COGS 260, Spring 2018: Assignment 4

May 17, 2018

Due: May 27, 2018 11:59 PM PDT
Late policy: Every 10% of the total received points will be deducted for every extra day past due.

Instructions

1. Reference materials and some useful codes can be found at the end of the document.
2. Submit the ipython notebooks you used for this project in pdf format.
3. Write the rest of the report in pdf including: a) abstract, b) method, c) experiment, d) discussion, and e) references. You can follow leading conferences like CVPR (`http://cvpr2017.thecvf.com/submission/main_conference/author_guidelines`), NIPS (`https://papers.nips.cc/`), or ICML (`http://icml.cc/2016/?page_id=151`). All of those formats are readily available through `www.sharelatex.com`. For the homework assignments, the report can be brief and you can try to summarize your experiments, results, and main findings in a concise way.
4. Submit any ipython notebooks in pdf separately and zip and upload related code and ipython notebooks as part of your submission on TritonEd in order for the TA to reproduce your result.
5. You are supposed to provide concise quantitative details and analysis of experiments whenever asked to report the details of the experiment. Please DO NOT write long paragraphs. You are encouraged to use plots and images to explain your results. For the last question, you only need to provide writings in a ReadMe.txt file that specifies steps that you used to run your experiments with a reference to the code base and a summary of the database you have used.

In this assignment, in the first problem, you will first write a single-layer feed forward network, then you are going to play with Recurrent Neural Networks and some neural networks for object detection. In the second problem you are going to learn how character-RNNs can be used to reproduce any text documents by predicting one character at a time. In the last problem you are going to get familiar with some trade offs of neural networks used for object detection.

1 Feed-forward Neural Network [40 pts]

Fill the provided ipython notebook with your own implementations to train a simple feed-forward neural networks with one hidden layer. You will add a layer between the input and output with the sigmoid activation function, and this hidden layer only contains 2 hidden units. The network can be formulated as a function:

$$f(x) = \sigma(w_2^T(\sigma(w_1^T x + b_1)) + b_2) \tag{1}$$

where $w_1 \in R^{4\times 2}$ and $w_2 \in R^{2\times 1}$ are the weights, b_1 and b_2 are the bias. The activation function is the logistic sigmoid function, which is defined as follows:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2}$$

The logistic sigmoid function is applied on the each entry in the vector or matrix. For example, we define $q = \sigma(w_1^T x + b_1)$ as the output of the first layer. The output of $(w_1^T x + b_1)$ is a 2-dimension

vector, and the sigmoid function $\sigma(\cdot)$ is applied at each entry of the 2-dimensional vector, so q is still a 2-dimension vector.

In order to train a binary classsifier with this network, we define a cross-entropy loss function L to measure the distance between the target and the prediction from this network, the L is defined as:

$$L(w_1, w_2, b_1, b_2) = -\sum_i (y^{(i)} ln f(x_i) + (1 - y^{(i)}) ln(1 - f(x_i))) \tag{3}$$

We aim to minimize the loss function L over the training set. Since there is no closed-form solution for the weights and the bias in the network, we consider gradient descent algorithm to train the network.

Your Task

1. Apply the **Chain Rule** to derive $\frac{\delta L}{\delta w_2}$ and $\frac{\delta L}{\delta w_1}$, respectively.
2. Download **q1_feedforward.zip** from the course website. This folder contains a dataset **q1_data.txt**, which is modified from the Iris dataset. Fill blanks in **q1_feedforward.ipynb** to train the network with **gradient descent** algorithm on the training set, and plot the training loss vs. the number of iterations.
3. Report your training error and test error.

2 Char RNN [30 pts]

Train a vanilla RNN on *tinyshakespeare.txt* data. The network should predict the next character when given a sequence of characters as input.

For example: Say if the input sequence at time t is {'T','h','i','s',' ','i','s',' ','a',' ','b','o'} then the most likely output chacter at time t could be 'y'.

