$	3

(f) Consider a new architecture:

$$\text{CONV2-3} \rightarrow \text{ReLU} \rightarrow \text{CONV2-3} \rightarrow \text{ReLU} \rightarrow \text{GAP (Global Average Pool)} \rightarrow \text{FC-3}$$

Each CONV2-3 layer has stride of 1 and padding of 1. Note that we use **circular padding** (i.e. wrap-around) for this task. Instead of using zeros, circular padding makes it as though the virtual column before the first column is the last column and the virtual row before the first row is the last row — treating the image as though it was on a torus.

Here, the GAP layer is an average pooling layer that computes the per-channel means over the entire input image.

You are told the behavior for an input image with a horizontal edge, $\mathbf{x}_1$ and an image with a vertical edge, $\mathbf{x}_2$:

$$\mathbf{x}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Suppose we knew that the GAP output features when fed $\mathbf{x}_1$ and $\mathbf{x}_2$ are

$$\mathbf{g}_1 = f(\mathbf{x}_1) = \begin{bmatrix} 0.8 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{g}_2 = f(\mathbf{x}_2) = \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix}$$

Use what you know about the invariances/equivariances of convolutional nets to compute the $\mathbf{g}_i$ **corresponding to the following** $\mathbf{x}_i$ **images.**

- $\mathbf{x}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Due to parameters-sharing, convolution has transformation invariance.

$g_3 = f(x_3) = \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix}$

- $\mathbf{x}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

$g_4 = f(x_4) = \begin{bmatrix} 0 \\ 0 \\ 0.8 \end{bmatrix}$.

4. Coding Question: Desigining 2D Filter

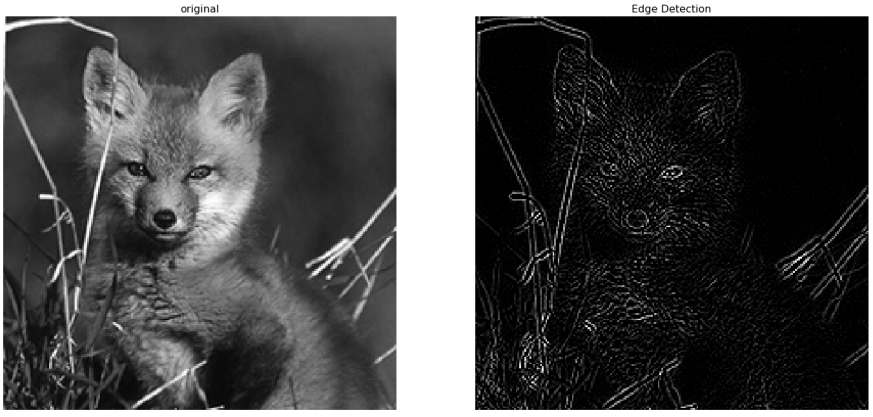
Convolutional layer, which is the most important building block of CNN, actively utilizes the concept of filters used in traditional image processing. Therefore, it is quite important to know and understand the types and operation of image filters.

Look at HandDesignFilters.ipynb. In this notebook, we will design two convolution filters by hand to understand the operation of convolution:

(a) **Blurring filter.**

(b) **Edge detection filter.**

For each type of filter, please **include the image generated by the Jupyter Notebook in your submission to the written assignment**. The image should be added to the PDF document that you will be submitting. Please do *not* submit the Jupyter Notebook to "Homework 3 (Code)".



5. Coding Question: Batchnorm, Dropout and Convolutions

In this assignment, you will implement batch normalization, dropout, and convolutions using NumPy and PyTorch. For this assignment, we recommend using Google Colab as some PCs or laptops may not support GPU-based PyTorch.

Attention: This coding task will take approximately 4 to 6 hours to complete, taking into account the time required for training the model. Please plan accordingly and start early to ensure adequate time for completion.

The assignment consists of three parts:

Implementing Batch Norm and Dropout. Open this url and follow the instructions in the notebook. You will implement the forward and backward propagation of dropout and batch normalization in NumPy. A GPU runtime is not required for this part.

Implement convolution and spatial batch norm. Open this url and follow the instructions in the notebook. You will implement the forward and backward propagation of convolutional layers and spatial dropout in NumPy. A GPU runtime is not required for this part.

Use deep learning framework. Open this url and follow the instructions in the notebook. For this part, you will need to switch to a GPU runtime (details can be found in the notebook). You will implement a convolutional neural network with convolution layers, batch normalization, and dropout using PyTorch and train it on a GPU. You also have the opportunity to improve or design your own neural network.

To submit your completed work, please upload a .zip file to the Gradescope assignment titled "Homework 3 (Code)". The instructions for packaging everything into an archive can be found in the last cell of the notebook of the last part.

