Seung Jun Baek

1 Outline

In this assignment, you are asked to design a neural network implementing backpropagation and gradient descent.

2 Specification

In this assignment, you are asked to write a Python code for implementing a 2-layer neural network (a neural network with 1 hidden layer). The implementation details are provided in the following sections.

3 Overview

Our goal is to implement the *XOR operation*. XOR, exclusive OR, is a logical operation which takes two binary numbers, denoted by $\oplus$. For binary numbers x and y, the operation $x \oplus y$ outputs 1 if $x \neq y$, and outputs 0 if $x = y$. That is

$$0 \oplus 0 = 0$$
$$1 \oplus 1 = 0$$
$$1 \oplus 0 = 1$$
$$0 \oplus 1 = 1$$

Alternatively, the XOR operation can be viewed as a function $XOR(x_1, x_2)$ such that, it takes two binary values x_1 and x_2, and outputs $x_1 \oplus x_2$. This can be also viewed as a *classification* problem in two-dimensional (x_1, x_2) plane. XOR function can be seen as assigning values 0 and 1 as the class labels, that is class 0 and 1. See Fig. 1. However, we can see that it is difficult to use linear classifiers for classification, i.e., to draw a linear decision boundary to classify label 0 and 1, as we have learned in the lecture. Our goal is to build a neural network, instead of linear classifiers, to classify XOR labels.

A two-layer NN to be implemented is depicted in Fig. 2. There is one hidden layer with 4 neurons, and the output layer produces the softmax output, which is the probability score of the given input belongs to either class 0 or 1. The loss function is defined to be the cross entropy loss. The details of the layers are given as follows.

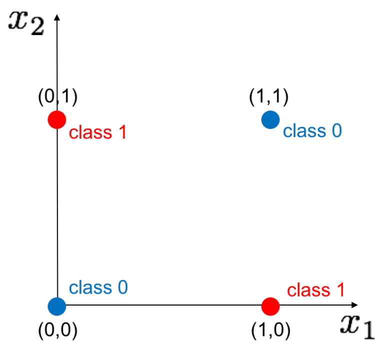

Figure 1: XOR problem.

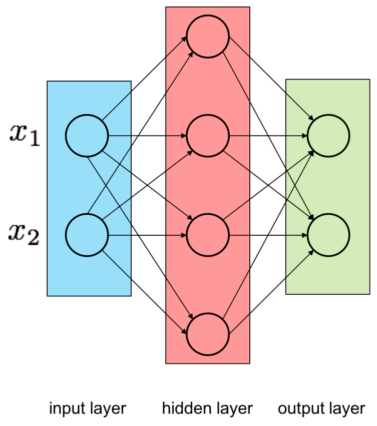

Figure 2: Plot of sample training dataset points with 4 classes.

3.1 input layer.

The input layer takes a two-dimensional point (x_1, x_2) in a plane as a single data sample.

3.2 hidden layer

The hidden layer consists of two computational stages. The first stage is the linear layer taking two-dimensional input and outputs 4-by-1 vector as follows.

$$s = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = W_1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b_1$$

The parameters are W_1 and b_1 which are 4-by-2 weight matrix and 4-by-1 bias vector, respectively. The output of the hidden layer is given by 4-by-1 vector y with sigmoid activation on s. In other words,

$$y = \sigma(s), \text{ or}$$
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \sigma(s_1) \\ \sigma(s_2) \\ \sigma(s_3) \\ \sigma(s_4) \end{bmatrix}$$

where

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

is the sigmoid function. In summary, the input-output relation of the hidden layer is

$$y = \sigma(W_1 x + b_1)$$

3.3 output layer

The output layer takes an input vector of size 4, and outputs 2-by-1 softmax output. Let us denote the output from hidden layer by y. The output layer consists of three computation stages: linear, softmax and cross-entropy.

The first stage is the linear layer, which takes input y which is 4-by-1, and outputs s which is 2-by-1 vector.

$$s = W_2 y + b_2$$

The parameters to be learned are: W_2 and b_2 are 4-by-2 weight matrix and 2-by-1 bias vector respectively.

The second stage is softmax layer. It takes s from the first linear layer then takes the softmax. The output is a 2-by-1 vector $(p, 1-p)$ as follows

$$\begin{bmatrix} p \\ 1-p \end{bmatrix} = \text{softmax}(s)$$

Note that the softmax function is defined in general

$$\text{softmax}(s) = \begin{bmatrix} \frac{\exp(s_1)}{\sum_{i=1}^{m} \exp(s_i)} \\ \frac{\exp(s_2)}{\sum_{i=1}^{m} \exp(s_i)} \\ \vdots \\ \frac{\exp(s_m)}{\sum_{i=1}^{m} \exp(s_i)} \end{bmatrix}$$

In our case, the softmax is done over 2-dimension. So the output is given by

$$\begin{bmatrix} p \\ 1-p \end{bmatrix} = \begin{bmatrix} \frac{\exp(s_1)}{\exp(s_1)+\exp(s_2)} \\ \frac{\exp(s_2)}{\exp(s_1)+\exp(s_2)} \end{bmatrix}$$

Here, p represents a probability score from softmax. There are two classes at the output, class 0 and class 1. Class 0 means the output takes the value of 0. Class 1 means that the output takes the value of 1. Therefore, p can be viewed as the probability that the output belongs to Class 0 (that is, the value of 0).

