Bài tập về nhà 1
Logistic Regression & Softmax Regression

Tóm tắt nội dung

Trong bài tập này, các bạn sẽ sử dụng kiến thức đã học về logistic regression và softmax regression để giải quyết bài toán phân lớp.

1 Giới thiệu

Để có thể hoàn tất bài tập này, các bạn cần nắm rõ những kiến thức sau:

- Logistic regression là gì, nguyên tắc hoạt động ra sao.
- Softmax regression là gì và nguyên tắc hoạt động.
- Cách lấy đạo hàm cho các tham số trong hai mô hình trên.
- Giải thuật gradient descent.

Bạn có thể tham khảo lại bài giảng của lớp để nắm vững các nội dung này. Ngoài ra, các bạn có thể đặt câu hỏi cho đội ngũ giảng dạy nếu có thắc mắc.

Bài tập này sẽ gồm có hai bài chính:

- Bài 1: phân loại hai lớp dùng logistic regression.
- Bài 2: phân loại 10 lớp dùng softmax regression.

Yêu cầu dành cho các bạn trong là giải quyết hai bài trên bằng cả numpy và TensorFlow.

2 Hướng dẫn làm bài và nộp bài

2.1 Cấu trúc file

Bài tập lớn này được đi kèm với các file python sau:

- util.py: cung cấp các hàm để đọc dữ liệu trong thư mục data thành các ma trận numpy. Bạn không cần chỉnh sửa file này.

2.2 Hướng dẫn nộp bài

- logistic_np.py: cung cấp các hàm dựng sẵn để giải quyết bài 1, dùng numpy.
- softmax_np.py: dành cho bài 2, dùng numpy.
- logistic_tf.py: dành cho bài 1, nhưng code phải dùng TensorFlow thay vì numpy.
- softmax_tf.py: dành cho bài 2, nhưng code phải dùng TensorFlow thay vì numpy.
- unit_test.py: dùng để kiểm tra các chức năng cần hiện thực trong code.
- data/vehicles.dat: dữ liệu train và test của bài 1.
- data/fashion-mnist/*.gz: dữ liệu train, validation và test của bài 2. Gồm tất cả 4 file.
- data/*_unittest.npy: dữ liệu để chạy unit test cho các hàm của các bạn.

2.2 Hướng dẫn nộp bài

3 Bài 1 - Phân loại hai lớp dùng logistic regression

3.1 Dữ liệu Vehicles

Tập dữ liệu Vehicles là tập gồm có 2 lớp, được gán nhãn lớp 0 và 1. Ta có thể đọc tập dữ liệu này bằng hàm get_vehicle_data():

```
# Lưu ý: trong thư mục data cần có file vehicles.dat train_x, train_y,
test_x, test_y = get_vehicle_data()
```

Ở đây, train_x là một numpy tensor có kích thước $2400 \times 64 \times 64$ (ý nghĩa: tập dữ liệu huấn luyện train_x có 2400 mẫu, mỗi mẫu là 1 ảnh có chiều cao (height) và rộng (width) bằng 64). train_y là ma trận chứa nhãn ứng với mẫu dữ liệu trong train_x. Tương tự, test_x có kích thước $600 \times 64 \times 64$, mỗi hàng trong test_y biểu diễn cho nhãn của mỗi mẫu trong test_x. Hai tensor train_x và train_y được dùng cho việc huấn luyện mô hình phân loại; hai tensor test_x và test_y được dùng cho quá trình đánh giá (test).

Tập dữ liệu này gồm các ảnh xám (gray images), mỗi ảnh chứa một trong hai loại phương tiện di chuyển: xe máy và xe hơi (Hình 1). Mỗi ảnh có thể chứa trọn vẹn hoặc một phần phương tiện. Cần lưu ý là dữ liệu ảnh ở đây chưa được chuẩn hóa, nên các giá trị vẫn nằm trong khoảng từ 0 đến 255.

3.2 Class LogisticClassifier

Hình 1: Một số ảnh trong tập dữ liệu vehicles.

