Application of Autoencoders*

Ke Chen

To carry out this assessed coursework, you will need the Python notebook, `AE_app.ipynb`, which contains code to get you started with each of the assignments, and the several data files required by this coursework. The files, `MNIST1_train.npz` of 3,000 instances and `MNIST1_test_1.npz` of 100 instances contain handwritten digit images selected from the MNIST dataset[1]. Those data files are required by the assignments in Part 1. `PATCHES.npy` is an image patch dataset[2], which contains 100,000 image patches of 8×8 size to be used in Part 2. `binaryalphadigits.npz` is a dataset used in Part 3. This dataset[3] contains images of 20×16 size corresponding to 36 alpha-digit symbols (A-Z and 0-9) where there are 39 instances for each symbol. This dataset has been pre-split into training and test subsets with the code provided in the notebook (you must keep this pre-split setting for doing this coursework). Also, you are provided with two python files: `Network.py` and `RBM.py`, which enable you to implement various autoencoders and the hybrid learning strategy required by this coursework. All above are available in a zipped file on BlackBoard alongside this document.

In general, autoencoders (AEs) are one of the most important methods used in representation learning. In this coursework, you are asked to apply the provided implementations in Python to real datasets for data compression, spatial filter learning and visualization.

Supportive Software

To do this coursework, you are provided with our own implementation of deep neural networks in Python: neural network module, `Network.py`, and restricted Boltzmann machine (RBM) module, `RBM.py`. `Network.py` enables you to carry out traditional AE, denoising AE (DAE) and sparse AE (SAE) required for tasks specified in the following assignments, while `RBM.py` allows you to apply RBM to relevant tasks as described in the assignments. Both modules allow you to construct AE or RBM objects in a similar manner. In order to use AEs or RBM, you need to create a neural network or RBM object first by using one of the provided constructors properly. Note that to avoid the stochastic effect caused by a random initialisation, a constant randomness seed has always been used in any random initialisation so that the results produced by a neural network for multiple trials should be always identical on the same data under the same settings.

To create a fully connected feedforward neural network object, the constructor is as follows:

```
Network(layer_units,activations=["activation"])
```

*__Assessed Coursework__: the deadline and requirements can be found at the end of this document.

[1]The information on the MNIST dataset is available: `http://yann.lecun.com/exdb/mnist/`

[2]This dataset was adapted from the dataset provided by G. Hinton on his homepage: `http://www.cs.toronto.edu/~hinton/data/patches.mat`

[3]This dataset was adapted from the following benchmark dataset: `https://www.tensorflow.org/datasets/catalog/binary_alpha_digits`.

where `layer_units` is an array of integers used to specify a neural network architecture with input, hidden and output layers along with the number of units in those layers. For instance, the array "[10,2,10]" indicates a neural network of input, one hidden and output layers where there are 10, 2 and 10 units in input, hidden and output layers, respectively. array "[10,8,4,1]" specifies an MLP of two hidden layers and there are 10, 8, 4 and 1 units in input, the first hidden, the second and output layers, respectively. You can use this argument to specify a fully-connected neural network of any arbitrary architecture you want.
The second argument, `activation`, can be either a string or a list of strings of the same length as `layer_units`. When this argument is set as a string, it means that the specified activation function is applied to all the layers. The options of activation function are "`sigmoid`", "`relu`", "`tanh`", "`linear`". When the argument is set as a list of strings of the same length as `layer_units` where each entry corresponds to the activation function used in a specific layer apart from the input layer where no activation function is needed. For instance, the argument is specified with the list of ["`sigmoid`", "`sigmoid`", "`linear]`", meaning that the neural network has sigmoid units in two hidden layers and the linear units in output layer.

To create a RBM object, the constructor is as follows:

```
RBM(n_vis,n_hid=1024, use_gaussian_visible_sampling=False,
use_gaussian_hidden_sampling=False, use_sample_vis_for_learning=False)
```

where `n_vis` and `n_hid` are used to specify the number of visible and hidden units in the RBM, respectively. The following two arguments specify whether the hidden or visible units use Gaussian or Bernoulli distribution for sampling. The last argument allows you to choose a way for collecting the statistics needed for learning[4].

