Hong Kong University of Science and Technology
COMP 4211: Machine Learning
Spring 2019

Programming Assignment 2
Due: 11 April 2019, Thursday, 11:59pm

1 Objective

The objective of this programming assignment is threefold:

1. To acquire a better understanding of deep learning by using a public-domain software package called `PyTorch` aided by `tensorboardX`, the `PyTorch` version of `TensorBoard`.

2. To evaluate and analyze the effectiveness of two convolutional neural network (CNN) classifiers for image classification: one trained from scratch and the other trained using pretrained weights.

3. To evaluate and analyze the effectiveness of a convolutional autoencoder (CAE) by making use of pretrained weights.

4. To write up a report that shows your understanding of the above-mentioned models and the experiment results.

2 Dataset

For image classification and reconstruction, the EMNIST balanced dataset is provided where each handwritten character image is of size $1\times28\times28$ and belongs to one of 47 classes. Only half of the original dataset is provided to account for those who do not have enough computational resources. As a result, the training and test sets contain a total of 52,640 and 13,160 images, respectively.

3 Major Tasks

The major tasks of this assignment are as follows:

1. Install `PyTorch` and `tensorboardX`, which will be used for building two CNN classifiers and a CAE decoder.

2. Build a CNN classifier from scratch.

3. Build a CNN classifier on top of the provided pretrained CAE encoder.

4. Build a CAE decoder on top of the provided pretrained CAE encoder.

5. Compile and summarize all of the above in a single, compact and concise report.

Each of these tasks is elaborated in the following subsections.

3.1 `PyTorch` and `tensorboardX` Installation

Please follow the installation instructions provided in Tutorial 5.

For this assignment, it suffices to use only CPUs for `PyTorch` with Python 3.x. You may also use GPUs to which you might have access.

Optionally, you may also install `matplotlib` for plotting figures. However, note that using `tensorboardX` as shown in the tutorial materials is **required** for this assignment. Also, using other high-level machine learning frameworks including Keras is not acceptable as we intend to provide you an opportunity to go deeper to liberate the full power and flexibility of `PyTorch` and `tensorboardX`.

3.2 CNN Classifiers

A CNN classifier can be seen as an encoder followed by a predictor. While the former takes an image as input and produces its image features as output, the latter takes those features as input and produces class probabilities as output.

3.2.1 Learning from Scratch

You need to build a CNN classifier from scratch with the structure as shown in Table 1 and Table 2 below.

Layer	# Filters	Kernel Size	Stride	Input Shape	Output Shape
Conv. 2d	4	3	1	1×28×28	4×26×26
Conv. 2d	8	3	2	4×26×26	8×12×12
Conv. 2d	16	3	2	8×12×12	16×5×5
Max Pool	-	3	1	16×5×5	16×3×3
Conv. 2d	32	3	1	16×3×3	32×1×1

Table 1: CNN encoder

Layer	# Units	Input Shape	Output Shape
Linear	H	32	H
Linear	47	H	47

Table 2: CNN predictor (a fully-connected layer is referred to as a 'linear' layer in `PyTorch`)

After each convolutional layer and the first linear layer in the encoder and predictor, respectively, **rectified linear units** (**ReLU**) should be used for activation while after the max pooling layer the **logistic** (i.e., **sigmoid**) function should be used.

3.2.2 Learning from Pretrained Encoder Weights

The encoder model has been pretrained with a denoising CAE. The model itself along with its pretrained weights is given in the file `pretrained_encoder.pt`. You can load the model with `torch.load` and build your own predictor on top of it. For your convenience, all the parameters in the pretrained encoder have been set up as `requires_grad=False`. Your own predictor should have the same structure as shown in Table 2.

3.2.3 Loss to be Minimized

For both CNN classifiers (i.e., learning from scratch and using the pretrained encoder), cross-entropy loss between the logits (i.e., output of the final linear layer of the predictor) and the labels should be used for training.

3.2.4 Hyperparameter Selection (Validation Phase)

The CNN classifiers have a few hyperparameters that need to be set. You are required to use **holdout** validation with the hyperparameter settings as shown in Table 3. Therefore, a total of $2 \times 3 = 6$ settings should be validated.

For each setting, you need to run at least 10 epochs and present the results. Make sure that no image from the test set should be included during the validation phase. For those who are not yet familiar with holdout validation, please refer to the Background Knowledge section provided later in this project description.