The third stage is the cross-entropy loss function. The final output is the loss L given by

$$L = \frac{1}{n}\sum_{i=1}^{n} L_i = -\frac{1}{n}\sum_{i=1}^{n}(b_i \log p_i + (1-b_i)\log(1-p_i)) \tag{1}$$

where n is the dataset size, p_i is the softmax output from i-th input data sample, and b_i is the ground-truth label of i-th data sample. So

$$b_i = \begin{cases} 1, & \text{if the label is class 0, that is the value of 0} \\ 0, & \text{if the label is class 1, that is the value of 1} \end{cases}$$

The computational stages for layers are summarized in Fig. 3.

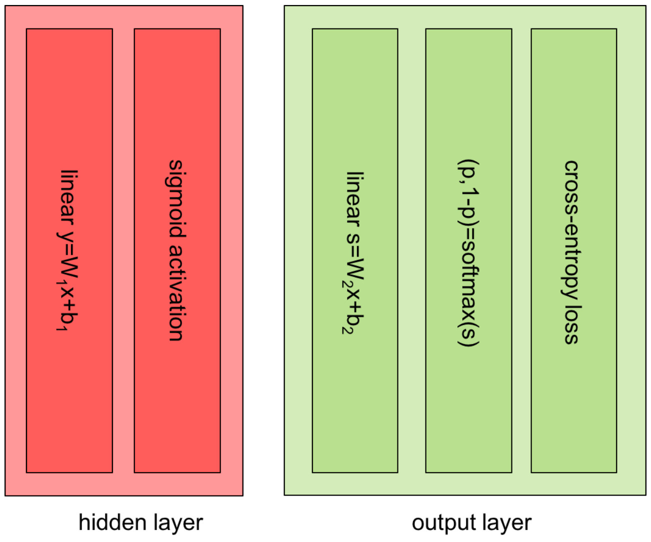

Figure 3: Details of NN layers

3.4 training data

The training data is generated as points in the neighborhood of $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ in the 2-dimensional plane. The data is added with Gaussian random points generated with mean $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$. An example plot is shown in Fig. 4. In the figure, blue points have label of class 0, and red points have labels of class 1. The training dataset in the code has name `x_training_d` and `y_training_d`.

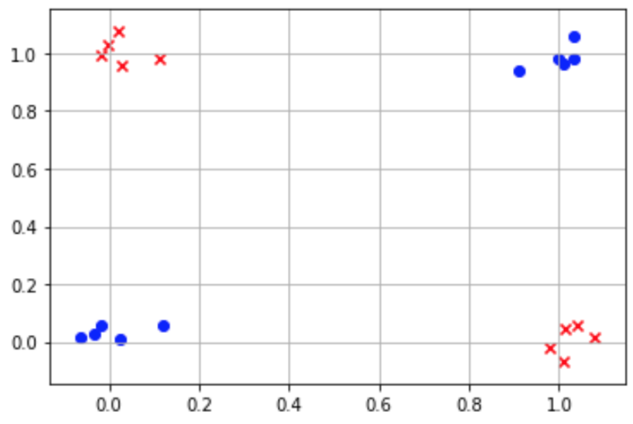

Figure 4: Plot of sample training dataset points with 4 classes.

The size of dataset is 20 points, 5 points per each of $(0,0),(0,1),(1,0)$ and $(1,1)$. The shape of training data `x_train_d` is $(20,2)$, and thus `x_train_d[i]` is the i-th sample of the training data. The labels are stored at `y_train_d`, which has shape $(20,1)$. Check the code for details.

3.5 test data

Because our neural net implements XOR operation, we have 4 points of test data: $(0,0),(0,1),(1,0)$ and $(1,1)$. The accuracy will be measured over `num_test` times (defined in `hw3.py`).

4 Implementation notes

4.1 classes

There are classes defined for each layer:

1. `nn_linear_layer`
2. `nn_activation_layer`
3. `nn_softmax_layer`
4. `nn_cross_entropy_layer`

All of them have two common methods: `forward` and `backprop`.

1. `nn_linear_layer`
   - `forward`: A forward pass of the input on each layer to evaluate the output.

