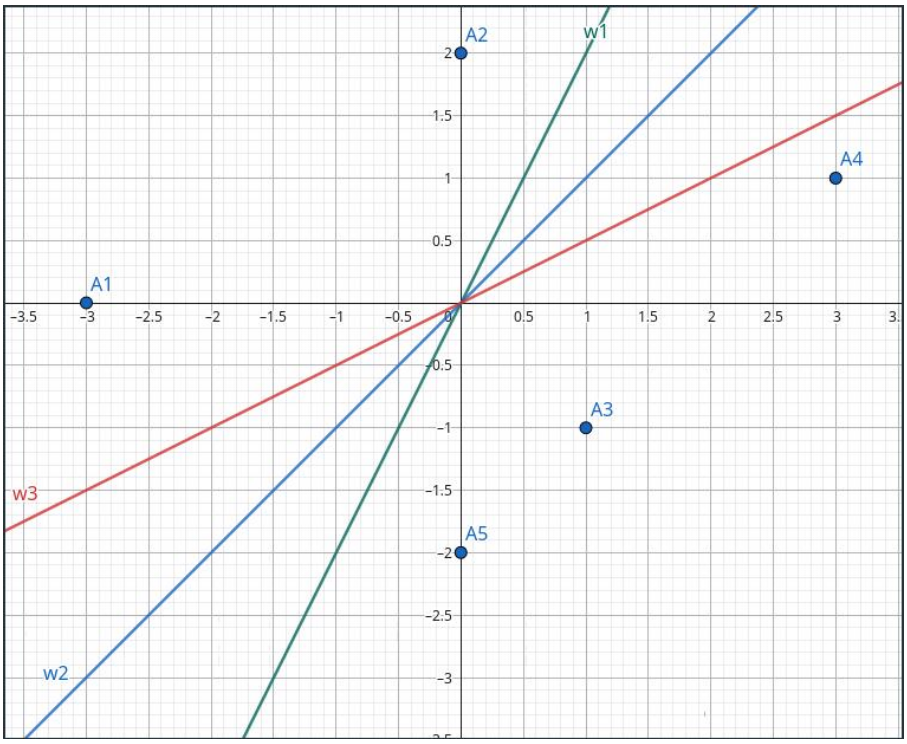
Project 5: Machine Learning

1 Binary Perceptron:

1.1 Phát biểu bài toán:

Xét không gian Euclid $\mathbb{R}^n$ với tích trong thông thường (tích chấm), cho tập hợp k vector $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_k$ và gán tùy ý mỗi vector $\mathbf{x}_i$ với một trọng số (weight) bất kỳ y_i (y_i chỉ nhận giá trị 1 hoặc -1). Hỏi có tồn tại một siêu mặt phẳng (hyperplane) nào đó có vector pháp tuyến $\mathbf{w}$ phân tách được các điểm A_i (ứng với vector $\mathbf{x}_i$) có giá trị trọng số khác nhau không?



Hình 1: Minh họa bài toán trong không gian $\mathbb{R}^2$

1.2 Hướng giải quyết bài toán:

Một siêu mặt phẳng tổng quát trong không gian vector $\mathbb{R}^n$ có phương trình dạng:

$$b + a_1x_1 + a_2x_2 + ... + a_{n-1}x_{n-1} + a_nx_n = 0$$

Vector pháp tuyến của mặt phẳng này chính là ma trận $\mathbf{w}$:

$$\mathbf{w} = \begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_{n-1} & a_n \end{bmatrix}^\top$$

Siêu mặt phẳng trên chia không gian $\mathbb{R}^n$ thành 2 nửa:

$$b + a_1x_1 + a_2x_2 + ... + a_{n-1}x_{n-1} + a_nx_n > 0 \quad (1)$$

$$b + a_1x_1 + a_2x_2 + ... + a_{n-1}x_{n-1} + a_nx_n < 0 \quad (2)$$

Xét một vector $\mathbf{x}$ bất kỳ có tọa độ:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_{n-1} & x_n \end{bmatrix}^\top$$