During the object creation, all the weights are initialised automatically. Afterwards, you have to set hyperparameters used in the optimiser with a function for neural networks (AEs in this coursework) or RBM:

```
Network.set_lr(lr,lr_decay=0.0,momentum=0.0,weight_decay=0.0)
```

or

```
RBM.set_lr(lr,lr_decay=0.0,momentum=0.0,weight_decay=0.0)
```

where, in order, the arguments correspond to learning rate, learning rate decay, momentum and weight decay penalty, respectively.

Finally, you can use a fit function to train a neural network (AEs in this coursework) or RBM:

```
Network.fit(x,y_true,x_val,y_true_val,batch_size,epochs,patience)
```

or

```
RBM.fit(x, x_val,batch_size,epochs,patience)
```

where `x`, `x_val` are training and validation datasets and have to be organised in the numpy matrix format where each row refers to the feature vector of a data point. `batch_size`, `epochs` specify the number of training instances in a batch and the maximum number of epochs for training. `patience`

[4]For details, see Section 3.3 in the RBM guide: `http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf`.

is a hyperparameter used to specify a period measured usually in epochs for early stopping. For example, if `patience` is set to 25 epochs, early stopping will occur if **no** performance improvement on validation dataset in last 25 training epochs. If `patience=-1`, then this early stopping criterion will be disabled and the training will be terminated by the maximum number of epochs specified in the `epochs` argument.

To see the complete summary of all the learnable parameters in a trained neural network after applying `Network.fit`, you are provided with a function, `Network.get_summary()`.

For test, you can use the `Network.predict(x)` and `Network.evaluate(x,y_true)` functions to achieve the prediction of `x` and the validated performance of a trained neural network. For RBM, you can use the function: `RBM.reconstruct(x,force_sample_visible=True)` to produce the input reconstruction. The second argument specifies whether the reconstructed entity yielded from hidden to visual layer is a realisation via sampling (`True`) or in a probability format (`False`).

More functions are provided for you to do different assignments. The details of those functions can be found in the relevant assignments.

To help you familiar with the provided supportive software, a complete example on a synthetic XOR dataset is offered in the notebook where you should be able to see how to use the appropriate functions to specify and train a neural network.

Key Points

In order to complete the assignments in this coursework, you must pay careful attention to several key points as follows:

- As a good practice, data pre-processing via normalisation is often applied before training AEs. **You are asked to look into this by yourself.** It is important for you to consider the nature of the data and the effect of different normalisation methods used for data pre-processing.
- One can find out that the provided training functions include resetting the normal seed that helps the markers replicate your results. That means that re-running the training functions (which include randomness) and network initialisation with the same hyperparameters always yields the same results. **You must NOT retrain the network without the weight re-initialisation unless the lecture notes demand doing so.**
- In all the assignments, some hyperparameter values have been fixed but other hyperparameters need to be tuned for their optimal value. For all the fixed hyperparameter values, **you must keep those unchanged** in your experiments. For those that you needs to tune, **you must comment out the code used for searching for the optimal hyperparameter values to avoid running the code for a long time and leave only the part where you show your results obtained with the optimal hyperparameter values and the training functions in your answer notebook. Failing to do so may incur a loss of marks.** Please strictly adhere to the above rule so that the notebook running-time does not go into the scale of hours. As a hint, planning your optimisation beforehand could save you a lot of time!

Once again, all the above key points must be taken carefully when you work on this coursework. **Violation of the above rules or missing those key points may incur a loss of marks!**

1 Data Compression

It is well known that AEs of a bottleneck code layer can fulfill data compression. In this work, you are asked to apply three different AEs, traditional AE, DAE and RBM, to the MNIST subset for data compression.

