Homework Assignment 4

COGS 181: Neural Networks and Deep Learning

Due: November 12, 2017, 11:59pm
Instructions:

1. Please answer the questions below, attach your code, and insert figures to create a pdf file; submit your file to TED (ted.ucsd.edu) by 11:59pm, 11/12/2017. You may search information online but you will need to write code/find solutions to answer the questions yourself. Write your program in Python.

Late Policy: 5% of the total points will be deducted on the first day past due. Every 10% of the total points will be deducted for every extra day past due.

Grade: ____ out of 100 points

1 (30pts) Hopfield Network

In this homework exercise, we will write a program to "train" and simulate a Hopfield network that memorizes 1-D vector patterns.

Training (Constructing Weights)

Instead of "training" a Hopfield network in the sense that you're used to, the weight matrix in Hopfiled networks can be calculated directly using a formula provided in the original paper. Suppose we have N nodes and M states and (here $M = 2$) is the number of states to calculate W and can be retrieved by using W. For each possible state V^s, $s = 1 \cdots M$, the value of elements in V^s is either -1 or +1. We can compute the weight matrix using:

$$W_{ij} = \frac{1}{N} \sum_{s=1}^{M} V_i^s V_j^s, \qquad \text{if } i \neq j \tag{1}$$

And the diagonal values of the weight matrix are all zero, i.e. $w_{ii} = 0$.

Probing Pattern

Given an unknown pattern V_{new}, you will initialize the Hopfield network output nodes at time t $U_i(t)$ as follows:

$$U_i(0) = V_i^{new}, \qquad i = 1 \cdots n \tag{2}$$

Dynamic Evolution

The inference procedure is really similar to what you have done in perceptron. Iterate until convergence:

$$U_i(t+1) = f_h\Big(\sum_{j=1, j\neq i}^{N} W_{ji}U_j(t)\Big) \tag{3}$$

$f_h(.)$ is a hard-limiting non-linearity function, where $f_h(x) = +1$ when $x \geq 0$ and $f_h(x) = -1$ when $x < 0$.

In this step, you need to determine an order for updating the nodes. Typically we use a random visiting sequence (e.g. for a 4-node network, an order can be 2,3,4,1,2,3,4,1,2,3,4,1...).

The stopping criteria is when the outputs from the node remains unchanged. This means that if you go through N consecutive neurons without changing any of their values, then you're at a fixed-point so you can stop.

Your Tasks

Write code to build a 5-node Hopfield Network which can converge to the following two states respectively following below procedure:

- (-1, +1, +1, -1, +1)
- (+1, -1, +1, -1, +1)

Please modularize your code to explicitly perform the three steps (Training, Probing Pattern, Dynamic Evolution).
For the Probing Pattern step, please use the initial input state vector $V^{new} =$

- (+1, +1, +1, +1, +1)

For the Dynamic Evolution step, please try two different node visiting orders respectively starting from V^{new}:

- (3, 1, 5, 2, 4, 3, 1, 5, 2, 4...)
- (2, 4, 3, 5, 1, 2, 4, 3, 5, 1...)

In your write up, please

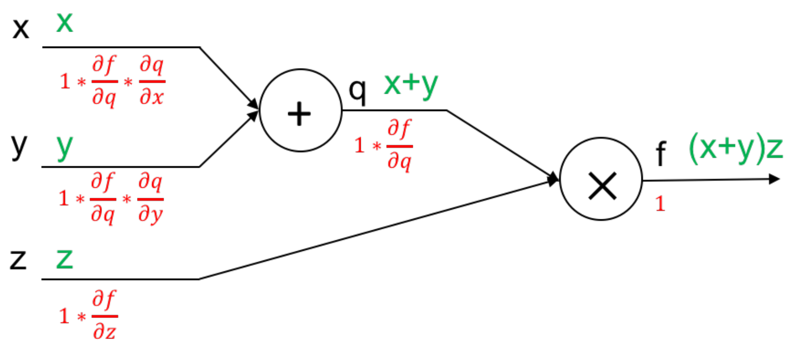
- attach the code,
- log the node updating history at each iteration (print out each $U_i(t+1)$ you get during evolution).
- show the matching between the retrieved pattern (the final fixed-point state your network converges to) and the evolving sequence.

