Introduction to Computer Vision (Spring 2025)
Assignment 2

Release date: April 11, due date: April 26, 2025 11:59 PM

The assignment includes 3 tasks: backpropagating an MLP, implementing batch normalization and training a simple CNN on CIFAR-10. This assignment is fully covered by the course material from Lecture 4 to 7.

The objective of this assignment is to get you familiar with some useful algorithms of deep learning, as well as to establish the concept of how to train and tune a network. We offer starting code for all the tasks and you are expected to implement the key functions of each task. Please read *README.md* before you start.

1 Notices

1. **To submit your homework, please update your personal information in *pack.py*, run the *pack.py* script to compress your code and results and then submit the zip file to course.pku.edu.cn.**

2. Feel free to post in the discussion panel for any questions and we encourage everyone to report the potential improvements of this assignment with a bonus of up to 5 points.

2 Tasks

1. **Backpropagation for an MLP (20 points):**

   Backpropagation is a fundamental technique for training neural networks. In this question, you will experience how to write the backpropagation for a toy MLP to achieve binary classification. Specifically, you will have 10 images from the MNIST dataset and the MLP learns to predict whether the given image is "0" or not.

```
iteration: 1, loss: 9.779285
iteration: 2, loss: 4.179959
iteration: 3, loss: 3.212628
iteration: 4, loss: 2.781939
iteration: 5, loss: 2.521984
iteration: 6, loss: 2.317483
iteration: 7, loss: 2.140868
iteration: 8, loss: 1.984311
iteration: 9, loss: 1.844338
iteration: 10, loss: 1.718717
...
iteration: 41, loss: 0.456483
iteration: 42, loss: 0.444388
iteration: 43, loss: 0.432814
iteration: 44, loss: 0.421728
iteration: 45, loss: 0.411100
iteration: 46, loss: 0.400905
iteration: 47, loss: 0.391119
iteration: 48, loss: 0.381723
iteration: 49, loss: 0.372699
iteration: 50, loss: 0.364031
```

Figure 1: Example results of our BP implementation.

We have already implemented the forward pass including the sigmoid activate function and the cross-entropy loss. Once you have figured out the workflow of the forwarding, try to implement your own

backpropagation. Note that, you are only recommended to use Numpy and without using any loop operations. You can check more details in the slides of lecture 3 and Appendix 4.

2. **Batch Normalization (30 points):**

   Batch normalization is one of the most common modules used in deep learning because it can greatly reduce the difficulty of optimizing a large model. In this section, you should implement batch normalization and evaluate the performance by integrating it with a neural network (the same toy MLP as in Assignment 1). Please complete the *bn.py* and you should expect the loss to decrease as Figure 2 (specific numbers can be different due to the implementation details).

   Note that you are only allowed to use Numpy in this question. *For loops* are also **not allowed**.

   a)[10 points] Implement the Forward Pass for Training

   First, you are required to implement the forward pass of batch normalization. Specifically, given an input $\mathbf{x} \in \mathbb{R}^{B \times C}$, in which B refers to the batch size and C refers to the number of channels, your network should output the correct $\mathbf{y}$ and update the running mean of $\mu \in \mathbb{R}^C$ and $\sigma \in \mathbb{R}^C$ for the purpose of inference.

   b)[10 points] Implement the Backward Pass for Training

   Afterwards, you are required to implement the backward pass of batch normalization. With the saved variables from forwarding, you are required to compute the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$, $\frac{\partial \mathcal{L}}{\partial \gamma}$ and $\frac{\partial \mathcal{L}}{\partial \beta}$.

   c)[10 points] Implement the Forward Pass for Inference

   An important characteristic of batch normalization is the inconsistency between training and inference. In this question, you are expected to perform inference of the given MLP with batch normalization.

```
iteration: 1, loss: 13.90185
iteration: 2, loss: 3.033449
iteration: 3, loss: 2.752287
iteration: 4, loss: 2.656002
iteration: 5, loss: 2.593305
iteration: 6, loss: 2.539740
iteration: 7, loss: 2.489078
iteration: 8, loss: 2.439408
iteration: 9, loss: 2.390089
iteration: 10, loss: 2.34090
...
iteration: 42, loss: 1.020985
iteration: 43, loss: 0.992716
iteration: 44, loss: 0.965264
iteration: 45, loss: 0.938619
iteration: 46, loss: 0.912765
iteration: 47, loss: 0.887688
iteration: 48, loss: 0.863374
iteration: 49, loss: 0.839805
iteration: 50, loss: 0.816967
validation loss: 0.028552
```

Figure 2: A result of our BN implementation. Your results do not have to be exactly the same as ours.