3.2 Class LogisticClassifier

Nhằm hỗ trợ cho việc lập trình, trong file có cung cấp sẵn (python) class LogisticClassifier. (Lưu ý: để tiện cho việc phân biệt giữa lớp python và lớp trong bài toán phân loại, người viết sẽ qui ước rằng khi viết class nghĩa là đang nói về python class, khi viết lớp nghĩa là đang ám chỉ lớp của dữ liệu cần phân loại)

Một trong các thành phần chính của class LogisticClassifier là biến w, chứa tham số mà ta cần tìm khi huấn luyện. Tham số này là một mảng có số hàng bằng số đặc trưng của dữ liệu đầu vào, số cột bằng 1. Cụ thể trong bài toán phân loại ảnh xe này, w sẽ là một ma trận 4096×1. w được khởi tạo ngẫu nhiên trong hàm __init__(w_shape). Để truy xuất w từ bên trong class, ta dùng self.w:

```
class logistic_classifier(object):
        def feed_forward(self, x):
                print(self.w)
```

Để truy xuất w từ bên ngoài class, ta cần có một thực thể của class và gọi thông qua thực thể này:

```
if __name__ == "__main__":
        num_feature = train_x.shape[1]
        bin_classifier = LogisticClassifier((num_feature, 1))
        print(bin_classifier.w)
```

Đối với các hàm thuộc class LogisticClassifier, việc truy xuất cũng hoàn toàn giống với w. Chúng sẽ được mô tả chi tiết trong mục tiếp theo.

3.3 Thực hiện bài 1 với numpy

Trong phần này, bạn thực hiện các bước trình bày bên dưới vào trong file tạo sẵn logistic_np.py.

3.3.1 Chuẩn hóa dữ liệu ảnh [TODO 1.1] & [TODO 1.2]

Như đã kể trên, ảnh đầu vào có giá trị từ 0 đến 255. Nếu ta đưa trực tiếp bộ ảnh vào quá trình huấn luyện sẽ làm cho gradient lớn. Vì vậy, trước khi huấn luyện, ta có thể sử dụng phương phápn chuẩn hóa dữ liệu để đưa trung bình (mean) của tập train về 0 và độ lệch chuẩn (standard deviation - std) của nó về
1.

Đối với việc xử lý hình ảnh, ta có hai cách chuẩn hóa khác nhau:

- (a) Xem mỗi pixel trong ảnh là một đặc trưng riêng rẽ. Ví dụ, pixel [1, 3] và pixel [4, 2] là hai đặc trưng khác nhau, được tính mean và std riêng.
- (b) Xem các pixel khác nhau trong ảnh là cùng 1 loại đặc trưng. Lúc này, pixel [1, 3] và pixel [4, 2] được xem là cùng 1 loại đặc trưng, được tính mean và std chung.

Trong mục này, bạn cần hiện thực cách chuẩn hóa (a) trong hàm normalize_per_pixel và cách (b) trong hàm normalize_all_pixel. Giả sử ta có m ảnh train, $x_{0..}x_{m-1}$, mỗi ảnh train có R hàng và C cột, thì mean và std tính theo cách (a) sẽ là:

$$\bar{x}_{rc} = \frac{1}{m}\sum_{i=0}^{m-1} x_{rc}^{(i)}, \quad 0 \le r \le R-1, 0 \le c \le C-1 \tag{1}$$

$$\sigma_{rc} = \sqrt{\frac{1}{m}\sum_{i=0}^{m-1}(x_{rc}^{(i)} - \bar{x}_{rc})^2} \tag{2}$$

Đối với cách (b), ta sẽ có:

$$\bar{x} = \frac{1}{mRC}\sum_{i=0}^{m-1}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} x_{rc}^{(i)} \tag{3}$$

$$\sigma = \sqrt{\frac{1}{mRC}\sum_{i=0}^{m-1}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}(x_{rc}^{(i)} - \bar{x})^2} \tag{4}$$

Sau khi có được mean và std trên toàn bộ data huấn luyện, ta chuẩn hóa các mẫu trong tập huấn luyện theo các sau:

$$x^{(i)} = (x^{(i)} - \bar{x})/\sigma \tag{5}$$

Đối với cách (a), việc này sẽ được áp dụng riêng cho từng pixel trong số $R \times C$. Với cách (b), thì ta dùng chung $\bar{x}$ và σ trong công thức (3) và (4) cho toàn bộ tất cả các pixel.

Cần lưu ý rằng $\bar{x}$ và σ chỉ được tính trên m mẫu dữ liệu huấn luyện. Sau đó, hai giá trị này sẽ được dùng lại để chuẩn hóa các mẫu dữ liệu test (và validation nếu có).

Việc tính $\bar{x}$ và σ mà có sử dụng các dữ liệu trong tập test là vi phạm nguyên tắc đánh giá các mô hình học máy.

