FIT3181/5215 Deep Learning

Week 04: Stochastic Gradient Descent and Optimization

Lecturer: Dr Trung Le

Trung Le and Tutor team

Email: trunglm@monash.edu

GROUP OF EIGHT AUSTRALIA

Department of Data Science and AI
Faculty of Information Technology, Monash University, Australia

Introduction

- The **purpose** of this tutorial is to practise:
  1. Logistic Regression with GD and SGD
  2. AutoGrad in PyTorch
  3. Use AutoGrad to implement Logistic Regression

Logistic Regression

- A shallow feed-forward neural network
  - Binary classifier with **one layer** and **softmax function**

- Forward propagation
  - $h = XW + b \in \mathbb{R}^{batch\_size \times 2}$ with $X \in \mathbb{R}^{batch\_size \times d}$ and $W \in \mathbb{R}^{d \times 2}$ and $b \in \mathbb{R}^{1 \times 2}$
  - $p = softmax(h, \dim = 1) = \left[p_1 = \frac{e^{h_1}}{e^{h_1}+e^{h_2}}, p_2 = \frac{e^{h_2}}{e^{h_1}+e^{h_2}}\right] \in \mathbb{R}^{batch\_size \times 2}$

- $batch\_loss = \frac{1}{batch\_size}\sum_{i=1}^{batch\_size} CE(1_{y^i}, p^i) = -\frac{1}{batch\_size}\sum_{i=1}^{batch\_size} \log p^i_{y^i}$

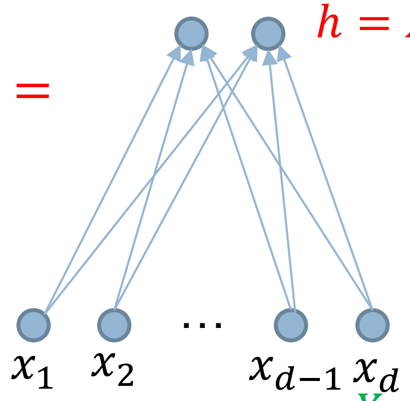
$$batch\_loss = \frac{1}{batch\_size}\sum_{i=1}^{batch\_size} CE(1_{y^i}, p^i)$$
$$= -\frac{1}{batch\_size}\sum_{i=1}^{batch\_size} \log p^i_{y^i}$$

$$\Uparrow$$

$$p = softmax(h) \in \mathbb{R}^{batch\_size \times 2}$$

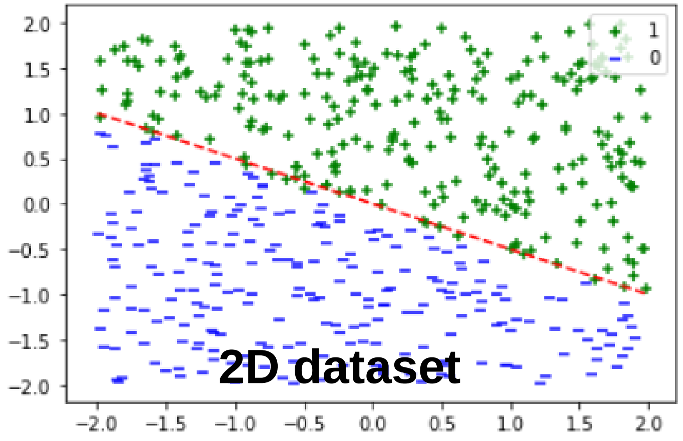
$$\Uparrow$$

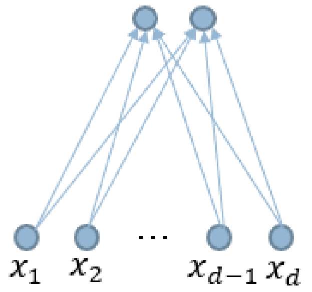
$$h = XW + b \in \mathbb{R}^{batch\_size \times 2}$$



$$X = \begin{bmatrix} x_1^1 & \dots & x_d^1 \\ x_1^2 & \dots & x_d^2 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ x_1^{bs} \dots & \dots & x_d^{bs} \end{bmatrix} \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \dots \\ \dots \\ \dots \\ y^{bs} \end{bmatrix}$$