In general, the performance of data compression is evaluated compression ratio defined as p/d, d and p are the dimensions of raw data and its code (low-dimensional representation of raw data), respectively, and the information loss in reconstruction measured by the mean squared error (MSE). In **Assignments 1-3** described below in this section, you are asked to train different shallow AEs[5], respectively, with 3,000 training images in `MNIST1_train.npz` on different code dimensions (the number of hidden units) in the range, {40,80,160,320}, for data compression. Then, you can test your trained models on 100 images in `MNIST1_test_1.npz`.

Assignment 1 [3 marks] Apply the traditional AE of tied weights to the MINST data files described above for data compression. In your experiment, set `batch_size=100, momentum=0.0, lr=1.0, patience=25` and `epochs=150` (stop training when either of two termination conditions is satisfied) in common for different code dimensions. To specify the tied weights in the traditional AE, you can use the function: `Network.tie_layer_weights(layer_1_index, layer_2_index)`. In your case, set `layer_1_index=1` and `layer_2_index=3`. It is worth clarifying that the function for tied weights can only be applied to one-pair of layers in a network.

Assignment 2 [3 marks] Apply the denoising AE (DAE) of tied weights to the MINST data files described above for data compression. In your experiment, use the same manner for specifying the tied weights in the DAE and the same hyperparameters given in **Assignment 1**. For training, add Gaussian noise sampled from $\mathcal{N}(0, 0.1)$ (zero mean, 0.1 standard deviation) to generate noisy data. **Hint**: you need to clip noisy data to ensure its value range to be the same as that of clean data.

Assignment 3 [3 marks] Apply the Gaussian-Bernoulli RBM to the MINST data files described above for data compression. In your experiment, specify the Gaussian-Bernoulli RBM by setting the parameter `use_sample_vis_for_learning=False`, set the hyperparameters `lr=0.01, momentum=0.5, epochs=1,500` and keep others identical to those given in **Assignment 1**.

For **each** of those three assignments described above, in your answer notebook, (a) describe how you train the AE with a provided function on the training dataset in a mark-up cell; (b) display the MSEs on 100 test images of AEs with a heat map (each row corresponding to a digit label) for the code dimension that leads to the minimum averaged MSE. To generate and display a heat map, you are provided a function, `plot_results(model,x_test,hidden_units)`, which returns the averaged MSE on all the images in `x_test` where `model` is a trained AE with `Network.fit` or `RBM.fit` function and `hidden_units` refers to a code dimension (e.g., `plot_results(NN,x_test,160)` indicates an trained model, `NN`, that has 160 hidden units); and (c) display the reconstructed images of 100 test instances created with the best AE of the minimum averaged MSE by arranging images in a 10 grid where each row corresponds to a digit label.

Assignment 4 [1 mark] In your answer notebook, draw a plot of $\log_2(\text{Compression Ratio})$ versus

[5] For details of different shallow AEs, see "Shallow Autoencoder" lecture note.

$\log_2$ (Averaged MSE Error) (compression ratio in x-axis and averaged MSE error in y-axix) on all test results yielded by traditional AE, DAE and RBM. Based on results shown in this plot, state which AE performs the best in your answer notebook.

Assignment 5 [2 marks] As an exploratory assignment, you are asked to significantly improve its performance of your chosen RBM reported in **Assignment 3** with the **same** number of hidden units in any manner you can figure out. In your answer notebook, it is essential to describe your idea clearly and provide the code and the solid experimental/computational evidence against those achieved in **Assignment 3** for justification of your idea.

2 Spatial Filter Learning

In visual perception, sparse coding forms spatial filters, which plays a crucial role in extracting non-trivial features for visual recognition. Sparse AE (SAE) provides a manner to learn such spatial filters encoded in its weights from images. In this work, you are asked to apply the SAE with KL-sparsity penalty[6] to the image patch dataset for spatial filter learning. To set up the hyperparameters, ρ and λ, in SAE, you are provided a function, `Network.layers[i].enable_sparsity(lambda,rho)`, in the `Network.py` module. This function allows for applying the KL-sparsity penalty to the ith hidden layer of the sigmoid units with the specified hyperparameters ($i = 1$ corresponding to the first hidden layer will be used in this coursework).