3.3.2 Duỗi dữ liệu [TODO 1.3]

Dữ liệu ở bước trên vẫn còn ở dạng tensor 3D ($2400 \times 64 \times 64$). Để có thể thực hiện các phép nhân ma trận trong bài toán logistic regression, ta cần chuẩn chúng về dạng tensor 2D (2400×4096). Các bạn cần thực hiện bước này trong hàm reshape2D.

3.3.3 Thêm đặc trưng 1 vào dữ liệu [TODO 1.4]

Để tính tích vô hướng dễ dàng, nối thêm một cột có giá trị bằng 1 vào train_x và test_x (concatenate có axis=1). Trong file có sẵn hàm add_one và ta nên thực hiện code trong hàm này. Sau bước này, dữ liệu huấn luyện sẽ có kích thước 2400×4097.

3.3.4 Tính các giá trị phân loại [TODO 1.5]

Các giá trị phân loại, $\hat{y}$ / $\hat{y}$, sẽ được tính trong hàm feed_forward của class LogisticClassifier. Công thức tính như sau:

$$z = xw \tag{6}$$

$$\hat{y} = \frac{1}{1 + e^{-z}} \tag{7}$$

Ở đây, $w = [w_0, w_1, .., w_{4097}]^T$ là các tham số cần học (lưu trong biến self.w trong class LogisticClassifier). Lẽ ra công thức 6 được viết là $z = xw + w_{4097}$, tuy nhiên ở bước trên ta đã thêm 1 vào làm đặc trưng cuối cho tất cả các mẫu. Việc này giúp cho quá trình nhân ma trận và quản lý các biến gọn hơn.

3.3.5 Tính độ lỗi [TODO 1.6]

Việc tính độ lỗi được thực hiện trong hàm compute_loss của class LogisticCla ssifier. Công thức tính độ lỗi như sau:

$$J(w) = -\frac{1}{m} \sum_{i=0}^{m-1} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \tag{8}$$

Trong đó:

- $y^{(i)}$ là nhãn của mẫu thứ i, mẫu thuộc lớp 0 sẽ có $y^{(i)} = 0$, mẫu thuộc lớp 1 sẽ có $y^{(i)} = 1$. Ta có thể truy cập các nhãn này thông qua biến train_y và test_y.
- $\hat{y}^{(i)} \in (0, 1)$ là phần tử thứ i trong vector $\hat{y}$.

- m = 2400 là tổng số mẫu huấn luyện.

Để tính trung bình trên ma trận theo hàng hoặc cột, ta có thể sử dụng hàm np.mean() với tham số axis tương ứng.

3.3.6 Tính đạo hàm [TODO 1.7]

Để tính đạo hàm riêng cho thành phần w_j trong w trong hàm get_grad, ta dùng công thức sau:

$$\frac{\partial J(w_j)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})x_j^{(i)} \tag{9}$$

Trong trường hợp này, sau khi thêm 1 vào train_x thì ta sẽ có $0 \le j \le 4097$.

3.3.7 Cập nhật w [TODO 1.8]

Để huấn luyện được mô hình phân loại trong hàm update_weight, ta cần cập nhật w theo công thức sau:

$$w = w - \alpha \times \frac{\partial J(w)}{\partial w} \tag{10}$$

Với α là hệ số học (learning_rate).

3.3.8 Cập nhật w dùng momentum [TODO 1.9]

Giải thuật cập nhật trình bày trong Mục 3.3.7 có điểm yếu là chậm và dễ rơi vào tối ưu cục bộ. Tuy trong bài này, giải thuật đó cũng đủ để giải quyết, nhưng ta vẫn có thể sử dụng giải thuật có quán tính để việc huấn luyện diễn ra nhanh hơn.

Khởi tạo ma trận quán tính trước khi vào vòng lặp chính:

$$\Delta w = 0 \tag{11}$$

Ở đây, Δw là ma trận có kích thước bằng chính kích thước của w. Quá trình cập nhật w sẽ được diễn ra như sau:

$$\Delta w = \gamma \Delta w + \alpha \frac{\partial J(w)}{\partial w} \tag{12}$$

$$w = w - \Delta w \tag{13}$$

Với γ là hệ số quán tính (thường được đặt là 0.9).