Logistic Regression





$p = softmax(h) \in \mathbb{R}^{batch\_size \times 2}$

$\Uparrow$

$h = XW + b \in \mathbb{R}^{batch\_size \times 2}$

$$X = \begin{bmatrix} x_1^1 & \dots & x_d^1 \\ x_1^2 & \dots & x_d^2 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ x_1^{bs} \dots & \dots & x_d^{bs} \end{bmatrix} \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \dots \\ \dots \\ \dots \\ y^{bs} \end{bmatrix}$$

Forward propagation

- $h = XW + b \in \mathbb{R}^{batch\_size \times 2}$ with X $\in \mathbb{R}^{batch\_size \times d}$ and $W \in \mathbb{R}^{d \times 2}$ and $b \in \mathbb{R}^{1 \times 2}$
- $p = softmax(h, dim = 1) = \left[p_1 = \frac{e^{h_1}}{e^{h_1} + e^{h_2}}, p_2 = \frac{e^{h_2}}{e^{h_1} + e^{h_2}}\right] \in \mathbb{R}^{batch\_size \times 2}$
- $batch\_loss = \frac{1}{batch\_size} \sum_{i=1}^{batch\_size} CE(1_{y^i}, p^i) = -\frac{1}{batch\_size} \sum_{i=1}^{batch\_size} \log p_{y^i}^i$

Backward propagation

- $\frac{\partial l}{\partial p} = p - \begin{bmatrix} 1_{y^1} \\ \dots \\ 1_{y^{bs}} \end{bmatrix} \in \mathbb{R}^{batch\_size \times 2}$
- $\frac{\partial l}{\partial W} = \frac{1}{batch\_size} X^T \frac{\partial l}{\partial p} \in \mathbb{R}^{d \times 2}$
- $\frac{\partial l}{\partial b} = \frac{1}{batch\_size} sum(\frac{\partial l}{\partial p}, dim = 0)$

SGD update

- $W = W - \eta \frac{\partial l}{\partial W}$
- $b = b - \eta \frac{\partial l}{\partial b}$

AutoGrad in PyTorch

- Declare a computational graph

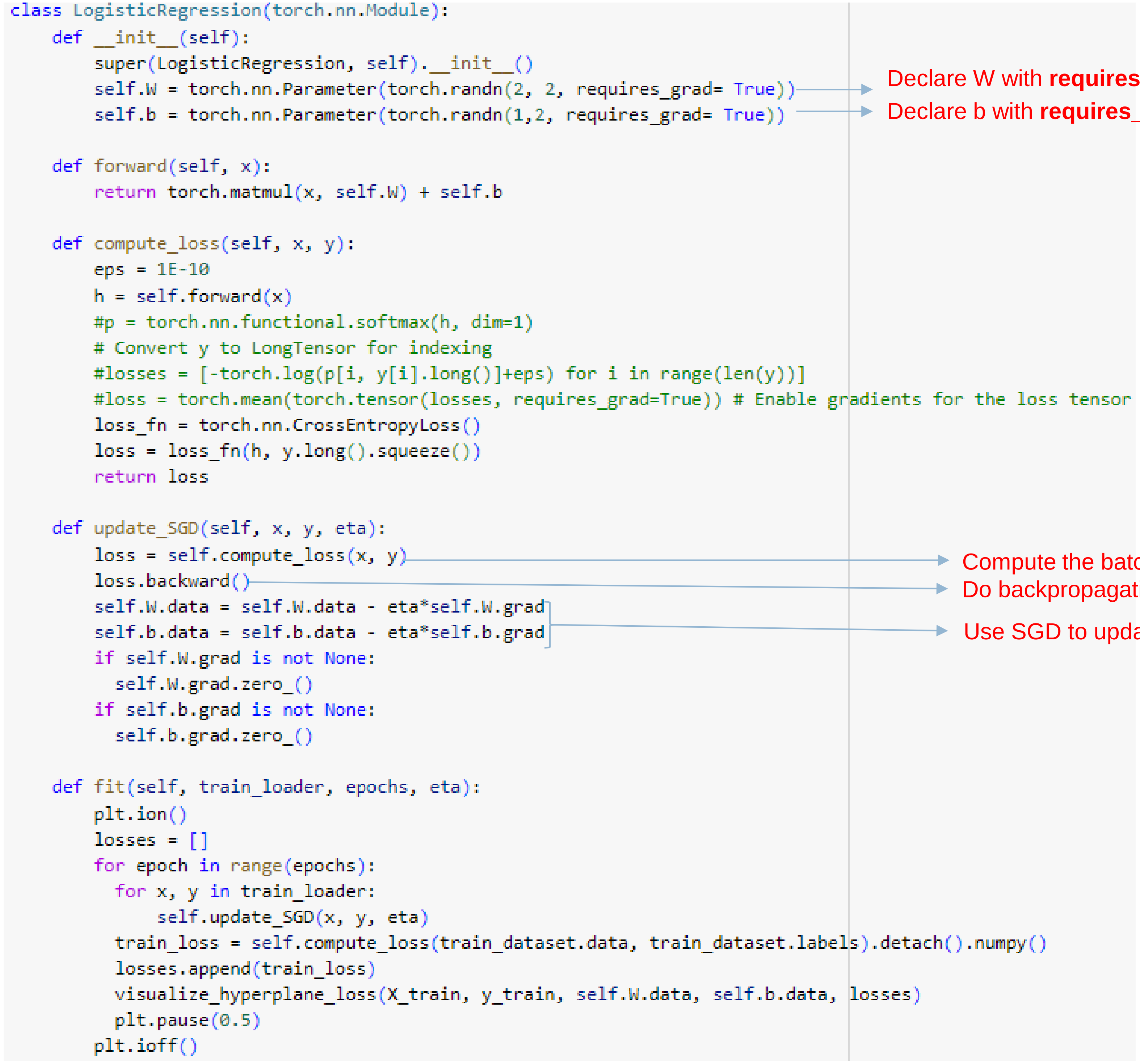
```
x = torch.tensor([1.0, 2.0, 3.0], requires_grad=False)
y = torch.tensor([4.0])
W = torch.rand(3,1, requires_grad=True)
b = torch.rand(1, requires_grad=True)
y_hat = torch.matmul(x,W) + b
l = (y_hat - y)**2
```

