DDA3020 Homework 3 Solution

Due date: Nov 28, 2023

Instructions

- The **deadline** is 23:59:00, Nov 28, 2023.
- The weight of this assignment in the final grade is 15%.
- **Electronic submission**: Turn in solutions electronically via Blackboard:
  - For writing part, submit a single PDF file **HW3_Written_part.pdf**
  - For programming part, submit 2 pdf reports and the executable codes as well as the required model parameters for the two questions. See details in programming part.
  - In total, you will upload 4 items to BB: 3 PDF files and a ZIP containing your codes and Neural Network model parameters.
- Note that **late submissions** will result in discounted scores: 0-24 hours → 80%, 24-120 hours → 50%, 120 or more hours → 0%.
- Answer the questions in English. Otherwise, you'll lose half of the points.
- Collaboration policy: You need to solve all questions independently and collaboration between students is **NOT** allowed.
- If you have any questions concerning this homework, feel free to reach out to TA Dong QIAO (dongqiao@link.cuhk.edu.cn) for writing part and Wenrang ZHANG (223040237@link.cuhk.edu.cn) for programming part. You're also welcome to physically visit them during their office hours with your questions.

1 Written Problems (50 points) - Inquiry: dongqiao@link.cuhk.edu.cn

Note that 50 Points have been splitted into 10 points for tree, 15 points for MLP, 15 pts for CNN, and 10 points for performance evaluation.

1.1. (Tree, 10 points) Consider the following table which predicts whether a student is likely to be overweight. Use crossentropy as criteria to construct the classification decision tree that predicts whether or not a student is likely to be overweight. Show your step of the computation.

Stu_ID	**Gender**	**Hyperlipidemia**	**Unhealthy Diet**	**Exercises**	**OverWeight**
1	boy	yes	no	yes	yes
2	boy	yes	yes	no	yes
3	girl	no	yes	no	yes
4	boy	no	no	yes	no
5	girl	yes	yes	yes	yes
6	boy	no	yes	yes	no

Let us use some labels for the simplicity. That is, G - Gender; H - Hyperlipidemia; U - Unhealthy Diet; E - Exercises; O - Overweight

$$H(O) = -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6} = 0.9183 \qquad \text{(1 pts)}$$

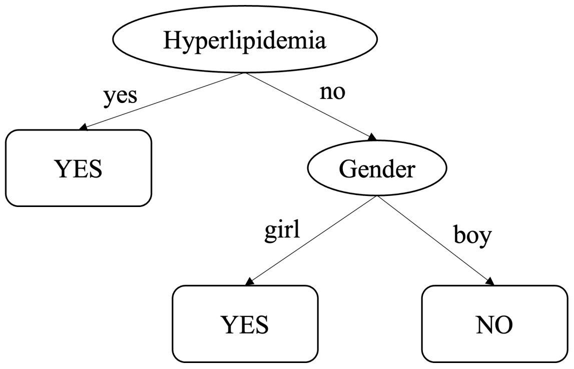
$$H(O|G) = \frac{4}{6}H(\frac{2}{4},\frac{2}{4}) + \frac{2}{6}H(\frac{2}{2},\frac{0}{2}) = 0.6667 \qquad G(G) = H(O) - H(O|G) = 0.2516 \qquad \text{(1 pts)}$$

$$H(O|H) = \frac{3}{6}H(\frac{3}{3},\frac{0}{3}) + \frac{3}{6}H(\frac{1}{3},\frac{2}{3}) = 0.4591 \qquad G(H) = H(O) - H(O|H) = 0.4591 \qquad \text{(1 pts)}$$