3.3.9 Đánh giá mô hình phân loại [TODO 1.10]

Để đánh giá mô hình phân loại trên tập kiểm thử (test_x và test_y), trước tiên, ta cần thực hiện tính các giá trị phân loại trên test_x. Sau khi đã có các giá trị này, ta sử dụng các tiêu chí sau để đánh giá mô hình:

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

$$Recall = \frac{TP}{P} \tag{15}$$

$$F_1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{16}$$

Trong đó:

- Lớp positive là lớp có giá trị y = 1.
- TP (true positive) là tổng số các mẫu mà mô hình dự đoán là positive ($\hat{y} = 1$) và thực sự có nhãn là positive ($y = 1$)
- FP (false positive) là tổng số các mẫu mô hình dự đoán là positive ($\hat{y} = 1$) nhưng thực chất có nhãn là negative ($y = 0$)
- P là tổng số mẫu positive trong tập test

Nhiệm vụ của bạn trong bước này là tính các thông số trên trong hàm test. Trong hàm này, bạn chỉ cần in các kết quả ra bằng hàm print, không nhất thiết phải return. Khi tiến hành kiểm thử, người ra đề đã tính được các giá trị $Precision = 0.766$, $Recall = 0.830$ và $F_1 - score = 0.797$. Bạn hãy cố gắng hoàn thiện bài làm của mình để đạt kết quả tương tự hoặc tốt hơn.

3.3.10 Vòng lặp huấn luyện

Vòng lặp của quá trình huấn luyện được xây dựng trong đoạn code main của file. Tất cả khung sườn cho việc thực thi đã được lập trình sẵn. Ta có thể thay đổi hai tham số tác động đến quá trình huấn luyện như sau:

- num_epoch: số lượng vòng lặp cho quá trình huấn luyện.
- learning_rate: hệ số học α.
- momentum_rate: hệ số momentum γ.
- epochs_to_draw: số lượng epochs cần đạt được để vẽ đồ thị độ lỗi trong lúc huấn luyện.

3.4 Thực hiện bài 1 với TensorFlow

Để thực hiện công việc này, bạn lập trình trong file tạo sẵn logistic_tf.py. Các bước để hoàn tất bài 1 hoàn toàn tương tự như đã trình bày ở phần trước.

- [TODO 1.11]: tạo TF placeholders cho train_x và train_y.

- [TODO 1.12]: tạo TF variable để chứa w.
- [TODO 1.13]: tạo toán tử feed forward.
- [TODO 1.14]: tính cost.
- [TODO 1.15]: tạo đối tượng SGD optimizer.
- [TODO 1.16]: tính cost và cập nhật w.

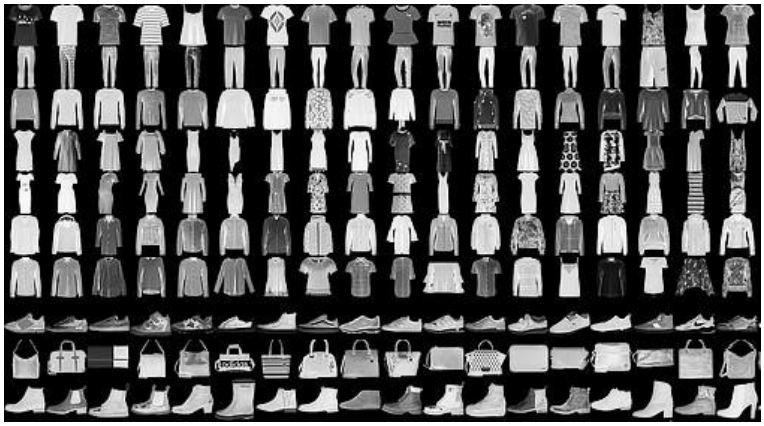
4 Bài 2 - Phân loại mười lớp

4.1 Dữ liệu fashion MNIST [1]

Ta có thể đọc tập dữ liệu này bằng hàm get_mnist_data():

```
# Lưu ý: cần có thư mục fashion-mnist trong thư mục data # Trong thư mục fashion-mnist cần có các file .gz liên quan train_x, train_y, val_x, val_y, test_x, test_y = get_mnist_data()
```

Tập dữ liệu này gồm các ảnh xám kích thước 28×28. Có tất cả 50000 ảnh train, 10000 ảnh validation và 10000 ảnh test (Hình). Mỗi ảnh thuộc một trong 10 loại quần, áo, giày, túi xách, v.v.



Hình 2: Một số mẫu ảnh trong tập dữ liệu fashion MNIST.

Lưu ý là giá trị của train_y và test_y sẽ có thể là 0, 1,.., 9 thay vì 0 và 1 như bài 1. Ngoài ra, dữ liệu này khi được đọc lên đã có dạng tensor 2D (50000×784).