Hyperparaemter	Candidates
H	32, 64
(Optimizer, Learning Rate)	(ADAM, 0.001), (SGD, 0.1), (SGD, 0.01)

Table 3: Hyperparameter settings

3.2.5 Testing Phase

In addition to the cross-entropy loss, you should compute the top-1 and top-3 accuracy values by training the CNN classifiers with the **whole** training set and testing them against the test set by choosing the optimal hyperparameters with the previous holdout validation. You are required to repeat five times with each run having at least 10 epochs. In the report, you are required to show the average and standard deviation of the three metrics, which will be further explained in the Report section later.

3.3 Image Reconstruction with CAE using Pretrained Encoder from Denoising CAE

A CAE is an autoencoder consisting of convolutional layers. The same pretrained encoder is provided, and you are required to reconstruct the images by building your own decoder on top

of the pretrained encoder from a denoising CAE. For your understanding, some examples of reconstructed images are provided below in Figure 1.

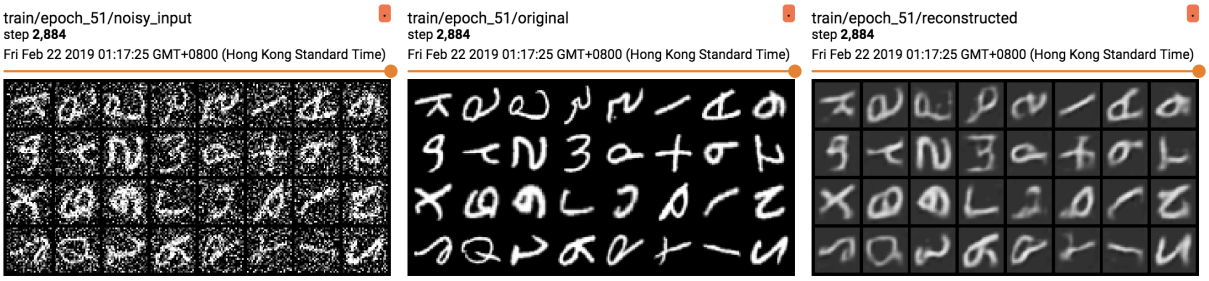

Figure 1: Images reconstructed with the pretrained denoising CAE

Note that you are **not required** to train a denoising CAE. Instead, you can use the images purely as given (without any Gaussian noise).

3.3.1 CAE Decoder Structure

The structure of the decoder is provided below in Table 4. After each ConvTranspose2d layer before the last one, you should use the **ReLU** activation function; after the last ConvTranspose2d layer, you should use the **logistic** (**sigmoid**) function.

Layer	# Filters	Kernel Size	Stride	Input Shape	Output Shape
Conv. Transpose 2d	16	3	1	32×1×1	16×3×3
	8	3	1	16×3×3	8×5×5
	8	3	2	8×5×5	8×11×11
	4	3	1	8×11×11	4×13×13
	1	4	2	4×13×13	1×28×28

Table 4: CAE decoder

3.3.2 Loss to be Minimized

You are required to use the mean squared error (MSE) as the loss function. Note that this is equivalent to viewing each pixel of the image as a Gaussian variable while each image is represented as a multivariate Gaussian with a constant diagonal covariance matrix.

3.3.3 Hyperparameter Selection (Validation Phase)

For CAE, you need to select the optimal hyperparameter setting from the candidates provided in Table 5.

Hyperparaemter	Candidates
(Optimizer, Learning Rate)	(ADAM, 0.001), (SGD, 0.1), (SGD, 0.01)

Table 5: CAE Hyperparameter settings

Similar to the CNN validation phase, you need to run at least 10 epochs. In addition to the MSE loss, you are also required to present some reconstructed images for each setting. Also, make sure that no image from the test set is included during the validation phase.

For CAE validation, if the reconstructed images do not look feasible, you should **not** choose the corresponding hyperparameter as optimal since the purpose of CAE is to learn "good" image features.

3.3.4 Testing Phase

You should compute the MSE loss by training the CAE using the **whole** training set and testing it against the test set with the optimal optimizer setting chosen from the validation phase. You are **not required** to run multiple times for this. Instead, you are **required** to present the reconstructed images in the report.

3.4 Report

For each of the tasks mentioned above, the requirements expected in the report are further explained below.

3.4.1 CNN Classifiers

1. For each classifier, take a screenshot of the code where you build your CNN networks, including the loading of the pretrained encoder for the second CNN classifier. This is to prevent any from copying others' code. Note that you are required to build **your own** networks. Therefore, no trick is acceptable (e.g., copying from others, using any kind of trick).

2. Present the holdout validation results that contain the cross-entropy loss (against the validation set) for each hyperparmeter setting. Note that you should report the "optimal" loss (i.e., minimum loss), not the loss after the final epoch.