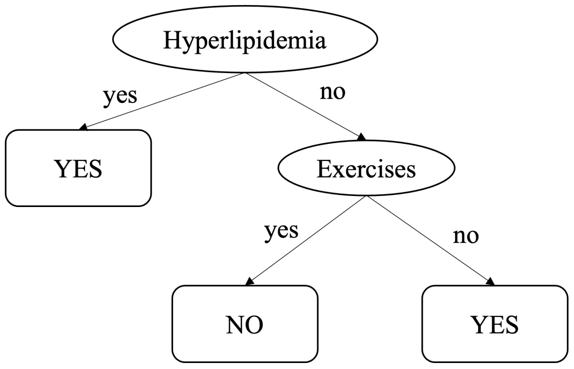
$$H(O|U) = \frac{4}{6}H(\frac{3}{4},\frac{1}{4}) + \frac{2}{6}H(\frac{1}{2},\frac{1}{2}) = 0.8742 \qquad G(U) = H(O) - H(O|U) = 0.0441 \qquad \text{(1 pts)}$$

$$H(O|E) = \frac{4}{6}H(\frac{2}{4},\frac{2}{4}) + \frac{2}{6}H(\frac{2}{2},\frac{0}{2}) = 0.6667 \qquad G(E) = H(O) - H(O|E) = 0.2516 \qquad \text{(1 pts)}$$

Since $G(H) > G(G) = G(E) > G(U)$, we select **Hyperlipidemia** as the best attribute as the root. ... 1 pts



(a) Alternative 1



(b) Alternative 2

Figure 1: Decision Tree

... 4 pts for either of them.

1.2. (Backpropagation for MLP, 10 points) Consider the following classification MLP with one hidden layer:

$$\begin{aligned}
\boldsymbol{x} &= \text{input} \in \mathbb{R}^D \\
\boldsymbol{z} &= \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}_1 \in \mathbb{R}^K \\
\boldsymbol{h} &= \text{ReLU}(\boldsymbol{z}) \in \mathbb{R}^K \\
\boldsymbol{a} &= \boldsymbol{V}\boldsymbol{h} + \boldsymbol{b}_2 \in \mathbb{R}^C \\
\mathcal{L} &= \text{CrossEntropy}(\boldsymbol{y}, \mathcal{S}(\boldsymbol{a})) \in \mathbb{R}
\end{aligned}$$

where $\boldsymbol{x} \in \mathbb{R}^D$, $\boldsymbol{b}_1 \in \mathbb{R}^K$, $\boldsymbol{W} \in \mathbb{R}^{K\times D}$, $\boldsymbol{b}_2 \in \mathbb{R}^C$, $\boldsymbol{V} \in \mathbb{R}^{C\times K}$, where D is the size of the input, K is the number of hidden units, and C is the number of classes. Draw the computational graph of forward pass and derive the specific form $\boldsymbol{\psi}_i, i = 1, \ldots, 5$ of gradients w.r.t. the parameters and input.

$$\begin{aligned}
\nabla_{\boldsymbol{V}}\mathcal{L} &= [\frac{\partial \mathcal{L}}{\partial \boldsymbol{V}}]_{1,:} = \boldsymbol{\psi}_1(\boldsymbol{u_2}, \boldsymbol{h}) \in \mathbb{R}^{C\times K} \\
\nabla_{\boldsymbol{b}_2}\mathcal{L} &= (\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2})^T = \boldsymbol{\psi}_2(\boldsymbol{u}_2) \in \mathbb{R}^{C} \\
\nabla_{\boldsymbol{W}}\mathcal{L} &= [\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}]_{1,:} = \boldsymbol{\psi}_3(\boldsymbol{u_1}, \boldsymbol{x}) \in \mathbb{R}^{K\times D} \\
\nabla_{\boldsymbol{b}_1}\mathcal{L} &= (\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1})^T = \boldsymbol{\psi}_4(\boldsymbol{u}_1) \in \mathbb{R}^{K} \\
\nabla_{\boldsymbol{x}}\mathcal{L} &= (\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}})^T = \boldsymbol{\psi}_5(\boldsymbol{W}, \boldsymbol{u}_1) \in \mathbb{R}^{D}
\end{aligned}$$

where the gradients of the loss *w.r.t.* the two layers (logit and hidden) are given by the following:

$$\begin{aligned}
\boldsymbol{u}_2 &= \nabla_{\boldsymbol{a}}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}})^T = (\boldsymbol{p} - \boldsymbol{y}) \in \mathbb{R}^C \\
\boldsymbol{u}_1 &= \nabla_{\boldsymbol{z}}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}})^T = (\boldsymbol{V}^T\boldsymbol{u}_2) \odot H(\boldsymbol{z}) \in \mathbb{R}^K
\end{aligned}$$

where $H(a) = \mathbb{I}(a > 0)$ is the heaviside function. Note that, in our notation, the gradient (which has the same shape as the variable with respect to which we differentiate) is equal to the Jacobian's transpose when the variable is a vector and to the first slice of the Jacobian when the variable is a mtrix.