In a similar way, start with some valid sequence of characters at t_0 and predict the next character. Use this predicted character along with the input sequence at t_0 as input sequence at t_1 to predict the next character. Repeat this step to produce at least 500-800 characters.

You may refer to [1] for more details about this problem. You may have to scroll down in the website to locate the part where sample output for Shakespeare is provided. A sample code to do this in python can be found in [2]. However, writing your own implementation won't be very difficult either.

Report the input and output representation of your network, network architecture and hyperparameters, i.e. all the information what one would need to reproduce the same network. Put at least 3 outputs generated after starting with different initial input sequence. Also, plot the training loss wrt. number of iterations (or epochs).

You may use a fraction of the dataset to overcome the computational challenges, but be sure to mention it in your report. You may optionally try 2-layered RNNs or may be LSTMs if you have the necessary computational power.

NOTE 1: The initial character sequence for reporting the output should end with the <start> token. (Can you guess why?)
NOTE 2: The <start> and the <end> tokens should be considered as single character rather than a sequence of characters (words). (Can you guess why?)

3 Object Detection [30 pts]

Select at least one model that are either based on SSD (Single Shot Detector) [13], RCNN (Region-Convolutional Neural Network) [14], R-FCN (Region-based Fully Convolutional Networks) [15] or YOLO (You Only Look Once)[11] and run the following two experiments.

3.1 Scale Invariance [15 pts]

Scale and create 4×4, 8×8 and 16×16 image collages of an image of your choice as shown here in Figure 1. Run object detection on the 3 image collages with the set network(s) of your choice and show and discuss the results. You can try the same set of experiments on a couple of images to get a better comparison.

Figure 1: An example 4×4 image collage of a dog image

3.2 Rotational Invariance [15 pts]

Rotate an image of your choice with 30 degree intervals and repeat object detection test on the rotated image with the same set of network(s) of your choice from previous question. Show and discuss the results. You can create a gif per network to see a better comparison.

NOTE:

Models from COCO-trained models in the model zoo [8] can be directly used for models based on SSD, RCNN, or R-FCN in tensorflow. The API code base is here [6]. An ipython notebook tutorial is readily available here[10]. You can also find the official YOLOv3 tutorial here [12], which uses C. You are welcome to use any other reliable code base for your experiments as well.

References

[1] Karpathy, Andrej. The Unreasonable Effectiveness of Recurrent Neural Networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

[2] Karpathy, Andrej. Char-RNN code. https://gist.github.com/karpathy/d4dee566867f8291f086

[3] Pathak, Apurva, et. al. Context-Based Music Composition using RNN Generative Model. http://www.apurvapathak.me/files/CSE253_Project.pdf

[4] The abc music standard 2.1, http://abcnotation.com/wiki/abc:standard:v2.1.

[5] The Nottingham Collection. http://abc.sourceforge.net/NMD/.

[6] Tensorflow Object Detection. https://github.com/tensorflow/models/tree/master/research/object_detection

[7] mAP and more https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/evaluation_protocols.md

[8] model zoo https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

[9] Speed/accuracy trade-offs for modern convolutional object detectors. https://arxiv.org/pdf/1611.10012.pdf

[10] Object Detection Demo https://github.com/tensorflow/models/blob/master/research/object_detection/object_detection_tutorial.ipynb

[11] YOLOv3: An Incremental Improvement https://pjreddie.com/media/files/papers/YOLOv3.pdf

[12] YOLOv3 tutorial https://pjreddie.com/darknet/yolo/

[13] Single Shot MultiBox Detector https://github.com/weiliu89/caffe/tree/ssd

[14] Mask RCNN https://github.com/facebookresearch/Detectron

[15] Object Detection via Region-based Fully Convolutional Networks https://github.com/YuwenXiong/py-R-FCN

[16] Inference and evaluation on the Open Images dataset https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/oid_inference_and_evaluation.md

[17] using your own dataset https://github.com/tensorflow/models/blob/505f554c6417931c96b59516f14d1ad65df6dbc5/research/object_detection/g3doc/using_your_own_dataset.md