Autoencoders and RNNs

Due at 4:00pm on **Tuesday 24 March 2020**

What you need to get

- `YOU_a4.ipynb`: a Python notebook (hereafter called "the notebook")
- `Network.py`: Module containing `Network` and `Layer` classes
- `mnist_loader.py`: Module for reading in MNIST data
- `mnist.pkl`: MNIST data
- `autoencoder.net`: Saved network file
- `origin_of_species.txt`: Text file

What you need to do

1. **Autoencoder** [13 marks total]

   In this question, you will create an autoencoder and train it on the MNIST digits.

   (a) [3 marks] Create a deep autoencoder neural network with at least 5 layers, and in which the middle layer has only 3 neurons. Train it on 1,000 digits from the MNIST dataset. Your network's input should have 784 neurons, and its output layer should have 784 neurons. Use stochastic gradient descent to train the network. You will have to find hyperparameters that work (eg. number of epochs, batch size, learning rate). You should use the supplied `Network` class.

   Warning: This training is a lot more computationally expensive than previous assignments, so don't save it until the last day and hope to run it on the School servers.

   (b) [2 marks] Show that your hidden layer successfully encodes the digits by encoding and reconstructing at least one sample of each digit class (0 through 9). Display the original and reconstructed digit images in 2 rows, like shown in here.

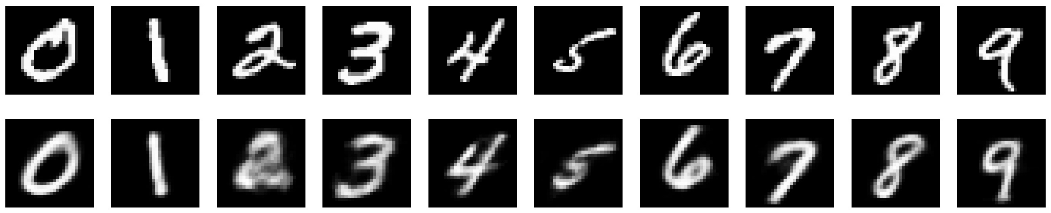


Figure 1: MNIST Reconstructions

   (c) [3 marks] Plot the compressed representation of each of the inputs in a 3-dimensional scatter plot. That is, perform a feedforward pass on each of the input images, record the activity of the 3 hidden neurons for each input, and plot all those 3-vectors in a 3-dimensional scatter plot, each point coloured according to its class.

   (d) [5 marks] Load the neural network supplied in the file `autoencoder.net` using the line

```
net = Network.Network.Load('autoencoder.net')
```

   Run the dataset through the network and display its 3D embedding (as you did for question 1c).

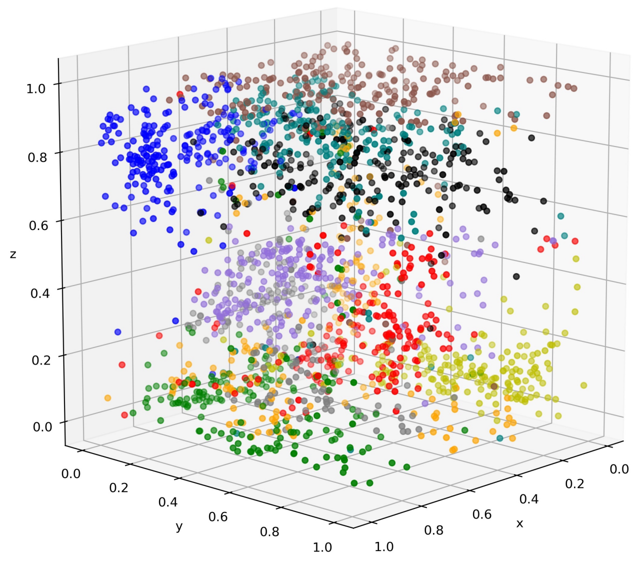


Figure 2: Embedding

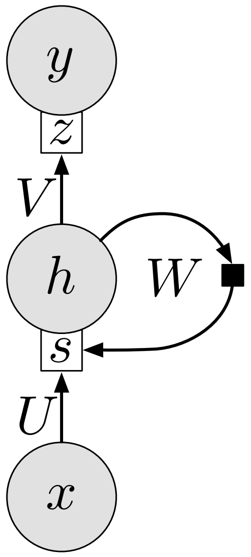
You will find the function `net.FeedForwardFrom` helpful for the following tasks. Using this network, generate the following outputs.

i. Choose a class that has points that are widely spread out, but still reasonably separated from the other classes. Select a location in the 3D embedding space that is within this cloud of points. Call this point A. Select another point within the same class, but far away from A. Call this point B. For each of A and B, reconstruct the corresponding output.

ii. Choose two classes that have well-delineated point clouds that are close together. Select a location from within each point cloud, but so that the selected points are still quite close together. Reconstruct the outputs from each of those two locations, and display the reconstructions side-by-side.

iii. There are places where the point clouds of different classes overlap. Choose a location from where two classes overlap, and display the reconstruction.