4.2 Class SoftmaxClassifier

Class này thừa kế lại một số thuộc tính từ class LogisticClassifier như __init__, w. Riêng với các hàm feed_forward, compute_loss và get_grad ta cần phải hiện thực lại.

4.3 Thực hiện bài 2 với numpy

Đối với bài này, bạn cần hoàn tất các mục TODO trong file softmax_np.py.

4.3 Thực hiện bài 2 với numpy

4.3.1 Chuẩn hóa dữ liệu [TODO 2.1]

Trong bước này, ta sẽ chuẩn hóa dữ liệu train_x, val_x và test_x theo cách (b) đã đề cập trong Mục 3.3.1. Tuy nhiên, do tập dữ liệu MNIST khi load đã được đặt dưới dạng tensor 2D, nên trong công thức tính tổng chỉ còn m và $R = 784$.

4.3.2 Tiền xử lý vector label thành dạng one-hot [TODO 2.2]

Các biến train_y, val_y, test_y lúc này là một vector chứa các giá trị 0, 1,.., 9; nhưng để tính hàm lỗi của softmax regression, ta nên chuyển chúng về dạng ma trận one-hot (one-of-k). Giả sử ta có vector label có 6 phần tử, mỗi phần tử nằm trong khoảng từ 0 đến 4:

$$y = [3,4,0,0,2,1]^T \tag{17}$$

Ta sẽ có biến đổi one-hot tương ứng của nó là:

$$y = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{18}$$

Label thứ nhất có giá trị là 3, vì vậy nên trong hàng thứ nhất ở ma trận trên cột 3 có giá trị 1, tất cả các cột khác trong hàng này là 0. Tương tự cho hàng thứ 2, label là 4, nên cột 4 trong hàng 2 có giá trị là 1.

Để việc biến đổi giá trị của mảng sang dạng one-hot được nhanh chóng, ta nên sử dụng index array hay index vector trên ma trận đơn vị. Tham khảo thêm ở đây: Numpy basic indexing - Index arrays.

4.3.3 Tính các giá trị phân loại [TODO 2.3]

Để tính các giá trị phân loại trong bài này, ta hiện thực các công thức sau trong hàm feed_forward và softmax:

$$z = xw, \quad x \in R^{m \times D}, w \in R^{D \times K} \tag{19}$$

Trong đó, m là số lượng mẫu dữ liệu, D là số lượng đặc trưng của dữ liệu đầu vào (785 sau khi thêm 1 vào cuối), K là số lượng nhãn trong bài toán ta đang làm (10).

$$z_{max} = [max(z^{(0)}), max(z^{(1)}), .., max(z^{(m-1)})]^T \tag{20}$$

Tại đây, z_{max} là một vector cột (kích thước $m \times 1$).

$$z' = e^{z - z_{max}} \tag{21}$$

4.3 Thực hiện bài 2 với numpy

Trong biểu thức trên, ta sẽ dùng cột thứ nhất của z trừ cho z_{max}, cột thứ 2 của z trừ cho z_{max}, v.v. Sau đó, ta tính lũy thừa cho từng phần tử trên hiệu đã tính. Kết quả tại bước này là một ma trận z^0 có kích thước $m \times K$.

Kế tiếp, ta cần tính tổng sau:

$$s = \sum_{k=0}^{K-1} z_k'^{(i)}, \quad 0 \le i \le m-1 \tag{22}$$

Như vậy, s sẽ là vector chứa tổng từng hàng của ma trận z^0. s có kích thước $m \times 1$. Sau cùng, ta có thể tính softmax bằng cách lấy mỗi phần tử trong z^0 chia cho tổng hàng tương ứng:

$$\hat{y}_k^{(i)} = \frac{z_k'^{(i)}}{s^{(i)}}, \quad 0 \le i \le m-1, 0 \le k \le K-1 \tag{23}$$

Sau khi tổng hợp lại toàn bộ các phần tử i,k của $\hat{y}$, ta sẽ có ma trận có kích thước $m \times K$. Trong ma trận này, mỗi hàng thứ i biểu diễn vector xác suất lớp của mẫu ảnh thứ i. Vì vậy, tổng của mỗi hàng luôn bằng 1.