For visualisation of weights required in the following assignment, you are provided functions in the `Network.py` module. To access weights and bias associated with the ith layer ($i = 1$ corresponds to the first hidden layer) in a trained neural network, you can use the provided functions, `Network.layers[i].w` and `Network.layers[i].b`.

Assignment 6 [5 marks] Apply the SAE of **non-tied** weights with the encoder architecture, **64-100**, to `PATCHES.npy` for spatial filter learning. In your experiment, set `batch_size=500`, `momentum=0.5`, `lr=1.0`, `patience=10` and `epochs=60` (stop training when either of two termination conditions is satisfied). In your experiment, you need to tune two hyperparameters in the KL-sparsity, ρ and λ, in the given ranges, $0.01 < \rho < 0.4$ and $0.001 < \lambda < 1.0$, to learn proper spatial filters from data. In your answer notebook, (a) implement a function that normalises the grey-level intensity of pixels to the range [0,1] within **each** patch image; (b) complete the implementation of two functions, `display_mean_activation` and `display_filters`, to visualise the mean activation and the weights associated with hidden units in the SAE; i.e., learned spatial filters, where the function must display the weight vector associated with each of all 100 hidden neurons in the 8×8 image format and arrange 100 spatial filter images in a 10×10 grid; (c) with your implemented function, `display_filters`, display weight vectors associated with all 100 hidden neurons. You need to display 3 results produced by using the different ρ and λ values (explicitly give two hyperparameter values linked to each result) and point out which one corresponds to the best learned spatial filters. Note that using a grid search throughout ρ and λ range is a poor way to find the optimal values. As an exploratory work, you are asked to figure out a more efficient strategy that considers the information about sparsity learned from the lecture and apply it in finding a good λ for a given ρ choice. It is essential for you to describe and justify your strategy in the answer notebook.

[6] For details of SAE and KL-sparsity penalty, see the "Autoencoder" lecture note.

Hints: (1) make a function to display the mean activation of hidden units, which allows you to check the sparse status of the hidden layer, and (2) use the display functions made by yourself to help you choose a λ value that can make **all** of the mean activations close to a specified ρ value.

3 Visualisation

Deep autoencoders (AEs) have turned out to be an effective nonlinear dimension reduction method. In this work, you are asked to apply deep AEs described in the lecture note to the alpha-digit data file, `binaryalphadigits.npz` that has been pre-split into training and test subsets, for visualisation. In this work, you need to devise an experimental procedure used to train a particular deep AE with the provided functions and measure its reconstruction of the data. Also, you are asked to compare a deep AE trained with the hybrid learning strategy[7] based on RBMs to another deep AE trained with random initialisation directly with stochastic gradient descent for the same task.

Both deep AEs stated above have the same encoder architecture, **320-250-150-50-25-2**, all units with the sigmoid activation function apart from the **linear** units used in the bottleneck (coding) layer. In your experiment, set `batch_size=64`, `momentum=0.5`, `patience=40` and `epochs=300` in fine-tuning and direct training of the deep AE as specified in the following two assignments, but you need to tune the learning rate `lr` by yourself to achieve an acceptable performance. For visualisation, you are asked to project all test instances onto the 2-D latent space with the encoder obtained by two trained deep AEs, respectively. The format for the required scatter plots has been completed partially in the notebook. Also, you may use the `plot_alphadigit_results(net,X,y_labels)` function for visualisation and showing reconstruction results. This function works in a manner similar to the `plot_results` function described in Part 1.

