|  |
| --- |
| **Homework 2****COSC 6342: Machine Learning****University of Houston****Department of Computer Science****Sent on: Sept. 28, 2017; Due: Oct. 9, 2017** |

|  |
| --- |
| Name : |

Teams of up to two members are allowed for this homework.

**Overview** the idea behind this homework is to provide an experience not too different from a small contract job taken up by a ML expert. We recommend to use certain tools because that is what the industry has mostly adopted, but ultimately students are given a lot of freedom in how they approach the assignment.

**Goals** to gain practical experience in Deep Learning by implementing a state-of-the-art algorithm, to become aware of tools and libraries that may be helpful,

**Recommended prior knowledge** Understanding of Neural Networks. Passing familiarity with the concept of Deep Learning.

**Tools** It is recommended to use the following software; 1) They enjoy the most support from the community so information is easy to find 2) Each is a de-facto industry standard that any collaborator or company would most likely expect you to be familiar with.

Language: Python (ideally 3.x, 2/x has a few issue with encodings)

OS: Any, Linux preferred.

Deep Learning API: Tensorflow, at least 1.0.

Other libraries you may find helpful: scikit-learn, pandas, scikit-imbalanced, matplotlib.

Hardware: Ideally, NVIDIA GPU. Don’t worry if you don’t have one, it will just take longer to train the model.

**Neural Networks**

**Part 0: Setup**

1. Install Tensorflow <https://www.tensorflow.org/>

**It is strongly recommended that you use a GPU, else training the model will take a long time! For Deep Learning most NVIDIA GPUs will work. See** <https://www.tensorflow.org/tutorials/using_gpu>

**Score:** 0 points, +5 bonus if you setup a GPU and use it for the rest of the homework. To get the bonus, take a screenshot when tensorflow is loading (as shown in previous link, or at any moment you start it) - it should be displaying the type and the number of GPUs used by the library.

**Part 1a: Perceptron**

Go to:

[**https://www.tensorflow.org/get\_started/mnist/pros**](https://www.tensorflow.org/get_started/mnist/pros)

Build a softmax regression model and train it on MNIST, as described. Visualize the training process using Tensorboard:

<https://www.tensorflow.org/get_started/summaries_and_tensorboard>

Report accuracy and tensorboard output (a screenshot will do for the tensorboard)**(10 points)**

**Part 1b: Convolutional Neural Network**

Following the same tutorial, build a Convolutional Neural Network. Train and evaluate it on MNIST, reporting results and tensorboard output. (**20 points)**

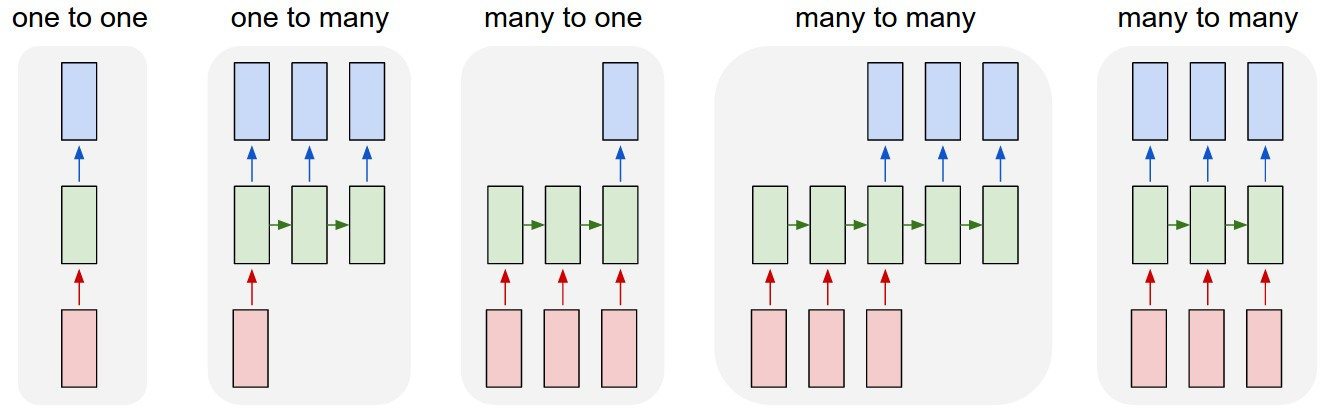
**Part 1c: Questions**

Which model performed better? Why do you think that happened? **(5 points)**

What would happen if we added many more layers to the network you implemented in 1b? How would it perform on MNIST and why? **(5 points)**

**Recurrent Neural Networks: Brief Introduction**

*What makes Recurrent Networks so special*? A glaring limitation of Vanilla Neural Networks (and also Convolutional Networks) is that their API is too constrained: they accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes). Furthermore, these models perform this mapping using a fixed amount of computational steps (e.g. the number of layers in the model). Recurrent nets allow us to operate over *sequences* of vectors:

**Figure 1**: Network layouts **(1)** Fixed size (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g.sentiment analysis.) **(4)** Sequence input and sequence output (e.g. Machine Translation) **(5)** Synced sequence input and output (e.g. video classification).

