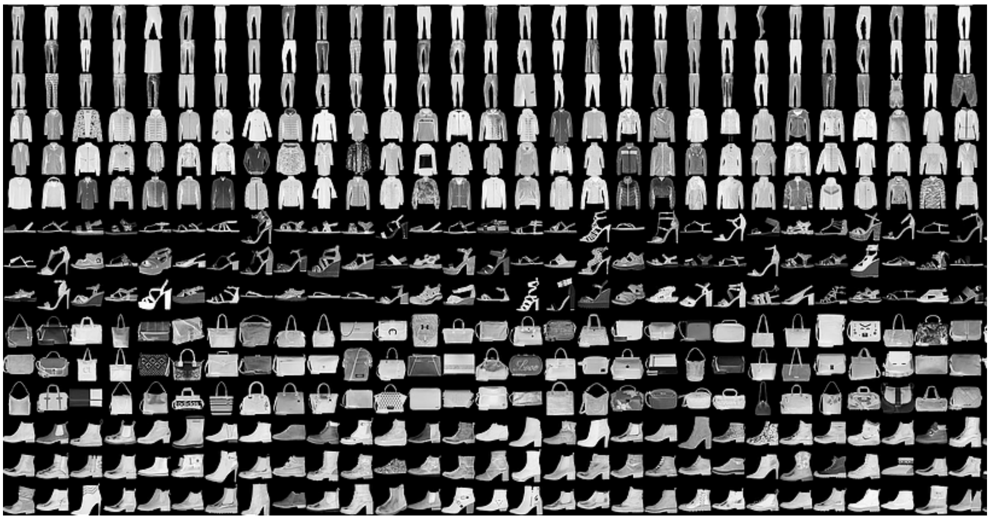
Deep Learning (Homework 1)

Due date : 2022/4/1 23:55:00 (Hard Deadline)

1 Feedforward Neural Network (60%)

You are given a dataset sampled from the Fashion-MNIST dataset. This dataset contains 5 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	Trouser
1	Coat
2	Sandal
3	Bag
4	Ankle boot

Dataset description:

- Training set contained 30000 images with 6000 images collected for each individual class. Test set contained 5000 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, 5
   - activation function for each layer (in order): relu, relu, softmax
   - number of training epochs: 20
   - learning rate: 0.001
   - 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 50 iterations, with training data as well as test data, also, **show** the final loss and accuracy values. (20%)

   (b) **Repeat 1(a)** by considering zero initialization for the model weights. And **do some discussion**. (8%)

2. Based on the model in 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 1 and **repeat 1(a)** (8%)

   (b) Based on the experimental results, how the dropout layers affect the model performance and why? Please **do some discussion**. (8%)

3. In 1 and 2, the model is trained by updating the weights $\mathbf{w}$ using their gradients, but in the process of backpropagation we can also obtain the gradients of the input $\mathbf{x}$. What happens if we fix $\mathbf{w}$ and update the input $\mathbf{x}$ instead? Using the trained model in 1, please randomly generate a batch of (256) noise images as the initial input and perform backpropagation to update these input $\mathbf{x}$ while fixing $\mathbf{w}$. You should use the learning rate 0.1 and update for 5 iterations. For the labels, you can randomly assign integer values in the range [0, 4].

   (a) **Show** the classification accuracy of these noise images with respect to the random labels before and after the updates. (8%)

   (b) Based on the result, **explain your findings**. What do these updated inputs tell us about the trained model? (8%)

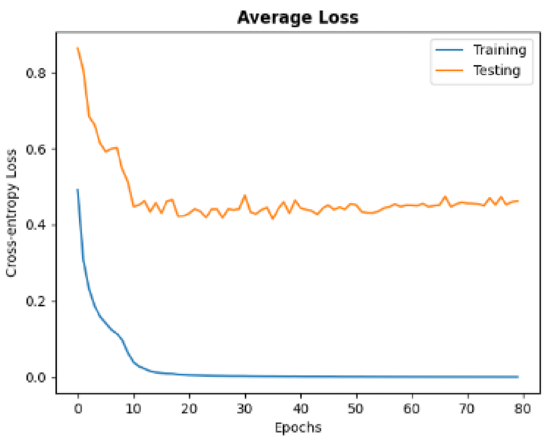
Note:

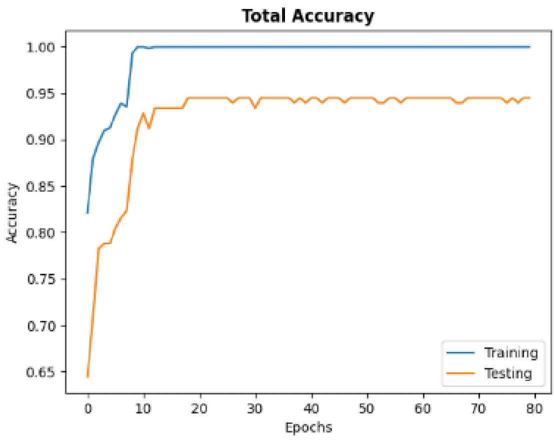
- When coding in Python, be careful to assign the value to variable (mutable vs immutable object). Double check the dimensions of your matrices.
- Normally, when training a deep neural network, you should shuffle the data for each epoch, but for convenience of grading we restrict the data order in 1(a).

2 Convolutional Neural Network (40%)

In this exercise, you will construct a convolutional neural network (CNN) for image recognition by using **Stanford Dogs Dataset**. This dataset consists of 1600 dog images from 8 categories.

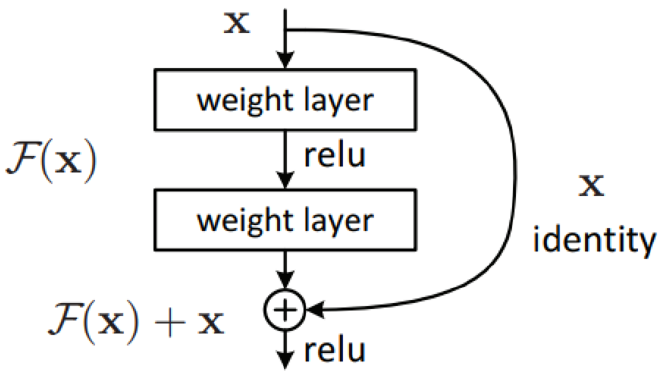
1. Please implement a CNN for image recognition by using **Stanford Dogs Dataset**,then **plot** the learning curve and the accuracy rate of training and test data. (20%)





NOTE: Figure above shows an example. The result might be different.

2. In order to deal with a real-world problem, we may stack some additional layers in the Deep Neural Network which results in the improved accuracy and performance. But it has been found that there is a maximum threshold for depth with the traditional convolutional neural network model. The problem of training a very deep network has been alleviated with the introduction of **ResNet** or residual network.



(a) Construct a **ResNet** with residual blocks for image recognition and **plot** the learning curve, accuracy rate, 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 (a), then do some discussion on the results of (a) and (b). Please **describe** what you found. (5%)

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

3 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, you should use PyTorch to implement the model, other deep learning APIs are forbidden.
- DO NOT PLAGIARIZE. (We will check program similarity score.)