3. **Train a CNN on CIFAR-10 (50 points):**

   This question is designed to give you a taste of how to train and tune neural networks. Here is a pipeline you will experience:

   1. Establish a baseline and make sure the hyperparameters are chosen appropriately.
   2. Identify the problems, *e.g.* overfitting, in this baseline via visualizing and analyzing the curves.
   3. To tackle the problems, either leverage the techniques introduced in the lectures or propose and implement your tricks.

4. Go back to Step 2 until the performance of your model meets your expectations. Sometimes you may need to tune the hyperparameters again.

Specifically, you are required to implement and train a convolution neural network for classification on a small dataset CIFAR-10. Don't worry about not having a GPU since this dataset is small enough to train on CPUs (we can achieve a performance of $> 75\%$ on the validation set within half an hour under the *TOP 1 accuracy* metric with CPUs). You can also use Colab from Google to access free GPUs, as long as you are willing to spend extra time for setup. We provide the starting code and you can follow the steps below for doing the experiment:

a)[10 points] Implement a CNN

A convolution neural network can extract features from the input images $\mathbf{I} \in \mathbb{R}^{B\times H\times W\times 3}$ and predict the probability distribution among the classes $\mathbf{Y} \in \mathbb{R}^{B\times N}$, in which B refers to the batch size and N refers to the number of classes ($N = 10$ for CIFAR-10).

For the first step, you are required to build a CNN in *network.py*. We recommend to leverage the basic modules/layers in PyTorch, *e.g.* torch.nn.Conv2d. There is no specific requirement about the structure of your CNN, but we provide an exemplar architecture in Figure 6 for your information (feel free to ignore it). Note that, you are not allowed to directly use the complete/off-the-shelf CNN modules, *e.g.* torchvision.models.resnet50.

b)[10 points] Implement the Cross Entropy Loss

Apart from the network, an important component of deep learning is the loss function. In this question, you are expected to implement the most popular loss, cross entropy loss, for the classification task. Once you have finished this step, you can run the *train.py* and see the results on the training set and the validation set.

Note that you are not allowed to directly use *torch.nn.CrossEntropyLoss* in your final submission, although you can use it to debug.

c)[10 points] Adjust Learning Rate and Visualize Curves

Appropriate hyperparameters can improve the performance of your model. To investigate hyperparameter tuning, we can draw the curves of the loss function and evaluation metrics to get a feeling about the training progress.

In this question, you are required to draw the **loss curve** and **accuracy curve** on the **training** set and **validation** set under different learning rates (1e-3, 1e-4, 1e-5) during the training process. You should submit a screenshot of your curves as Figure 3. As long as the performance is reasonable, you will be fine.