    - input parameter: `x`: input to this layer. Shape:(B, I) where B is batch size, I is input feature dimension
    - Output: $Wx + b$. Shape: (B, O) O is output dimension
  - `backprop`: backpropagation. This method takes 2 parameters:
    - Input parameters:
      * `x`. This is the input to this layer. Shape:(B, I)
      * `dLdy`. This is the upstream gradient coming from the next layer. Shape:(B, O) Specifically, this parameter represents $\frac{\partial L}{\partial y}$ where y denotes the output of this layer. Thus `backprop` need to return the following gradient, which will be passed as the upstream gradient to the previous layer: we have two trainable parameters W and b. In order to perform gradient descent, the following should be evaluated: for W

        $$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial W}$$

        and for b

        $$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial b}$$

    - Returns:
      * $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$, which has variable name `dLdx, dLdW, dLdb`. The variables `dLdW, dLdb` are passed to `nn_linear_layer` instance which uses `update_weights` method to perform a gradient descent update. Always have in mind that, `dLdx` has the same shape as `x`, `dLdW` has the same shape as `W`, and `dLdb` as `b`.

2. `nn_activation_layer`
  - `forward`: A forward pass of the input on each layer to evaluate the output.
    - input parameter: `x`: input to this layer. Shape:(B, I)
    - Returns $\sigma(x)$ where $\sigma(\cdot)$ is the sigmoid function applied to each element of x. Shape:(B, I)
  - `backprop`: backpropagation. This method takes 2 parameters:
    - Input parameters:
      * `x`. This is the input to this layer. Shape:(B, I)
      * `dLdy`. This is the upstream gradient coming from the next layer. Shape:(B, I). Specifically, this parameter represents $\frac{\partial L}{\partial y}$ where y denotes the output of this layer. Thus `backprop` need to return the following gradient, which will be passed as the upstream gradient to the previous layer:

        $$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

    - Returns:

        * $\frac{\partial L}{\partial x}$, which has variable name `dLdx` in the code. Shape:(B, I)

3. `nn_softmax_layer`
   - `forward`: A forward pass of the input on each layer to evaluate the output.
     - input parameter: `x`: input to this layer. Shape:(B, I)
     - Returns softmax(x). Shape:(B, I)
   - `backprop`: backpropagation. This method takes 2 parameters:
     - Input parameters:
       * `x`. This is the input to this layer. Shape:(B, I)
       * `dLdy`. This is the upstream gradient coming from the next layer. Shape:(B, I) Specifically, this parameter represents $\frac{\partial L}{\partial y}$ where y denotes the output of this layer. Thus `backprop` need to return the following gradient, which will be passed as the upstream gradient to the previous layer:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

     - Returns:
       * $\frac{\partial L}{\partial x}$, which has variable name `dLdx` in the code. Shape:(B, I)

4. `nn_cross_entropy_layer`
   - `forward`: A forward pass of the input on each layer to evaluate the output.
     - input parameter:
       * `x`: input to this layer. Shape:(B, I)
       * `y`: ground truth labels. Shape:$(B, 1)$
     - Returns the cross-entropy loss defined in Equation (1).
   - `backprop`: backpropagation. This method takes 3 parameters:
     - Input parameters:
       * `x`. This is the input to this layer. Shape:(B, I)
       * `y`: ground truth labels. Shape:$(B, 1)$
     - Returns:
       * $\frac{\partial L}{\partial x}$, which has variable name `dLdx` in the code. Has the same shape as x.

You are asked to complete `forward` and `backprop` methods for every class.

4.2 hyperparameters

You will need to set two hyperparameters: `lr` which is the learning rate and `num_gd_step` which is the number of steps for gradient descent iterations. You can try several values of those parameters. They are related to how fast your neural net converge to a solution. For example, if you set `lr` too small, you may need to make `num_gd_step` very large, which takes it long time to finish testing. You may try to set hyperparameters by looking at how loss function decreases over gradient descent. In the code, you can set `show_loss` variable to `True` to do that.

Caution: Setting proper learning rate and number of iteration is important. Even if you have implemented forward and backpropagation correctly, setting wrong learning rate and number of iteration may give you a bad accuracy. **Test a wide range** of learning rates (typically less than one) and iteration number (more than thousands of iterations).

5 What to submit

You will be given `hw3.py`. **You are asked to complete Q1–Q9 parts in the file.** Note: Q9 is about hyperparameters setting. Q9 has no definitive answers, and something that can be found by multiple experiments. So you can simply set those parameters that works for you. As long as your neural net provides accurate answer, any hyperparmeters are fine. But be careful that one test run takes not too long time.

- Submit a modified Python file `hw3.py`.
- Upload your `hw3.py` file at Blackboard before deadline. (Please submit the file in time, no late submission will be accepted).

6 Grading

- 16 points for the neural network works correctly. Specifically, if your accuracy is more than 80%.
- 4 points if your module does not work correctly, accuracy is less than 80%.
- 0 point if you do not submit the file by deadline.

In the blackboard, you can upload your file as many times as you like, before the deadline. The last uploaded file will be used for grading. After deadline, the submission menu will be closed and you will not be able to make submission.