- Do backpropagation and compute the gradients

```
l.backward(retain_graph=True)
#l.backward() raises RuntimeError if we try to through the graph a "second time"
#suggests that you might be attempting to call the .backward() function
#on a tensor that has already had its gradients calculated and the underlying computation graph has been freed.
print(f'W_grad = {W.grad}')
print(f'W_value={W.data}')
print(f'b_grad = {b.grad}')
print(f'b_value={b.data}')
```

```
W_grad = tensor([[-0.6609],
        [-1.3217],
        [-1.9826]])
W_value=tensor([[0.6718],
        [0.2688],
        [0.6840]])
b_grad = tensor([-0.6609])
b_value=tensor([0.4080])
```

Implement Logistic Regression using AutoGrad



```python
class LogisticRegression(torch.nn.Module):
    def __init__(self):
        super(LogisticRegression, self).__init__()
        self.W = torch.nn.Parameter(torch.randn(2, 2, requires_grad= True))
        self.b = torch.nn.Parameter(torch.randn(1,2, requires_grad= True))

    def forward(self, x):
        return torch.matmul(x, self.W) + self.b

    def compute_loss(self, x, y):
        eps = 1E-10
        h = self.forward(x)
        #p = torch.nn.functional.softmax(h, dim=1)
        # Convert y to LongTensor for indexing
        #losses = [-torch.log(p[i, y[i].long()]+eps) for i in range(len(y))]
        #loss = torch.mean(torch.tensor(losses, requires_grad=True)) # Enable gradients for the loss tensor
        loss_fn = torch.nn.CrossEntropyLoss()
        loss = loss_fn(h, y.long().squeeze())
        return loss

    def update_SGD(self, x, y, eta):
        loss = self.compute_loss(x, y)
        loss.backward()
        self.W.data = self.W.data - eta*self.W.grad
        self.b.data = self.b.data - eta*self.b.grad
        if self.W.grad is not None:
          self.W.grad.zero_()
        if self.b.grad is not None:
          self.b.grad.zero_()

    def fit(self, train_loader, epochs, eta):
        plt.ion()
        losses = []
        for epoch in range(epochs):
          for x, y in train_loader:
              self.update_SGD(x, y, eta)
          train_loss = self.compute_loss(train_dataset.data, train_dataset.labels).detach().numpy()
          losses.append(train_loss)
          visualize_hyperplane_loss(X_train, y_train, self.W.data, self.b.data, losses)
          plt.pause(0.5)
        plt.ioff()
```

Declare W with **requires_grad = True**

Declare b with **requires_grad = True**

Compute the batch loss

Do backpropagation

Use SGD to update **W** and **b**

Thanks for your attention!

Enjoy the tutorial 4!