d)[10 points] Data Augmentation and Visualization

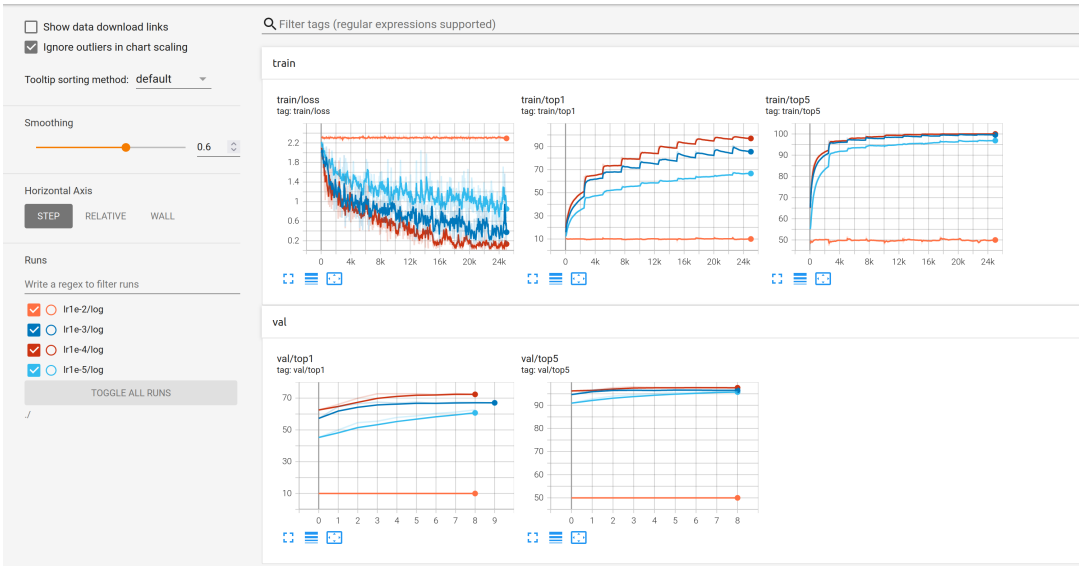

Figure 3: Screenshot of the curves under different learning rates. To save your time, you don't have to run learning rate=1e-2.

In Figure 3, we can find that a large gap between the training accuracy and the validation accuracy, which indicates that the model is suffering from overfitting. This could be a very common phenomena for a small dataset. To alleviate this problem, we can leverage data augmentation.

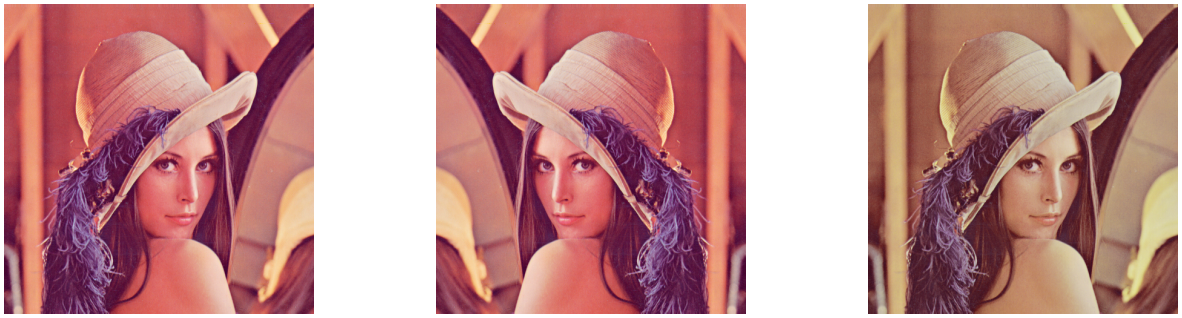

Figure 4: Left: Without augmentation. Middle: Position augmentation. Right: Color augmentation.

In this question, you are required to implement at least two different kinds of data augmentations in *data.py* and save the augmented images for visualization as Figure 4. Since the resolution of the images from CIFAR-10 is too small, you can visualize *Lenna.png* to check your code and submit the augmented images of it. After checking the correctness of your augmentation by visualization, run the *train.py* again to see if the gap between training accuracy and validation accuracy becomes smaller.

Note that, the point of this question is to investigate the effect of the data augmentation. Therefore you are allowed to use any existing publicly available libraries, even in one line if you can find any.

e)[10 points] Further Improve your Network

In d), you have experienced how to identify and resolve the overfitting problem during training neural networks. Furthermore, sometimes we need the model to achieve certain performance bars. In this

question, you are free to improve your own model until the performance meets the bar and your expectation. You should submit a screenshot as Figure 5 to show your achievement.

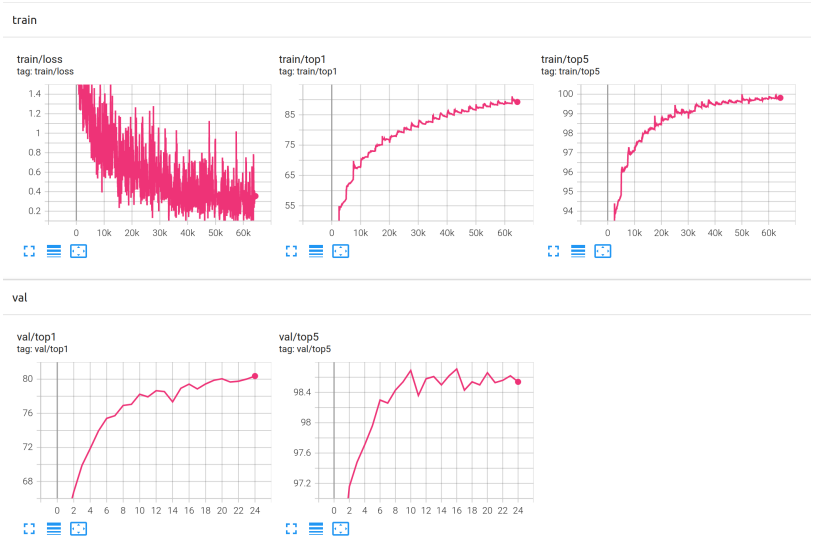

Figure 5: Best performance of my model.