$$\nabla_{\boldsymbol{V}}\mathcal{L} = [\frac{\partial \mathcal{L}}{\partial \boldsymbol{V}}]_{1,:} = \boldsymbol{\psi}_1(\boldsymbol{u_2}, \boldsymbol{h}) = \boldsymbol{u}_2\boldsymbol{h}^T \in \mathbb{R}^{C\times K} \qquad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{b}_2}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2})^T = \boldsymbol{\psi}_2(\boldsymbol{u}_2) = \boldsymbol{u}_2 \in \mathbb{R}^{C} \qquad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{W}}\mathcal{L} = [\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}]_{1,:} = \boldsymbol{\psi}_3(\boldsymbol{u_1}, \boldsymbol{x}) = \boldsymbol{u}_1\boldsymbol{x}^T \in \mathbb{R}^{K\times D} \qquad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{b}_1}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1})^T = \boldsymbol{\psi}_4(\boldsymbol{u}_1) = \boldsymbol{u}_1 \in \mathbb{R}^{K} \qquad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{x}}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}})^T = \boldsymbol{\psi}_5(\boldsymbol{W}, \boldsymbol{u}_1) = \boldsymbol{W}^T\boldsymbol{u}_1 \in \mathbb{R}^{D} \qquad \text{(2 pts)}$$

1.3. (Connection of neural network to logistic regression, 5 points) Consider a neural network for a K class outcome that uses cross entropy loss. If the network has no hidden layer, show that the model is equivalent to the multinomial logistic model.

- For a 1-layer K-class neural network, we have

$$\hat{\boldsymbol{y}} = \text{Softmax}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

where $\hat{y}_k = \text{Softmax}_k(\boldsymbol{a}) = \frac{e^{a_k}}{\sum_{k'=1}^K e^{a_{k'}}}$ for $a_k = \boldsymbol{w}_k^T\boldsymbol{x} + b_k$, $k = 1, \dots, K$.
The model fitting is to minimize the cross entropy:

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{n=1}^N \sum_{k=1}^K y_k^{(n)} \log \hat{y}_k^{(n)} \qquad \text{(2 pts)}$$

- For a multinomial logistic regression:

$$p(y = k|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{e^{a_k}}{\sum_{k'=1}^K e^{a_{k'}}}$$

where $\boldsymbol{a} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^K$. We compute the maximum likelihood estimate (MLE) by minimizing the negative log-likelihood (NLL).

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{N} \log \prod_{n=1}^N \prod_{k=1}^K (\mu_k^{(n)})^{y_k^{(n)}} = -\frac{1}{N} \sum_{n=1}^N \sum_{k=1}^K y_k^{(n)} \log \mu_k^{(n)} \qquad \text{(3 pts)}$$

where $\mu_k^{(n)} = p(y^{(n)} = k|\boldsymbol{x}^{(n)}; \boldsymbol{\theta})$.
We can see that the above two models are equivalent.

1.4. (CNN, 10 points) Consider the convolutional network defined by the layers below. The input shape is $32 \times 32 \times 3$ and the output is 10 neurons. Consider layers:

$$\text{Conv5}(10) + \text{Maxpool}_2 + \text{Conv5}(10) + \text{Maxpool}_2 + \text{FC10}$$

where

- Conv5(10): 10 filters with each size $5 \times 5 \times D$, where D is the depth of the activation volume at the previous layer, stride = 1, padding = 2;
- Maxpool_2: 2×2 filter, stride = 2, padding = 0;
- FC10: A fully-connected layer with 10 output neurons.

(a) (5 pts) Compute the shape of activation map of each layer.

(b) (5 pts) Compute the total number of parameters of each layer.

The answer is as in Table 1.