2 (20pts) Visualizing Backpropagation with Circuit Diagrams

The computation of the backpropagation can be nicely visualized with a circuit diagram. Let's consider a complicated expression which involve multiple composed functions, and the function is defined as

$$f(x, y, z) = (x + y)z. \tag{4}$$

We visualize the function with a circuit diagram, which helps us intuitively understand the backpropagation. The visualization is shown in Fig.1.



x
x
$1 * \frac{\partial f}{\partial q} * \frac{\partial q}{\partial x}$
y
y
$1 * \frac{\partial f}{\partial q} * \frac{\partial q}{\partial y}$
z
z
$1 * \frac{\partial f}{\partial z}$
+
q
x+y
$1 * \frac{\partial f}{\partial q}$
×
f
(x+y)z
1

Figure 1: Circuit Diagram of the function $f(x, y, z) = (x + y)z$.

Particularly, this expression can be broken down into two expressions: $q = x + y$ and $f = qz$. According to **Chain Rule**, the derivative/gradient of f with respect to its inputs x, y, and z can be calculated easily with the intermediate variable q. Although we don't really care about the gradient of f with respect to q, but the gradient $\frac{\partial f}{\partial q}$, which simplifies the calculation of the gradient.

Let's take a look into the circuit diagram. The letters in black represent the variables, which are input variables x, y, z, intermediate variable q and output f. The expressions in green represent the current values of the variables respectively, and the expressions in red represent the gradient of f with respect to the particular variables.

The circuit diagram visualizes how the gradient of f with respect to each variable is calculated. Let's follow the circuit diagram to perform a forward calculation and a backward calculation by hand.

In Fig. 2, suppose we have three inputs, which are $x = -2$, $y = 5$ and $z = -4$, and the circuit diagram represents the function $f(x, y, z) = (x + y)z$. We follow the steps below to calculate the value and the gradient with respect to each variable in the diagram.

1. $q = x + y = 3$
2. $f = qz = -12$
3. $\frac{\partial f}{\partial q} = \frac{\partial (qz)}{\partial q} = z = -4$
4. $\frac{\partial f}{\partial z} = \frac{\partial (qz)}{\partial z} = q = 3$

5. $\dfrac{\partial f}{\partial x} = \dfrac{\partial f}{\partial q}\dfrac{\partial q}{\partial x} = -4 \times 1 = -4$

6. $\dfrac{\partial f}{\partial y} = \dfrac{\partial f}{\partial q}\dfrac{\partial q}{\partial y} = -4 \times 1 = -4$

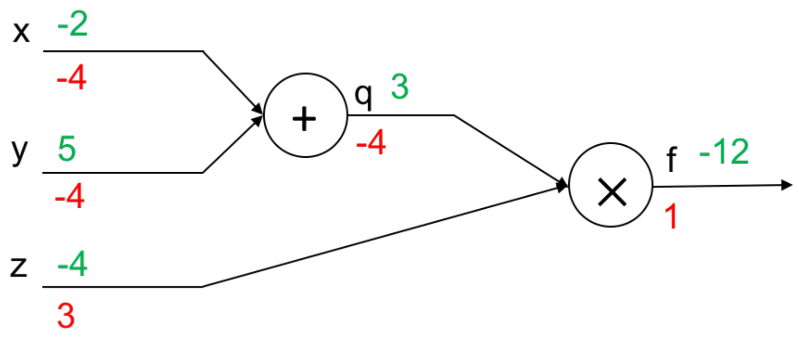


Figure 2: The real-valued circuit diagram of the function $f(x, y, z) = (x + y)z$. (Details can be found from *Fei-Fei Li*, *Andrej Karpathy* and *Justin Johnson*'s course, Stanford CS231n. Link: http://cs231n.github.io/optimization-2/)

Your Tasks

1. Draw a circuit diagram for the function defined as below:

$$f(x, y, z, w) = \frac{1}{\max(x, y) \times (z + w)}, \tag{5}$$

   where $\max(x, y)$ takes the maximum value of x and y as output.

2. Suppose the value for each input is $x = 2.5$, $y = -1$, $z = 1$ and $w = 3$. Perform the forward computation and the backward computation of the function f, and complete the circuit diagram with your calculated values and gradients.