We will grade this question according to the criteria in Table 1. You should expect to achieve it within a few trials even without a GPU (the TA reaches 75% after adding data augmentation without any other tricks) if you are tackling the correct spots.

TOP1 accuracy(%) ↑	Score
$\geq 75\%$	10
$70\% \sim 75\%$	8
$60\% \sim 70\%$	6
$50\% \sim 60\%$	4
$40\% \sim 50\%$	2

Table 1: Grading policy.

Appendix

1. Some helpful libraries for data augmentation.
   - OpenCV-Python, https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

- Pillow(PIL), https://pillow.readthedocs.io/en/stable/
- torchvision.transforms, https://pytorch.org/vision/0.9/transforms.html
- ...

2. Draw curves by TensorboardX https://pytorch.org/docs/stable/tensorboard.html
3. Colab tutorial https://colab.research.google.com/
4. Backpropagation for a Linear Layer from Justin Johnson.

http://cs231n.stanford.edu/handouts/linear-backprop.pdf

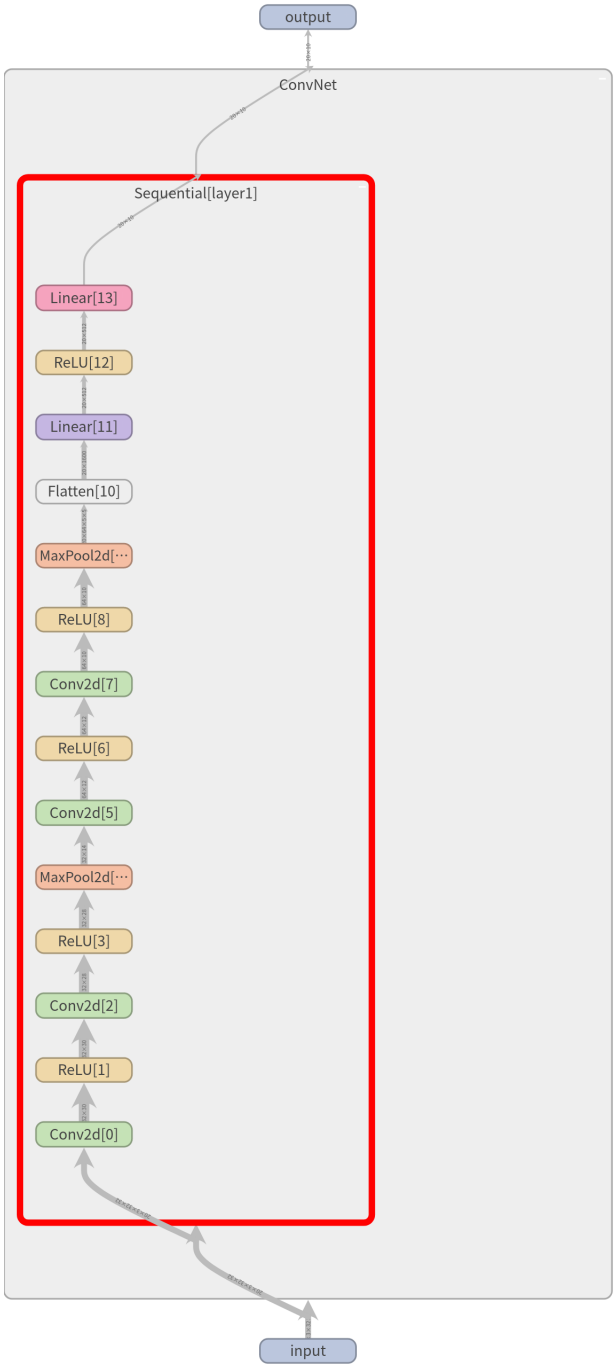

Figure 6: An example of a CNN.