Table 1: Understanding and calculating the number of parameters in CNN

NO.	**Layer**	**Activation Shape**	**# Parameters**	Mark
1	Input Layer	(32,32,3)	0	-
2	Conv5(10)	(32,32,10)	(5*5*3 + 1)*10 = 760	1 pts
3	Maxpool2	(16,16,10)	0	1 pts
4	Conv5(10)	(16,16,10)	(5*5*10 + 1)*10 = 2510	1 pts
5	Maxpool2	(8,8,10)	0	1 pts
6	FC10	(10,1)	8*8*10*10 + 10 = 6410	1 pts

1.5. (Dropout, 5 points) To expect a good predictive model. We want it to peform well on unseen data. Classical generalization theory suggests that we should close the gap between train and test performance. Dropout is a standard regularization technique for training neural networks. The method is called dropout because we literally drop out some neurons during training as in Fig. 2. Throughout training, on each iteration, standard dropout consists of zeroing out some fraction of the nodes in each layer before calculating the subsequent layer. This prevents co-adaptations where features are only useful when present in the presence of other features. It can dramatically reduce overfitting and has been very widely used.

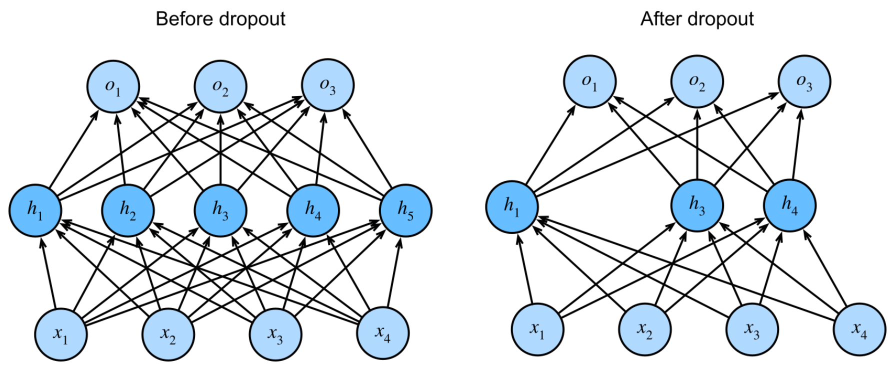


Figure 2: MLP before and after dropout

(a) (1 pts) Give some intuitive reason why dropout prevents complex co-adaptation of the hidden units.

- Each unit must learn to perform well even if some of the other units are missing at random;
- This prevents the units from learning complex, but fragile, dependencies on each other.

References:

[1] Kevin Patrick Murphy, Probabilistic Machine Learning: An Introduction, `https://probml.github.io/pml-book/book1.html`

(b) (2 pts) In standard dropout regularization, one zeros out some fraction of the nodes in each layer and then debiases each layer by normalizing by the fraction of nodes that were retained (not dropped out). In other words, with dropout probability p, each intermediate activation h is replaced by a random variable h' as follows:

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$

Explain why we have a proportional coefficient $\frac{1}{1-p}$;

Note that dropout is used only during training: We just want the outputs of neurons at test time to be identical to their expected outputs at training time. By design, the expectation remains unchanged, i.e., $\mathbb{E}[h'] = h$. That is, it replaces an activation h with a random variable h' with expected value h.

References:

[1] Dive into deep learning, `https://d2l.ai/chapter_multilayer-perceptrons/dropout.html`

[2] CS231n Convolutional Neural Networks for Visual Recognition, `https://cs231n.github.io/neural-networks-2/`

(c) (2 pts) Suppose we apply dropout to the activation $h^{(\ell)}$ to get the output $h^{(\ell+1)}$ of the dropout layer, *i.e.*, $h^{(\ell+1)} = h^{(\ell)} \odot \text{mask}$, derive the gradient $\frac{\partial \mathcal{L}}{\partial h^{(\ell)}}$ as a form of chain rule.