2. **Backprop Through Time** [10 marks total]



The figure on the right shows an RNN. Note that

$$\begin{aligned}
s &= Ux + Wh + b \\
h &= \sigma(s) \\
z &= Vh + c \\
y &= \sigma(z)
\end{aligned}$$

Notice that we are using the mathematical convention of assuming vectors are column-vectors by default.

For the following questions, assume you are given a dataset that has many samples of sequences of inputs and output targets. Each sequence consist of inputs x^i, for $i = 1, \ldots, \tau$, that produces a sequence of network outputs y^i, which you wish to match to a corresponding sequence of targets, t^i. The cost function for such a sequence is,

$$E(y^1, \ldots, y^\tau, t^1, \ldots, t^\tau) = \sum_{i=1}^{\tau} C(y^i, t^i)$$

(a) [3 marks] Show that the gradient of the cost with respect to the weights V can be written

$$\frac{\partial E}{\partial V} = \sum_{i=1}^{\tau} \left(\frac{\partial C\left(y^i, t^i\right)}{\partial y^i} \odot \sigma'(z^i) \right) (h^i)^{\mathrm{T}}$$

(b) [2 marks] Suppose you have computed $\frac{\partial E}{\partial h^i}$ for $i = 1, \ldots, \tau$. Show that

$$\frac{\partial E}{\partial U} = \sum_{i=1}^{\tau} \left(\frac{\partial E}{\partial h^i} \odot \sigma'(s^i) \right) (x^i)^{\mathrm{T}}$$

(c) [4 marks] Also, show that

$$\frac{\partial E}{\partial W} = \sum_{i=1}^{\tau-1} \left(\frac{\partial E}{\partial h^{i+1}} \odot \sigma'(s^{i+1}) \right) (h^i)^{\mathrm{T}}$$

(d) [1 mark] Finally, show that

$$\frac{\partial E}{\partial b} = \sum_{i=1}^{\tau} \left(\frac{\partial E}{\partial h^i} \odot \sigma'(s^i) \right)$$

3. **Recurrent Neural Network** [14 marks total]

In this question, you will complete the Python implementation of backprop through time (BPTT) for a simple recurrent neural network (RNN). The notebook contains a definition for the class `RNN`. The class has a number of methods, including `BPTT`. However, `BPTT` is incomplete.

For training and testing, the notebook also reads in a corpus of text (a simplified version of *On the Origin of Species* by Charles Darwin), along with the character set, and creates about 5000 training samples. The notebook also creates a few utility functions that help convert between the various formats for the data.

(a) [8 marks] Implement the function `BPTT` so that it computes the gradients of the loss with respect to the connection weight matrices and the biases. Your code should work for different values of `seq_length` (this is the same as τ in the lecture notes).

(b) [2 marks] Create an instance of the RNN class. The hidden layer should have 400 ReLU neurons. The input to the network is a one-hot vector with 27 elements, one for each character in our character set. The output layer also has 27 neurons, with a softmax activation function.

(c) [2 marks] Train the RNN for about 15 epochs. Use categorical cross entropy as a loss function (see A2 Q2 for help with this). You can use a learning rate of 0.001, but might want to break the training into 5-epoch segments, reducing the learning rate for each segment. Whatever works.

(d) [2 marks] What fraction of the time does your RNN correctly guess the first letter that follows the input? Write a small bit of Python code that counts how many times the next character is correct, and express your answer as a percentage in a print statement.

What to submit

Your assignment submission should be a single jupyter notebook file, named (`<WatIAM>_a4.ipynb`), where `<WatIAM>` is your UW WatIAM login ID (not your student number). The notebook must include solutions to **all** the questions. Submit this file to Desire2Learn. You do not need to submit any of the modules supplied for the assignment.