3 (20pts) Feed-forward Neural Network

We previously used logistic regression with cross-entropy loss as the objective function in homework assignment 2, to learn a binary classifier, which is a single-layer network to classify the modified IRIS dataset. Here, we add another 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:

$$f(\mathbf{x}) = \sigma\left(\mathbf{w}_2^T\left(\sigma(\mathbf{w}_1^T\mathbf{x} + b_1)\right) + b_2\right), \tag{6}$$

where $\mathbf{w}_1 \in \mathbb{R}^{4\times 2}$ and $\mathbf{w}_2 \in \mathbb{R}^{2\times 1}$ are the weights, b_1 and b_2 are the bias. The activation function is the logistic sigmoid function, which is defined as follows:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{7}$$

The logistic sigmoid function is applied on the each entry in the vector or matrix. For example, we define $\mathbf{q} = \sigma(\mathbf{w}_1^T\mathbf{x} + b_1)$ as the output of the first layer. The output of $\left(\mathbf{w}_1^T\mathbf{x} + b_1\right)$ is a 2-dimension vector, and the sigmoid function $\sigma(\cdot)$ is applied at each entry of the 2-dimensional vector, so $\mathbf{q}$ is still a 2-dimension vector.

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

$$\mathcal{L}(\mathbf{w}_1, \mathbf{w}_2, b_1, b_2) = -\sum_i \left(y^{(i)} \ln f(\mathbf{x}_i) + (1-y^{(i)})\ln(1-f(\mathbf{x}_i))\right) \tag{8}$$

We aim to minimize the loss function $\mathcal{L}$ 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.
Hint: You can compute $\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial f(x)}\frac{\partial f(x)}{\partial q}\frac{\partial q}{\partial W_1}$ where $q = \sigma(W_1^T X + b_1) = [q_1, q_2]^T$. For $\frac{\partial f(x)}{\partial q}\frac{\partial q}{\partial W_1}$, you can compute it using $\frac{\partial f(x)}{\partial q}\frac{\partial q}{\partial W_1} = \frac{\partial f(x)}{\partial q_1}[\frac{\partial q_1}{\partial W_1^1}^T, \mathbf{0}] + \frac{\partial f(x)}{\partial q_2}[\mathbf{0}, \frac{\partial q_2}{\partial W_1^2}^T]$ where $W_1 = [W_1^1, W_1^2]$.

Your Tasks

1. Apply the **Chain Rule** to derive $\dfrac{\partial \mathcal{L}}{\partial \mathbf{w}_2}$ and $\dfrac{\partial \mathcal{L}}{\partial \mathbf{w}_1}$, respectively.

2. Download the data file **Q4_data.txt** from the course website. This dataset is still modified from **Iris** dataset.

   Training/Test set
   The dataset currently contains 100 instances. You need to combine first 15 instances of each class into test set, and leave the rest of them into the training set.

3. Train the network with **gradient descent** algorithm on the training set, and plot the training loss vs. the number of iterations.

4. Report your training error and test error.

4 (30pts) Multilayer Perceptron

In Q3, we have implemented the feed forward network with 2 hidden units. In this question, we will expand the network and train a multi-class MLP classifier. We will use MNIST dataset for this question. You can fetch the dataset with the following code

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata("MNIST original")
```

There are 70000 images in total and each image is of size 28×28 and has a label from 0 to 9. The first 60000 images are the training images and the rest images are testing images. To train the classifier, you can use the package

```
from sklearn.neural_network import MLPClassifier
```

Preparation

1. Shuffle the training images at the very beginning with the random seed set to 1. Please keep the shuffled training images the same over the all the experiments.
2. For each of experiment, please plot the curve for loss v.s. iterations.
3. The default batch size for the the MLP-classifier is 200, but you may raise the batch size up to 1000 depending on your computer memory.
4. Note: one epoch = (# of training images / batch size) number of iterations

Your Tasks

1. Train the classifier on the training images for one epoch with one layer of 60 hidden units with relu activation with “Adam” solver and ”SGD” solver respectively. Plot the loss curves. What can you tell from your curves?
2. Train the classifier on the training images for one epoch with “Adam” solver with relu activation using one hidden layer with 20,50,100 hidden units respectively. Plot the loss curves. What can you tell from your curves?
3. Train the classifier on the training images for one epoch with “Adam” solver with relu activation with the following settings: 1) one hidden layer with 60 hidden units, 2) 2 hidden layers with 30 hidden units in each layer and 3) 3 hidden layers with 20 hidden units in each layer respectively. Plot the loss curves. What can you tell from your curves?
4. Report the testing accuracy results on all the above classifiers and the setting with the highest performance.