Backward propogation over the dropout layer is rather simple to calculate because we are dealing with element-wise multiplication.

$$\frac{\partial \ell}{\partial h^{\ell}} = \frac{\partial \ell}{\partial h^{\ell+1}} \odot \frac{\partial h^{\ell+1}}{\partial h^{\ell}} = \frac{\partial \ell}{\partial h^{\ell+1}} \odot \text{mask}$$

Hint: You can check the following materials:

- Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, `https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf`
- Stanford. CS231n: Dropout, `https://cs231n.github.io/neural-networks-2/`

1.6. (Assessing AUC and Performance of a Binary Classifier, 10 points) Given a 2-d dataset of 5 positive and 5 negative samples, and consider a simple linear model with a sigmoid function $f(x) = \sigma(w_1x_1 + w_2x_2)$ where $w_1 = 0.5$ and $w_2 = -0.1$, please complete the following tasks:

$$\text{Positive samples} : \begin{bmatrix} 5 & 5 \\ 3 & 8 \\ -1 & 8 \\ 5 & 4 \\ -1 & -2 \end{bmatrix} \quad \text{Negative samples} : \begin{bmatrix} -5 & 1 \\ 2 & -2 \\ -1 & 1 \\ 5 & -10 \\ -10 & -9 \end{bmatrix}$$

(a) (2 pts) Compute the predictions for all data points in the dataset using the provided model $f(x)$;

$$\begin{bmatrix} \hat{y}_+ \\ \hat{y}_- \end{bmatrix} = \begin{bmatrix} 0.8808 & 0.6682 & 0.2142 & 0.8909 & 0.4256 \\ 0.0691 & 0.7685 & 0.3543 & 0.9707 & 0.0163 \end{bmatrix}$$

Grading criteria: 1 mark for positive samples; 1 mark for negative samples.

(b) (3 pts) Assume the threshold is 0.5, evaluate the performance of the model by calculating the following evaluation scores:

- Accuracy
- Precision
- Recall
- F1 Score
- Confusion Matrix

Since the predicted labels are

$$\begin{bmatrix} \hat{y}_+ \\ \hat{y}_- \end{bmatrix} = \begin{bmatrix} + & + & - & + & - \\ - & + & - & + & - \end{bmatrix},$$

we first list the basic info as follows.

$$TP = 3 \qquad FN = 2$$
$$FP = 2 \qquad TN = 3$$

- The accuracy is

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{3+3}{3+3+2+2} = \frac{6}{10} = 0.6$$

- The Precision is

$$\text{Accuracy} = \frac{TP}{TP + FP} = \frac{3}{3+2} = \frac{3}{5} = 0.6$$

- The Recall is

$$\text{Accuracy} = \frac{TP}{TP + FN} = \frac{3}{3+2} = \frac{3}{5} = 0.6$$

- The F1 Score is

$$\text{F1 Score} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} = \frac{2 * 0.6 * 0.6}{0.6 + 0.6} = \frac{0.72}{1.2} = 0.6$$

- The confusion matrix as

Table 2: Confusion matrix

	$\hat{P}$	$\hat{N}$
P	3	2
N	2	3

Grading criteria: 1 mark for 1 correct answer; 2 marks for 2-3 correct answers; 3 marks for 4-5 correct answers.

(c) (3 pts) Construct a ROC curve for the model as follows

- The list of threshold is $[0, 0.2, 0.4, 0.6, 0.8, 1.0]$;
- Hand-drawn is required;
- x-axis is FPR and y-axis is TPR;
- Use Fig. 3 as a reference.

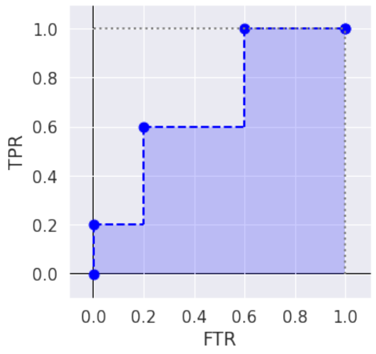


Figure 3: Example of step-wise ROC curve

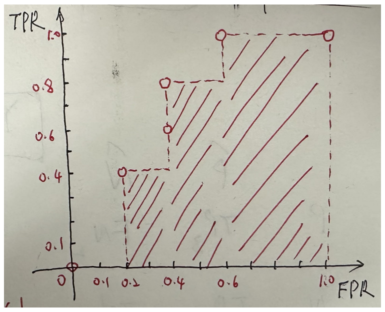


Figure 4: ROC

Grading criteria: 3 marks only for correct answer; 0 mark for wrong answer.