Từ định nghĩa tích trong thông thường, ta có:

$$\langle \mathbf{x}, \mathbf{w} \rangle = a_1 x_1 + a_2 x_2 + ... + a_{n-1} x_{n-1} + a_n x_n$$

Vậy hiển nhiên, nếu $b + \langle \mathbf{x}, \mathbf{w} \rangle > 0$ thì vector này thuộc nửa không gian (1), và $b + \langle \mathbf{x}, \mathbf{w} \rangle < 0$ thì nó thuộc nửa không gian (2).
Chọn các hệ số b và a_k ngẫu nhiên, siêu mặt phẳng luôn tồn tại với trường hợp chỉ có một vector $\mathbf{x}$. Không mất tính tổng quát ta giả sử $\mathbf{x}$ thuộc nửa không gian (1), với giá trị $y = +1$; ta quy ước gán cố định trọng số $+1$ cho nửa không gian (1), và -1 cho nửa không gian (2).
Xét trường hợp tổng quát, giả sử ta có hệ k vector $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k$ với các trọng số tương ứng $y_1, y_2, \cdots, y_k$. Sau khi thực hiện quy trình trên với vector $\mathbf{x}_1$, ta lần lượt xét các bất phương trình:

$$y_i(b + \langle \mathbf{w}, \mathbf{x}_i \rangle) > 0 \quad (i = 2, 3, \cdots, k) \tag{3}$$

Nếu (3) đúng, hiển nhiên $\mathbf{w}$ đã được chọn đúng; còn nếu (3) sai, ta cần phải điều chỉnh lại một vài tham số nào đó trong $\mathbf{w}$ cho đến khi bất phương trình trên đúng. Dễ dàng nhận thấy rằng nếu $k > n$, hệ bất phương trình có số phương trình nhiều hơn số ẩn nên tồn tại khả năng các miền nghiệm không giao nhau, dẫn đến $\mathbf{w}$ có thể không xác định.

1.3 Thuật toán học máy: Perceptron

Algorithm 1 Binary Linear Regression

Require: Training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, with $x_i \in \mathbb{R}^n$, $y_i \in \{+1, -1\}$
Ensure: Vector w will classtify all of the data points correctly (if exists), and b is a bias value

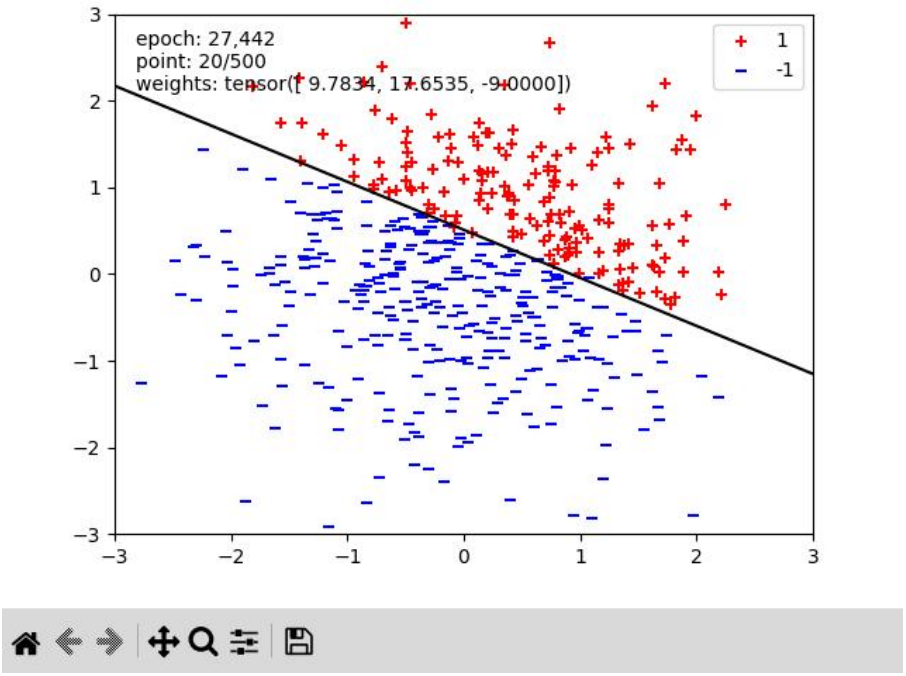
```
Initialize w ← 0
Initialize b ← 0
while true do
    mistake ← 0
    for each (x_i, y_i) ∈ D do
        if y_i · ⟨w, x_i⟩ ≤ 0 then
            w ← w + y_i x_i
            b ← b + y_i
            mistake ← mistake + 1
        end if
    end for
    if mistake = 0 then
        break
    end if
end while
return (w, b)
```

