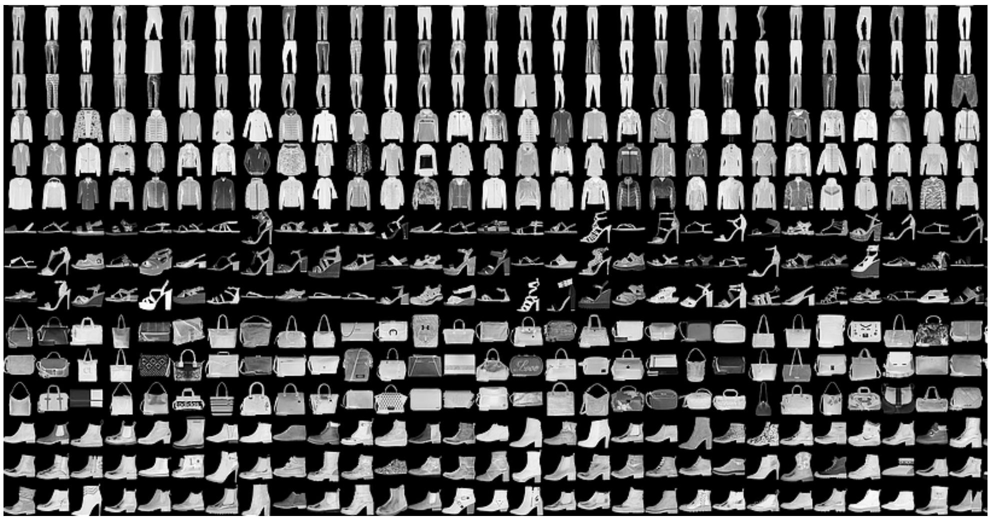
Deep Learning (Homework 1)

Due date : 2025/4/6 23:55:00

1 Feedforward Neural Network (25%)

You are given a dataset sampled from the Fashion-MNIST dataset. This dataset contains 10 classes. In this exercise, you need to implement a feedforward neural network (FNN) model by yourself to recognize images, and use the backpropagation algorithm to update the parameters.



Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Dataset description:

- Training set contained 60000 images with 6000 images collected for each individual class. Test set contained 10000 images with 1000 samples collected for each individual class.
- The images were 28 by 28 in size and were flattened in row-major order into the shape of 784. The labels are integers that indicate the corresponding class. Details of these classes are provided in the above table.
- This dataset is given in the form of numpy array files (.npy) named "train_x.npy", "train_y.npy", "test_x.npy" and "test_y.npy", where "x" indicates images and "y" indicates the corresponding labels.

Please follow the steps below to implement your program:

- Understand how the "forward pass" and "backward pass" in FNN work in accordance with the backpropagation algorithm.
- Both the training and test images need to be normalized (divided by 255).
- Use the cross entropy error function $J(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\log y_k(\mathbf{x}_n, \mathbf{w})$ as the objective function where t_{nk} is the target value, N is the number of samples in a batch and $y_k(\mathbf{x}_n, \mathbf{w}_n)$ is the FNN output.

1. Design a FNN model architecture and use the file of the initial weights and biases "weights.npy". Run the backpropagation algorithm and use the mini-batch SGD (stochastic gradient descent) $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla J(\mathbf{w}^{(\tau)})$ to optimize the parameters (the weights and biases), where η is the learning rate. You should implement the FNN training under the following settings:

    - number of layers: 3
    - number of neurons in each layer (in order): 2048, 512, 10
    - activation function for each layer (in order): relu, relu, softmax
    - number of training epochs: 30
    - learning rate: 0.005
    - batch size: 200
    - **important note**: For 1(a), DO NOT RESHUFFLE THE DATA. We had already shuffled the data for you. Reshuffling will make your results differ from our ground-truth implementation, and any difference will result in reduction of your points. On the same note, when splitting the samples into batches, split them in the given sample order.

    (a) **Plot** the learning curves of $J(\mathbf{w})$ and the accuracy of classification for every 30 iterations, with training data as well as test data, also, **show** the final loss and accuracy values. (10%)

    (b) **Repeat 1(a)** by considering zero initialization for the model weights. And **do some discussion**. (3%)

2. Based on the model in Problem 1, please implement the dropout layers and apply them after the first two hidden layers, i.e. the layers with 2048 and 512 neurons. The dropout rate should be set as 0.2 for both layers. Note that the dropout operation should only be applied in the training phase and should be disabled in the test phase.