(d) (2 pts) Calculate the Area Under the ROC Curve, *i.e.*, AUC.

$$\text{AUC} = 0.2 \times 0.4 + 0.2 \times 0.8 + 0.4 \times 1 = 0.64 \qquad \text{(2 pts)}$$

2 Programming (50 points) - Inquiry: 223040237@link.cuhk.edu.cn

2.1 Tree (15 points)

The goal of this exercise is to practice tree-based methods on Diabetes data-set[1] and predict if a patient has diabetes. You are required to conduct some analysis and submit a complete report as well as your executable code. Your report should contain :

- Preliminary data analysis. Some visualization of the data distribution/variability.
- Some data processing if you find it necessary. Data splitting with ratio 0.33.
- One implementation of uni-variate tree method for the classification problem.
- One implementation of an ensemble model for the classification problem. You can choose any ensemble model that you have heard of.
- Result visualizations, comments on the results, your personal thinking, etc.
- *Optional*: can you think of a way to show how your tree splits the whole space?

You are free to use sklearn and other published packages. Your work is graded according to the readability, the logic and the degree of completion presented in your report. The coherence of you code is also taken into account.

Submission (-15 points if you do not submit code)

1. A PDF report **HW3_Programming_Tree.pdf**. Your statements should be well supported by figures or code explanation.
2. Your executable code script or notebook.

2.2 Neural Networks (35 points)

In this exercise, you will build your own fully-connected Neural Network (MLP, multi-layer perceptron) as well as your own Convolutional Neural Network. You will apply them on MNIST data-set to perform hand-written digit classification.
You are provided with a notebook guide **HW3_Programming_NN_guide.ipynb** where PyTorch framework is adopted. You have 2 options:

1. Follow the guide and fill in the blanks. Then, export the notebook to pdf file as your report. You need to have either *pickle* or *torchvision* in your python environment to load the data-set.

[1] https://www.kaggle.com/datasets/piyushborhade/diabetes-dataset

2. If you are significantly more familiar with some other deep learning framework (e.g. TensorFlow). You can perform freely on this work. Please refer to **part 2: simple NN** and **part 4: CNN** of the guide for required procedures and outputs. You will be graded according to the standard given in the guide, and your grade for 2.2 will be re-scaled with $\frac{35}{25}$.

Note that:

- Do not modify the data-sets that the TA splitted for you in the guide (The train set is the first 5500 samples of the original MNIST train set; the validation set is the last 500 samples of the original MNIST train set; the test set is the MNIST test set with 10000 samples).
- If you obtain some bonus points, those points will ameliorate or saturate your grade of this exercise (2.2) exclusively (not the whole programming part nor the whole homework).
- If you really struggle with the **PyTorch Basics** part, you can ask the TA for solution directly:
  - subject: Demand for PyTorch Basics solution; content: your name + student ID

  You can still do the bonus part to obtain at most 32 points on this exercise.
- If any of your classmates asks to copy your code, please make sure that they have already asked the TA for the Basics solution. They are supposed to show you a screen shot of the TA's reply ("copy") to their demand.
- No model score (3) nor performance score (4.5) for **part 4: CNN** if you directly load any published CNN model to solve the problem.

Submission (-35 points if you do not submit code; no performance scores if you do not include your model parameters)

1. A PDF report **HW3_Programming_NN.pdf**.
2. Your executable code script or notebook, your best NN model, your best CNN model.

Discussion for computational cost:
If you have NVIDIA card on your PC, you can use **cuda** to accelerate the calculation. However, theoretically, 2 layers of CNN are enough to have at least 97% test accuracy for this exercise.
If your PC has no worse than Intel Core i5-8250U Processor (as had the TA's PC in 2019) and your code is well done, the program shall be guaranteed to run fast.
Conventionally, you are encouraged to construct not too complicated CNN to save computational cost, since MNIST is a relatively simple problem. However, as long as your code works out for you, no point will be removed if your CNN is huge.