3. Provide the reasoning for choosing a particular set of hyperparameters based on your validation results.

4. During the testing phase, for each metric (cross-entropy loss, top-1 accuracy, and top-3 accuracy), present the average and standard deviation (std.) of the corresponding values (against the test set). Note that the mean and std. should be computed with the "optimal" values. For example, the average of five top-1 accuracy values should be presented, where each of the accuracy values is the maximum test accuracy (**not** the one after the final epoch).

5. During the testing phase, randomly select only one run out of the total of five runs (you can do for all the five runs, which is optional and does not count for scores). For that run, after each epoch, the three metrics should be recorded with `tensorboardX.SummaryWriter` for **both** the training and test sets. After that, for each metric, the learning curves for both the training and test sets should be presented in one diagram. Therefore, you are required

to present a total of three diagrams (for the three metrics), each of which contains two learning curves (for training and test sets). Note that all of this can be done conveniently with `tensorboardX`, and you are **required** to use it.

6. After the testing phase, provide the analysis on the performance of the CNN classifiers based on the empirical test results. Please focus on the use of the pretrained encoder, and indicate its effectiveness based on: 1) training time; 2) number of epochs; and 3) the three performance metrics (cross-entropy loss, top-1 accuracy, top-3 accuracy).

3.4.2 CAE with Pretrained Encoder

1. Take a screenshot of the code where you build your CAE, including the loading of the pretrained encoder. Again, this is to prevent any from being dishonest, and no trick is acceptable.

2. Present the holdout validation results that contain the "optimal" (minimal in this case) MSE loss (against the validation set) for each hyperparameter setting. Note that "optimal" loss does not necessarily refer to the one after the final epoch.

3. Present some examples of reconstructed images (against the validation set) from the validation phase. Those images indicate the quality of CAE, and therefore a hyperparameter setting that has a poor quality of reconstructed images should not be selected as optimal even though it might show the best MSE loss. Note that recording the reconstructed images can also be done conveniently with `tensorboardX.SummaryWriter`.

4. Provide the reasoning for choosing a particular hyperparameter based on your validation results.

5. During the testing phase, report the best MSE loss (against the test set) after training the CAE against the whole training set and testing against the test set. Note that "best" loss does not necessarily refer to the loss after the final epoch.

6. During the testing phase, after each epoch, MSE loss for **both** the training and test sets should be recorded with `tensorboardX.SummaryWriter`, and the two learning curves with respect to the MSE loss should be presented in one diagram.

7. Present some examples of the reconstructed images (from the test set) at the "best" epoch at which the "test" MSE loss was the minimum.

3.5 Important Notes

For your convenience, a number of important points are listed below:

1. Note that the provided train / test splits for EMNIST are customized (they do not correspond to the original train / test splits). Therefore, please read the sample code provided and make sure to use the provided train / test splits (do not set the `save` argument to `True` since it will make another train / test split indices).

2. For all the tasks, there is no restriction on the mini-batch size. Rather, a mini-batch size

of at least 32 is recommended to facilitate the speed for training.

3. You are not required to schedule the learning rates using `torch.optim.lr_scheduler`.

4. If there is any training failure (e.g., loss becomes NaNs, etc.), you are required to present the learning curves and provide the reasons for the optimization failure.

5. For both validation and testing phases, you are **required** to shuffle the training set in the beginning of each epoch. Note that this can be automatically done with `torch.utils.data.DataLoader`, and be sure to include the option when you use it for the training sets.

6. For holdout validation, **never** use any part of the test set.

7. For holdout validation, set the train / validation split ratio to 4:1 (i.e., 80% of the original training set is used as the training set while the rest, 20% of the original training set, is used as the validation set).

8. You are required to complete all the tasks as described above. That is, you should show the code and the report for the architectures and experiments described here. Additionally, you are free to add implementation of your own network architecture that might boost the performance of the CAE and/or the CNN (again, those are optional).

9. For those who are unfamiliar with `PyTorch` and `tensorboardX`, you are strongly recommended to follow the tutorials available online and/or review the course materials.

10. For the grading, besides the explicitly mentioned criteria, you are also expected to pay attention to the fundamentals of machine learning in practice. For example, it has been emphasized multiple times in the instructions above that the optimal metric values should be reported, rather than the ones after the final epoch.

4 Provided Files

The provided ZIP file includes `pretrained_encoder.pt`, `train_test_split.npz`, and `pa2_sample_code.py`.