    (a) **Train** the model using the same settings in Problem 1 and **repeat Problem 1(a)** (4%)

    (b) Based on the experimental results, how the dropout layers affect the model performance and why? Please **do some discussion**. (2%)

3. Based on the model in Problem 1, please implement mini-batch SGD (stochastic gradient descent). In this problem, we need to reshuffle the data in every batch. Note that the other settings remain the same. Please set the random seed as **42**, and please use **random** library that we have imported.

    (a) **Plot** the learning curves of $J(\mathbf{w})$ and the classification accuracy for every 30 iterations. Please **show** the final values of loss and accuracy. (4%)

    (b) Based on the experimental results, how the process of reshuffling images affect the model performance and why? Please **make some discussion**. (2%)

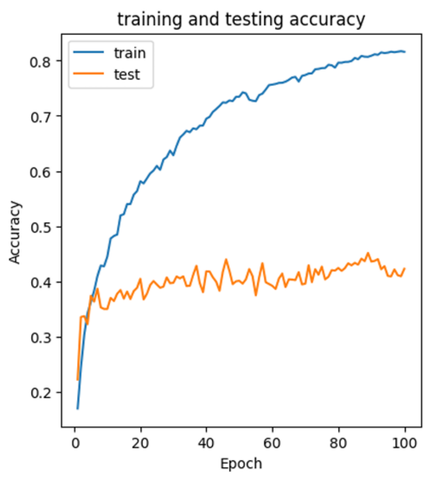
Note:

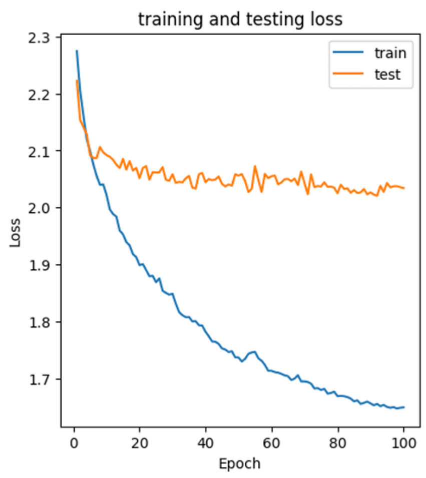
- When coding in Python, be careful to assign the value to variable (mutable vs immutable object). Double check the dimensions of your matrices.
- Normally, when training a deep neural network, you should shuffle training data for each epoch, but for convenience of grading we restrict the data order in 1(a).

2 Convolutional Neural Network (40%)

In this exercise, you will train a convolutional neural network (CNN) for image recognition by using **Oxford 102 Flower Dataset**. This dataset totally has 102 categories. The flowers chosen to be flower commonly observing in the United Kingdom. Each class consists of between 40 and 258 images. Some examples are shown below.

1. Please implement a CNN for image recognition by using **Oxford 102 Flower Dataset** dataset. Your model should include **two convolutional layers** with **batch normalization** applied after each convolutional layer. Then **plot** the learning curves for accuracy and loss of training and test data. (20%)





NOTE: The figures shown above are the example of learning curves for accuracy and loss of training and test data. The actual results might be different.

2. Please replace the batch normalization in the model from problem 1 with layer normalization, then **plot** the learning curves for accuracy and loss of training and test data. Compare the results of using batch normalization and layer normalization and provide the detailed comparison and discussion on the results. (10%)