Please answer the following question in your written assignment submission:

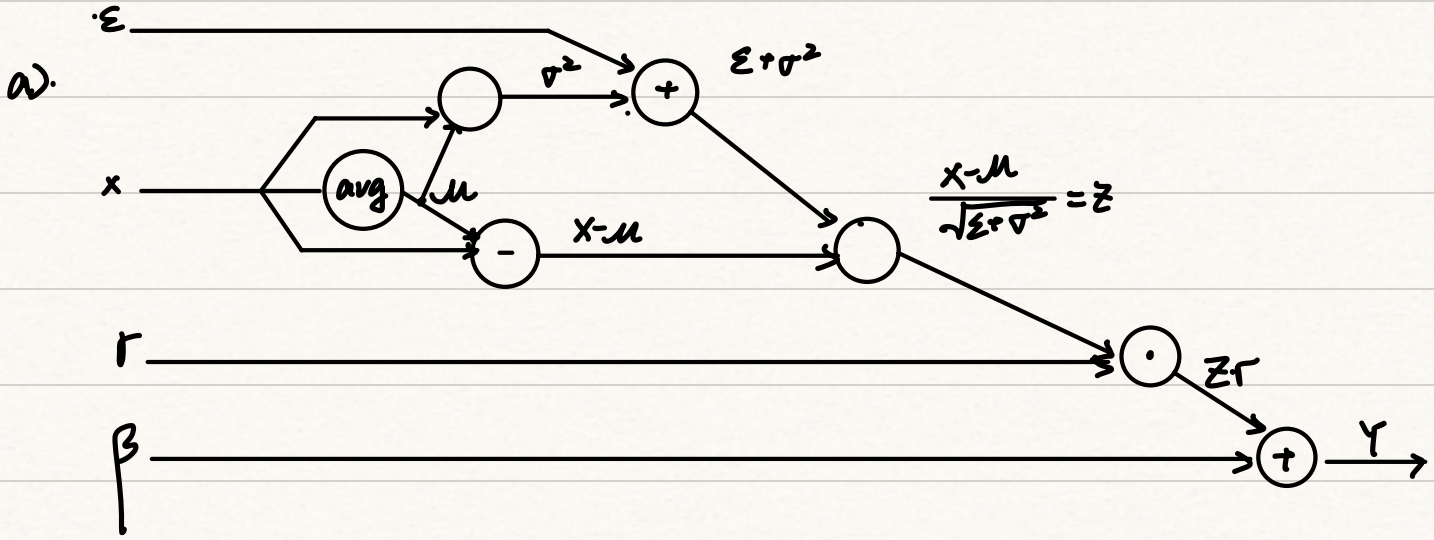
(a) **Draw the computational graph of training-time batch normalization** in your written assignment. In input of the computational graph should be $\mathbf{X}, \gamma, \beta$, the output of the computational graph should be $\mathbf{Y}$, and the intermediate nodes are $\mu, \sigma^2, \mathbf{Z}$.

(b) **(Optional) Derive the closed-form back-propagation of a batch normalization layer (during training).** Include the answer in your written assignment.

Specifically, given $\mathrm{dy}_{i,j} = \dfrac{\partial \mathcal{L}}{\partial Y_{i,j}}$ for every i, j, Please derive $\dfrac{\partial \mathcal{L}}{\partial X_{i,j}}$ for every i, j as a function of $\mathrm{dy}, \mathbf{X}, \mu, \sigma^2, \epsilon, \gamma, \beta$.

(c) **Explain what you see in this experiment. What does it suggest about dropout?**

(d) **Briefly design your neural network design and the procedure of hyperparameter tuning.**



b) $y_{ij} = z\,\gamma_j + \beta_j$

$$z = t_{ij}\left(\frac{1}{n}\sum_{j=1}^{n} t_{ij}^2 + \varepsilon\right)^{-\frac{1}{2}}$$

$$t_{ij} = x_{ij} - u_j = \frac{n-1}{n}x_{ij} - \frac{\sum_{k\neq i} x_{kj}}{n}$$

$$\frac{\partial z}{\partial t_{ij}} = \left(\frac{1}{n}\sum t_{ij}^2 + \varepsilon\right)^{-\frac{1}{2}} - t_{ij}^2\left(\frac{1}{n}\sum t_{ij}^2 + \varepsilon\right)^{-\frac{3}{2}} \cdot \frac{1}{n}.$$

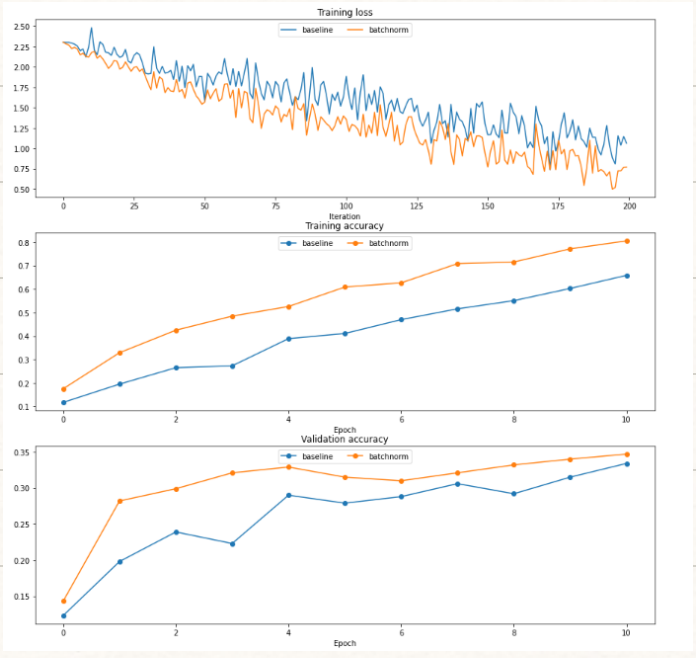
$$\frac{\partial t_{ij}}{\partial x_{ij}} = \frac{n-1}{n}.$$