1.4 Thực thi giải thuật bằng Python

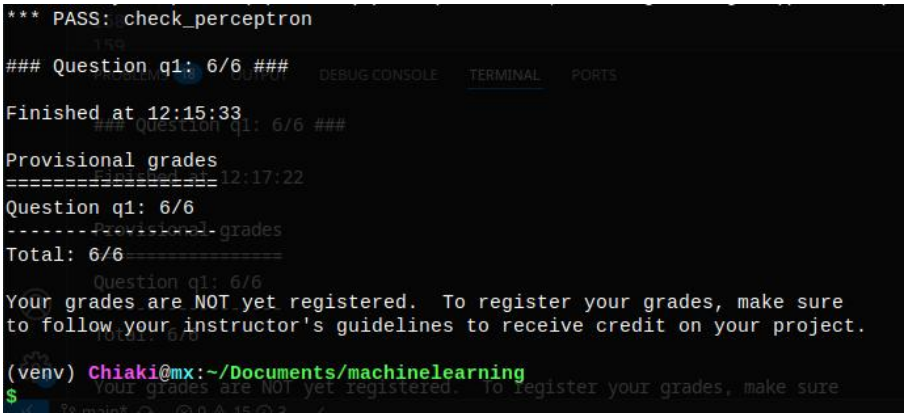
```
class PerceptronModel(Module): # class definition
    def __init__(self, dimensions): # dimensions = n
```

```
#  this function allows only one parameter, so we just ignore bias b from now on.

        super(PerceptronModel, self).__init__()
        self.w = Parameter(torch.ones(1, dimensions)) # w in R^n, x has shape (1,n)
                                                      # w is a learnable parameter

    def get_weights(self):
        return self.w   # return learned vector w

    def run(self, x):
        return (x*self.w).sum() # return inner product <w, x>

    def get_prediction(self, x):
        score = self.run(x)
        if score.item() >= 0:
            return 1
        else:
            return -1

        # checking the score item value of <w, x> positive or negative

    def train(self, dataset):   # uploading training data
        with no_grad():
            dataloader = DataLoader(dataset, batch_size=1, shuffle=True)
            while True: # implement the algorithm above
                mistakes = 0
                for sample in dataloader:
                    x = sample['x']
                    y = sample['label'].item()

                    score = self.run(x)
                    if y*score.item() <= 0:
                        self.w += y*x
                        mistakes += 1
                if mistakes == 0:
                    break
```

1.5 Kết quả



Hình 2: Quá trình thực thi code



Hình 3: Final result, 6/6 testcases passed

2 Non-linear regression:

2.1 Phát biểu bài toán:

Ước lượng xấp xỉ hàm $\sin(x)$ trên đoạn $[-2\pi, 2\pi]$ sử dụng neural network từ một tập dữ liệu rời rạc cho trước dưới dạng $(\mathbf{x}, \mathbf{y})$.
Định nghĩa: một mạng lưới neural đơn giản là một hàm ước lượng xấp xỉ gồm 2 lớp (layers), lớp phi tuyến và lớp tuyến tính. Lớp tuyến tính được sử dụng để thực hiện các phép toán tuyến tính, còn lớp phi tuyến được dùng để ước lượng xấp xỉ.

$$\mathbf{f}(\mathbf{x}) = \text{relu}(\mathbf{x} \cdot \mathbf{W}_1 + \mathbf{b}_1) \cdot \mathbf{W}_2 + \mathbf{b}_2 \tag{4}$$
$$\text{relu}(x) = \max(x, 0)$$