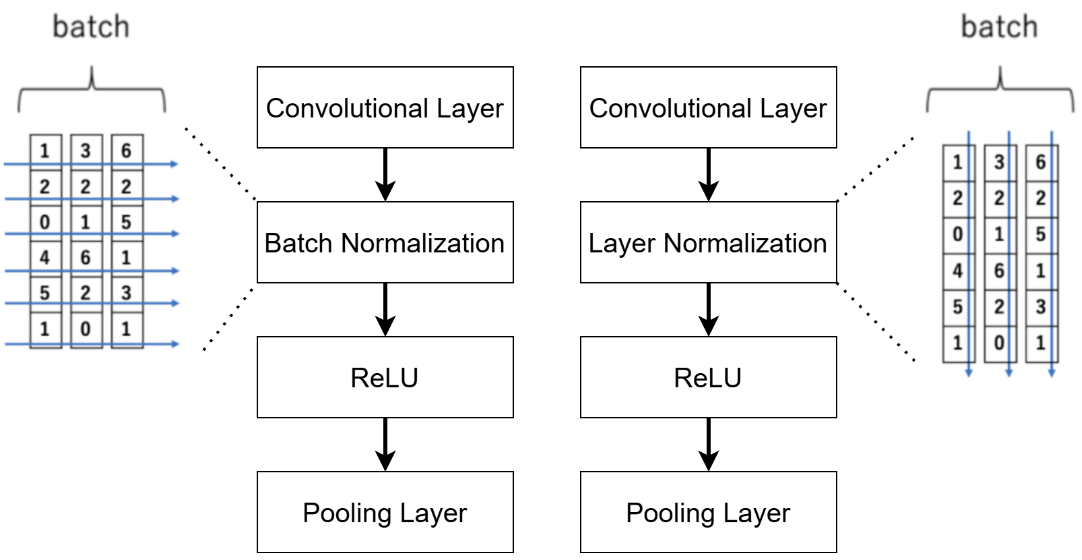
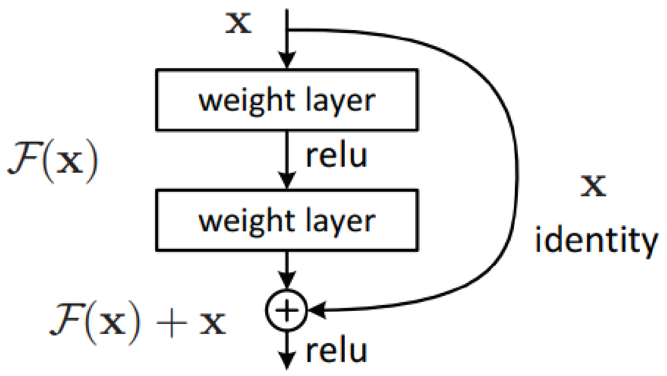


Figure 1: CNN implementation using batch normalization and layer normalization.

3. To deal with a real-world problem, we may stack some additional layers in the Deep Neural Network which results in improvement in classification accuracy. However, it has been found that there is a maximum threshold for the depth of the layers with the traditional convolutional neural network model. The challenge of training a very deep network has been alleviated with the introduction of **ResNet** or residual network.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

(a) Construct a **ResNet** with residual blocks for image recognition and **plot** the learning curves, for accuracy and loss, try to stack more blocks as you can (ResNet-18 is recommended), you can refer to the paper for implementation. (15%)

(b) Remove the identity mapping and repeat Problem (a), then do some comparison and discussion on the results of Problems (a) and (b). Please **describe** in details about what you found. (5%)

NOTE: Please implement the model by yourself, directly load the pre-traind model from pytorch is not allowed.

3 Image Classification Competition (35%)

You will participate in a Kaggle competition. Using the techniques you have learned in the previous sections, try to develop a model that achieves the highest accuracy.

Competition Rules:

- To ensure fairness, please implement your model on the Kaggle platform so that everyone uses the same hardware resources.
- Make sure your model runs within the memory limits of the Kaggle platform.
- If your **accuracy surpasses the baseline**, you will receive a **basic score**.
- The **remaining score** will be determined based on the **competition rankings**.
- You may use additional techniques, but you must clearly explain them.

1. Please join the competition link and aim to achieve the highest accuracy.
    - Score exceeding the baseline. (10%)
    - Public ranking score. (10%)
    - Private ranking score. (10%)
2. Explain your model's design strategy, including the mechanisms you used, the number of layers in your model, the reasons behind your design choices, and provide some **findings** and **discussions** from your experiments as detailed as possible. (5%)

4 Rule

- In your submission, you need to submit two files. And only the following file format is accepted:
    - **hw1_<ProblemNumber>_<StudentID>.ipynb** file which need to contain all the results, codes and reports for each exercise (e.g. **hw1_2_0123456.ipynb**).
- Implementation will be graded by
    - Completeness
    - Algorithm correctness
    - Description of model design
    - Discussion and analysis
- Only Python implementation is acceptable.
- For problem 1, any tools with automatic differentiation are forbidden, such as Tensorflow, PyTorch, Keras, etc. You should implement backpropagation algorithm by yourself.
- For problem 2 & 3, you should use PyTorch to implement the model, other high level deep learning APIs and pre-trained weights are forbidden.
- DO NOT PLAGIARIZE. (We will check program similarity score.)