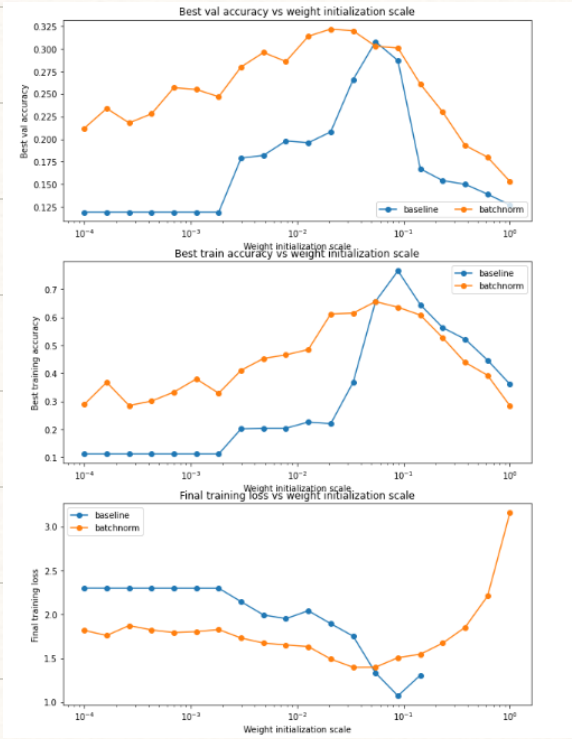
Let $tmp = \left(\frac{1}{n}\sum t_{ij}^2 + \varepsilon\right)^{-\frac{1}{2}}$

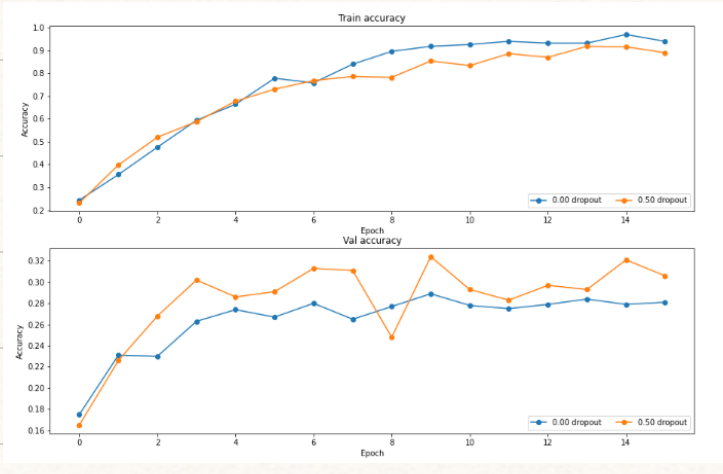
$$\frac{\partial z}{\partial x_{ij}} = tmp - \frac{t_{ij}^2}{n}\,tmp^3.$$

$$\frac{\partial L}{\partial x_{ij}} = \frac{\partial L}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial z} \cdot \frac{\partial z}{\partial t_{ij}} \cdot \frac{\partial t_{ij}}{\partial x_{ij}} = \frac{\partial L}{\partial y_{ij}} \cdot \gamma_j \left[tmp - \frac{t_{ij}^2}{n}\,tmp^3\right] \cdot \frac{n-1}{n}.$$

c).







Using batch normalization improves the performance of the model, i.e., higher acc and lower loss, and weight initialization scale should be set to a moderate value (10^{-1} in this experiments to achieve the best performance.

Using dropout may lead to lower acc. in training time, but is likely to increase the accuracy when validated.

d) Network design:

① 2 FC layers → 1 FC layers.

I tried to go wider at first, but was soon blocked by too many parameters. Based on what we learned. it is the FC layer who owns the largest amount of params. I once found it seemed to make no huge difference between using one FC layer or two FC layer in the end, so I adopted this pattern and merged the two FC layers into one.

② Deeper + Wider.

After removing one FC layer, there is much more space for me to add more layers & channels per layer.

The architecture is in submitted code

- Hyper-parameter tuning

① lower learning rate

② dropout rate ↓.

③ After adding dropout, it is usually the case train-acc < val-acc, so there is no need for us to add regularization.

④ lr-decay ↑.

⑤ epochs ↑.

7. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so w
can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you need to remember it for the self-grade form.**

a) Pytorch : nn module document

b) Name : Xiang Fei

SID : 3038733024.

c) 15h

https://ieeexplore.ieee.org/document/7410480