1. `pretrained_encoder.pt` is a file that could be loaded with `torch.load`. Once it is loaded, you can use the pretrained encoder model by calling `data['model']`, where `data` refers to the loaded instance. The pretrained parameters are already set to `requires_grad=False`, so that you can build your own model on top of it and train your model without any further setup.

2. `train_test_split.npz` contains the customized train / test split indices. You do not need to modify anything inside it.

3. `pa2_sample_code.py` is a file that contains two functions: `get_datasets` and `show_image`. To load the customized EMNIST splits, you only need to call `get_datasets()`. Do not set the function argument `save` to `True` since it will modify your train / test splits. You can read the code under the `if __name__=='__main__':` statement to see the usage of them.

5 Background Knowledge

Note that this is only to provide some useful ideas of background knowledge in a TA's words.

5.1 Holdout Validation

It is a technique to select an optimal set of hyperparameters. For example, a multilayer perceptron (MLP) with a single hidden layer would have a variable number of hidden units. To do this, a set of candidates is decided, and you have to choose from them based on validation.

For holdout validation, you first partition the given training set into two sets with a manually determined split ratio, which are called training and validation sets. Note that the term 'training' is used twice, which may cause confusion. Then for each candidate set of hyperparameters, you train a model and validate (test) it against the validation set.

For the MLP example, you may choose a hidden layer size H from the set $\{32, 64\}$. After partitioning the training set into the training and validation sets with ratio 4:1 (80% of the original training set samples go to the training set while the remaining 20% belong to the validation set, with the partition done randomly), you train your model with $H = 32$ and $H = 64$ separately. Note that the same partitions are used for the two candidate values.

In short, holdout validation could be thought of as a simple version of cross validation.

5.2 Adam and SGD Optimization methods

They refer to two of the most popular optimization methods while SGD is an acronym for stochastic gradient descent. For details, you are recommended to read the original papers or some other materials.

Note that those methods at times do not work well when the learning rates are set too high, especially for SGD.

5.3 ConvTranspose Layer (a.k.a. Deconvolution)

The transposed convolution operation can be thought of as a reverse operation of convolution. It is usually used for up-sampling as an alternative to other methods (e.g., bilinear, nearest-neighbor up-sampling methods). For detailed explanation, you may refer to a number of materials online, such as those in blogs (e.g., `https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0`).

6 Assignment Submission

Assignment submission should be done electronically using the Course Assignment Submission System (CASS) as before:

`https://cssystem.cse.ust.hk/UGuides/cass/student.html`

There should be two files in your submission with the following naming convention required:

1. **Report** (with filename `report`): in PDF format.
2. **Source code and a README file** (with filename `code`): all necessary code for running your program as well as a brief user guide for the TA to run the programs easily to verify your results, all compressed into a single ZIP file. The data should not be submitted to keep the file size small.

When multiple versions with the same filename are submitted, only the latest version according to the timestamp will be used for grading. Files not adhering to the naming convention above will be ignored.

7 Grading Scheme

The maximum scores for different tasks are shown in Table 6. All the criteria are categorized based on their purposes. Each criterion is marked with either (R) or (R/C), where R and C refer to the report and the code, respectively. You can earn points for a criterion only when it is satisfied in all the corresponding submissions. For example, you can earn 3 points for providing the validation metric (cross-entropy loss) results for CNN from scratch only when you write code for them and present them in the report.

Late submission will be accepted but with penalty. The late penalty is deduction of one point (out of a maximum of 100 points) for every minute late.

Categories	Subcategories	CNN (scratch)	CNN (pretrained)	CAE (pretrained)	Sum
Network Building	Network Screenshots / Codes (R/C)	5	5	5	15
Val. Phase	Val. Metric(s) (R/C)	3	3	3	9
	Val. Reconstructed Images (R/C)	-	-	15	15
	Reason(s) for Hyperparam. Selection (R)	3	3	5	11
Testing Phase	Test Metric(s) (R/C)	7	7	4	18
	Test Metric Learning Curve(s) (R/C)	7	7	4	18
	Test Reconstructed Images (R/C)	-	-	6	6
Analysis	Analysis of Scratch vs. Pretrained (R)	4	4	-	8
	Sum	29	29	42	100

Table 6: Grading Scheme. R: Report, C: Code. You can earn a score from an R/C subcategory only when the item is present in both report and code.

8 Academic Integrity

Please read carefully the relevant web pages linked from the course website.

While you may discuss with your classmates on general ideas about the assignment, your submission should be based on your own independent effort. In case you seek help from any person or reference source, you should state it clearly in your submission. Failure to do so is considered plagiarism which will lead to appropriate disciplinary actions.