Ta có thể thêm nhiều lớp để ước lượng chính xác hơn:

$$\mathbf{f}(\mathbf{x}) = \text{relu}(\text{relu}(\mathbf{x} \cdot \mathbf{W}_1 + \mathbf{b}_1) \cdot \mathbf{W}_2 + \mathbf{b}_2) \cdot \mathbf{W}_3 + \mathbf{b}_3 \tag{5}$$

2.2 Hướng giải quyết bài toán:

Để đơn giản, ta chỉ xem xét hướng giải cho trường hợp neural network đơn giản nhất, từ đó tổng quát hóa bài toán với trường hợp mạng có nhiều lớp hơn.
Ta định nghĩa hàm loss như sau:

$$\mathscr{L} = \frac{1}{2N} \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{6}$$

Điều kiện lý tưởng nhất để $\mathscr{L}$ nhỏ nhất là từng hàm loss của các neuron con trong mạng lưới cũng phải đạt giá trị cực tiểu.
Ta xét hàm loss của một neural con:

$$L = \frac{1}{2}(y - f(x))^2 \tag{7}$$

Với:

$$f(z) = \text{relu}(z)w_2 + b_2$$
$$z = xw_1 + b_1$$

Ta tìm gradient descent của từng biến trong hàm L:

$$\nabla_{w_1} L = \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial z}\frac{\partial z}{\partial w_1} = (y - f(x))w_2 x 1_{z \geq 0}$$
$$\nabla_{b_1} L = \frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial z}\frac{\partial z}{\partial b_1} = (y - f(x))w_2 1_{z \geq 0}$$
$$\nabla_{w_2} L = \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial w_2} = (y - f(x))\text{relu(z)}$$
$$\nabla_{b_2} L = \frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial b_2} = (y - f(x))$$

Vector ∇L chỉ hướng có độ dốc lớn nhất trong không gian $\mathbb{R}^4$, khi ta cập nhật từng giá trị:

$$\begin{aligned} w_1 &\leftarrow w_1 - \eta \nabla_{w_1} L \\ b_1 &\leftarrow b_1 - \eta \nabla_{b_1} L \\ w_2 &\leftarrow w_2 - \eta \nabla_{w_2} L \\ b_2 &\leftarrow b_2 - \eta \nabla_{b_2} L \end{aligned} \tag{8}$$

Khi đó L dần tiến về 0, với η là một hằng số cho trước (learning rate).
Tương tự như vậy, ta có thể tổng quát hóa bài toán cho mạng n lớp.

2.3 Thuật toán học máy: Non-linear regression

Algorithm 2 Training a One-Hidden-Layer Neural Network

Require: Dataset $\{(x_i, y_i)\}_{i=1}^{N}$, learning rate $\eta > 0$
Ensure: Learned parameters W_1, b_1, W_2, b_2
  Initialize W_1, b_1, W_2, b_2 randomly
  repeat
    Forward pass:
    for $i = 1$ to N **do**
      $h_i \leftarrow x_i W_1 + b_1$
      $a_i \leftarrow \text{relu}(h_i)$
      $\hat{y}_i \leftarrow a_i W_2 + b_2$
    end for
    Compute loss:

$$L \leftarrow \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

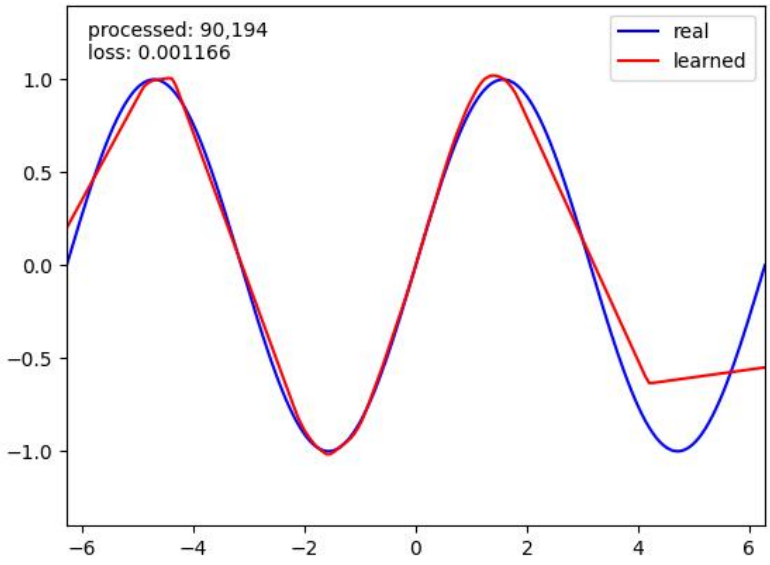
    Backward pass (compute gradients):
    Compute $\nabla_{W_2} L$, $\nabla_{b_2} L$
    Compute $\nabla_{W_1} L$, $\nabla_{b_1} L$ using chain rule
    Gradient descent update:
    $W_1 \leftarrow W_1 - \eta \nabla_{W_1} L$
    $b_1 \leftarrow b_1 - \eta \nabla_{b_1} L$
    $W_2 \leftarrow W_2 - \eta \nabla_{W_2} L$
    $b_2 \leftarrow b_2 - \eta \nabla_{b_2} L$
  until L converges (or $L < \varepsilon$)

2.4 Thực thi giải thuật bằng Python

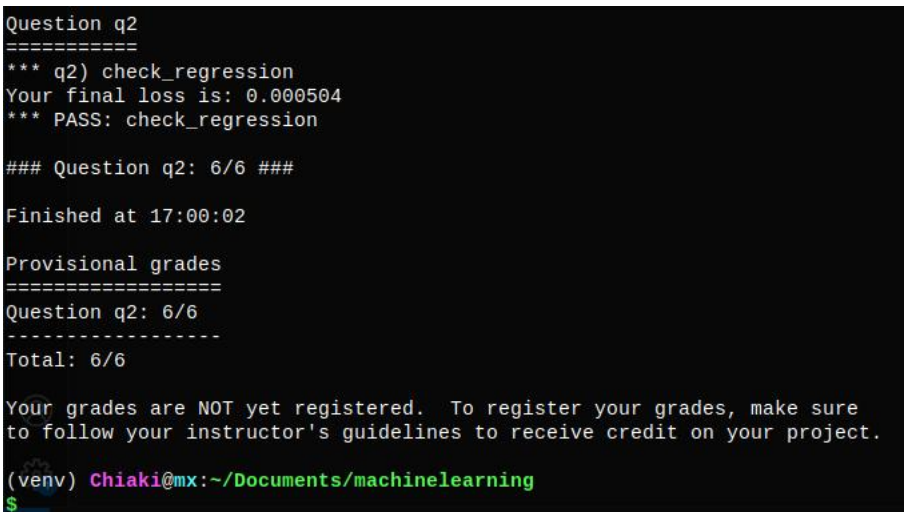
```
class RegressionModel(Module):
    def __init__(self):
        super().__init__()
        hidden_size = 200 # x in R^200
        self.fc1 = Linear(1, hidden_size) # initialize W1, b1 in R^200
                                          # and assign x = x*W1 + b1

        self.fc2 = Linear(hidden_size, 1) # initialize W2 in R^(200*1), b2 in R

    def forward(self, x):
        return self.fc2(relu(self.fc1(x))) # f(x) = relu(xw1 + b1)w2 + b2

    def get_loss(self, x, y): # loss function
        pred = self.forward(x)
        return mse_loss(pred, y)



    def train(self, dataset):
        dataloader = DataLoader(dataset, batch_size=32, shuffle=True) # 32 samples
```

```
optimizer = optim.Adam(self.parameters(), lr=0.001) # theta<-theta - eta*nabla
while True:
    total_loss = 0.0
    for sample in dataloader:
        x = sample['x']
        y = sample['label']
        optimizer.zero_grad() # assign zero to grad at first
        loss = self.get_loss(x, y) # core algorithm here
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    if total_loss < 0.02:
        break
```

2.5 Kết quả



Hình 4: Quá trình thực thi code



Hình 5: Final result