Autoencoders and RNNs

Due at 4:00pm on Friday 22 March 2019

What you need to get

- `YOU_a4.ipynb`: a Python notebook (hereafter called "the notebook")
- `Network.py`: Module containing Network class
- `mnist_loader.py`: Module for reading in MNIST data
- `mnist.pkl`: MNIST data
- `origin_of_species.txt`: a 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) [4 marks] Consider the *cosine proximity* loss function

   $$C(\vec{y}, \vec{t}) = \frac{-\left(\vec{y} \cdot \vec{t}\right)}{\|\vec{y}\| \, \|\vec{t}\|} \, .$$

   It is the negative of the cosine of the angle between $\vec{y}$ and $\vec{t}$. Based on that loss function, we can define cosine proximity cost as the expected loss,

   $$E(Y, T) = \left\langle C\left(\vec{y}, \vec{t}\right) \right\rangle_{\vec{y} \in Y, \vec{t} \in T} \, .$$

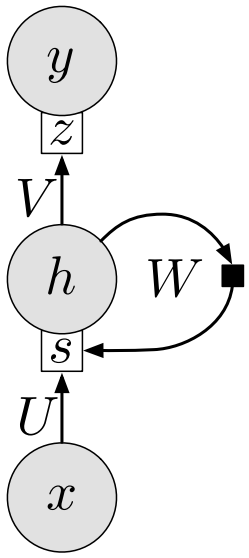
   Find a formula for the gradient of the cost function with respect to the output, $\vec{y}$. That is, find a formula for $\frac{\partial E}{\partial \vec{y}}$. Simplify the formula as much as you can.

   (b) [3 marks] Complete the function `CosineProximity_p`. It computes $\frac{\partial E}{\partial \vec{y}}$ for the entire batch. See the function's documentation for more details.

   (c) [4 marks] Create a 3-layer autoencoder neural network and train it on 10,000 digits from the MNIST dataset. Your network's input should have 784 neurons, and its output layer should have 784 neurons that use the identity activation function. The hidden layer should have only 50 logistic neurons. Use stochastic gradient descent to minimize the cosine proximity loss function for at least 20 epochs, and a learning rate of 1. Batch size should be between 30 and 130. You should use the supplied `Network` class.

   (d) [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).

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.