**RNN Implementation:** At the core, RNNs have a deceptively simple API: They accept an input vector x and give you an output vector y. However, crucially this output vector’s contents are influenced not only by the input you just fed in, but also on the entire history of inputs you’ve fed in in the past. Written as a class, the RNN’s API consists of a single step function:

rnn **=** RNN()  
y **=** rnn**.**step(x) *# x is an input vector, y is the RNN's output vector*

The RNN class has some internal state that it gets to update every time step is called. In the simplest case this state consists of a single *hidden* vector h. Here is an implementation of the step function in a Vanilla RNN:

**class** **RNN**:  
 *# ...*  
 **def** **step**(self, x):  
 *# update the hidden state*  
 self**.**h **=** np**.**tanh(np**.**dot(self**.**W\_hh, self**.**h) **+** np**.**dot(self**.**W\_xh, x))  
 *# compute the output vector*  
 y **=** np**.**dot(self**.**W\_hy, self**.**h)  
 **return** y

The above specifies the forward pass of a vanilla RNN.

**Part 2: Writing Poetry**

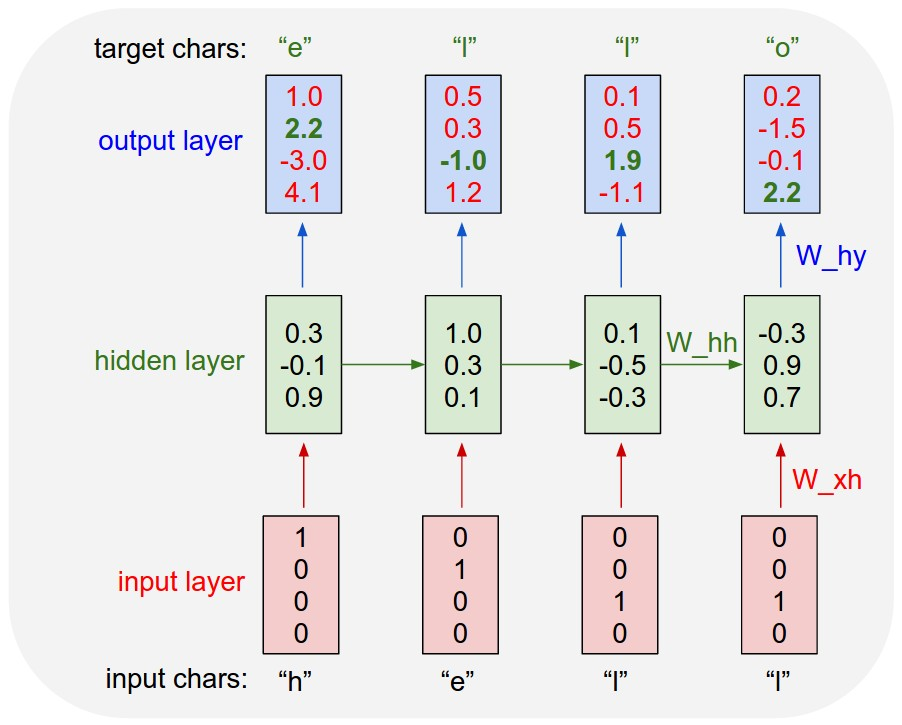
In this part, we will give an RNN a huge chunk of text and ask it to model the probability distribution, essentially predicting the next character in the sequence. We use one-hot encoding for each character, where the vector length of the encoding is equal to the number of possible characters and only one item in the vector is set to 1, with the rest being 0:

Figure 2: Character-level encoding.

There are many articles, tutorials and blog posts describing RNNs and this problem in particular. The research component of this homework consists of using whatever resources you have at your disposal (without plagiarism) to do the following:

1. Find a collection of works by your favorite poet that is in text format. **(5 pts)**
2. Use tensorflow, scikit-learn, or your own code to convert the texts into one-hot-encoding format: **(10 pts)**
3. **Using Tensorflow (there are many examples on the official website),** implement a multi-layer RNN that can take these encodings as inputs. Here is an example of a very minimal RNN that you can use for reference: <https://gist.github.com/karpathy/d4dee566867f8291f086> **Simply copying it or only slightly modifying will earn you no credit. (20 pts)**
4. Train the model on the poetry you have obtained. **(10 pts)**
5. Feed a character (can start with random character) into the RNN and get a distribution over what characters are likely to come next. Sample from this distribution, and feed it right back in to get the next letter. Repeat this process and you’re sampling text! **(15 pts)**

What to submit:

A pdf document in a reasonably nice format. For each question in **Part 1** where training/testing were involved, report accuracy, also attach screenshots and answer questions wherever asked. For **Part 2,** submit the output text produced by the RNN.

***All code for both parts and part 2’s input text should be attached in a separate directory.***

Zip your pdf and the directory with the code and the text and submit.