Assignment 7 [4 marks] Use RBMs as the building blocks in the hybrid learning strategy to construct the deep AE stated above. For the greedy layerwise pre-training, you are provided a function in in the `Network.py` module, which enable to pre-train a RBM-based deep AE with a pre-specified architecture. In your experiment, you can pre-train the RBM-based deep AE by using the function in the manner as follows:

```
pretrain_autoencoder(net, x, x_val, rbm_lr=0.1, rbm_use_gauss_visible=False,
rbm_use_gauss_hidden=False, rbm_mom=0.5, rbm_weight_decay=0.0, rbm_lr_decay=0.0,
rbm_batch_size=100, rbm_epochs=500,rbm_patience=25)
```

where `net` is a Network object (see the Supportive Software section for details) and needs to be set to the deep AE architecture including both encoder and decoder. Apart from training and validation data settings, other hyperparameters required in the pre-training stage for construction and training of RBMs have been set up for this assignment so you can pre-train the deep AE with no change on those hyperparameters related to RBM (see the Supportive Software section for the details of those RBM hyperparameters). In your answer notebook, (a) describe your experimental procedure in detail; (b) report the minimum averaged reconstruction error achieved on the test data along with the learning rate leading to this result; and (c) display the 2-D representations of all test images coloured with their symbol labels (the colour scheme for visualisation must be given explicitly

[7] For details of the hybrid learning strategy, see the "Deep Autoencoder" lecture note.

in the plot).

Assignment 8 [4 marks] Train the deep AE of the architecture stated above with random initialisation. You must use the exactly same held-out validation procedure used in **Assignment 7**. In your answer notebook, (a) describe your investigation in hyperparameter tuning and explicitly list those hyperparameter values that lead to the minimum averaged reconstruction error; (b) with those hyperparameter values leading to the minimum averaged reconstruction errors, plot all the learning curves during fine-tuning and direct training for two deep AEs used in **Assignments 7** and **8** in **one single plot** and describe when you stop the training (Hint: To achieve the losses of a deep AE during training, you can use the objects, `Network.train_err_hist` and `Network.val_err_hist`.); and (c) display the 2-D representations of all test images coloured with their digit labels (the colour scheme for visualisation must explicitly appear in the plot), describe any differences between results achieved by two deep AEs and explain the reason causing the difference.

Requirement: Before starting working on this assessed coursework, you need to

1. download all the files required by this coursework from Blackboard as specified at the beginning of this document;
2. unzip the file then you should be see a Jupyter notebook file named `AE_app.ipynb` and two sub-directories named `Data` and `Code`, respectively (you must keep this directory/sub-directory structure and their names unchanged when you work on this coursework);
3. rename `AE_app.ipynb` in the directory as `yourfullname_AE_app.ipynb`. For instance, if your name is "John Smith", your filename should be `john_smith_AE_app.ipynb`. This file will be your answer notebook to be submitted for marking, so you must include everything required by the coursework in this Jupyter notebook.

Deliverable: Only your answer notebook, `yourfullname_AE_app.ipynb`, which should include all your code, output, answers and your interpretation/justification. In this Jupyter notebook, all assignments have been separated with the clear delimiters. You must put your stuff regarding an assignment in those cells related to this assignment and, if necessary, create new cells within the delimiters of this assignment.

Your answer notebook, `yourfullname_AE_app.ipynb`, must be submitted via the Blackboard.

Marking: Marking is on the basis of (1) correctness of results and quality of comments on your code; (2) rigorous experimentation; (3) how informative and clear your description/answers presented in your answer book; and (4) your knowledge exhibited, interpretation and justification.

Late Submission Policy: The default university late submission policy (for details, see the official document: `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=29825`) is ap-

plied to this coursework.

Extension Policy: The default departmental extension policy is applied to this coursework. That is, no extension is allowed unless you have a mitigating circumstance. If you have any mitigating circumstance and want to make an extension, you should submit the completed mitigating circumstance form to SSO (for details, see the departmental Mitigating Circumstances page: `http://studentnet.cs.manchester.ac.uk/assessment/mitigatingcircumstances.php?view=pgt`). Note: the decision will be made by the departmental mitigating circumstance panel rather than the lecturer.

Deadline: 18:00 GMT, 9th December 2021 (Thursday)