4.3.4 Tính độ lỗi [TODO 2.4]

Công thức tính độ lỗi category như sau:

$$J(w) = -\frac{1}{m} \sum_{i=0}^{m-1} \sum_{k=0}^{K-1} y_k^{(i)} \log \hat{y}_k^{(i)} \tag{24}$$

Trong đó:

- $y_k^{(i)}$ là phần tử hàng i cột k trong ma trận nhãn one-hot y (đã đề cập trong Mục 4.3.2).
- $\hat{y}^{(i)} \in (0,1)$ là hàng i cột k trong ma trận $\hat{y}$.

4.3.5 Tính đạo hàm [TODO 2.5]

Công thức tính đạo hàm theo từng tham số w_{jk} được biểu diễn như sau:

$$\frac{\partial J(w_{jk})}{\partial w_{jk}} = \frac{1}{m}\sum_{i=0}^{m-1} x_j^{(i)}(\hat{y}_k^{(i)} - y_k^{(i)}) \tag{25}$$

Với $0 \le j \le D-1$. Viết lại dưới dạng ma trận ta có được:

$$\frac{\partial J(w)}{\partial w} = \frac{1}{m}x^T(\hat{y} - y) \tag{26}$$

4.4 Thực hiện bài 2 với TensorFlow

4.3.6 Đề xuất điều kiện dùng vòng lập huấn luyện [TODO 2.6]

Trong bài này, ngoài việc sử dụng train_x, train_y để huấn luyện, file mẫu còn cung cấp cho các bạn các biến val_x, val_y để thực hiện việc validation. Toàn bộ các giá trị lỗi validation được lưu trong biến all_val_loss. Trong phần này, nhiệm vụ của các bạn là đề xuất ra điều kiện dừng khác (ngoài việc e đạt num_epoch) trong quá trình huấn luyện. Dựa vào độ lỗi validation, bạn có thể sử dụng các tiêu chí mình tự đề ra để tránh việc quá trình huấn luyện bị overfitting.

4.3.7 Đánh giá mô hình trên dữ liệu test [TODO 2.7]

Để đánh giá mô hình phân loại nhiều lớp, ta cần sử dụng một khái niệm gọi là confusion matrix. Giả sử ta có bài toán phân loại 3 lớp, thì confusion matrix có dạng sau:

	Lớp 1	Lớp 2	Lớp 3
Lớp 1	N_{11}	N_{12}	N_{13}
Lớp 2	N_{21}	N_{22}	N_{23}
Lớp 3	N_{31}	N_{32}	N_{33}

Trong đó, N_{kl} là tổng số mẫu thực chất thuộc lớp k và bộ phân loại phân loại thành lớp l. Bộ phân loại càng tốt thì các ô trên đường chéo chính sẽ càng cao hơn so với các ô xung quanh. Khi tiến hành kiểm thử, người ra đề đã tính được confusion matrix như sau:

0.93	0	0	0.02	0	0	0.05	0	0	0
0	0.95	0	0.05	0	0	0	0	0	0
0.02	0.02	0.59	0.02	0.22	0	0.13	0	0	0
0.05	0.03	0.03	0.82	0.05	0	0.03	0	0	0
0	0	0.07	0.05	0.77	0	0.12	0	0	0
0	0	0	0	0	0.89	0	0.09	0.02	0
0.21	0	0.08	0.02	0.17	0	0.51	0	0.02	0

0	0	0	0	0	0.04	0	0.86	0	0.1
0	0	0	0.02	0	0.04	0	0.02	0.93	0
0	0	0	0	0	0.02	0	0	0	0.98

Bạn cần đạt được kết quả tương tự hoặc tốt hơn sau khi hoàn tất bài tập 2 này.

4.4 Thực hiện bài 2 với TensorFlow

Để thực hiện công việc này, bạn lập trình trong file tạo sẵn softmax_tf.py. Các bước để hoàn tất bài 2 hoàn toàn tương tự như đã trình bày ở phần trước.

- [TODO 2.8]: tạo TF placeholders cho train_x và train_y, tạo TF variable để chứa *w*, tạo đối tượng SGD optimizer, cập nhật *w* trong vòng lặp chính.
- [TODO 2.9]: tạo toán tử feed forward.
- [TODO 2.10]: tính cost.
- [TODO 2.11]: tạo điều kiện dừng để break khỏi vòng lặp huấn luyện khi cần.

5 Kiểm tra code numpy bằng unit test

Để kiểm tra xem việc hiện thực cho các TODO là đúng hay sai, các bạn có thể sử dụng file unit_test.py đi kèm. Ta có thể sử dụng lệnh sau trong command line hoặc terminal để kiểm tra các chức năng đã hiện thực trong file logistic_np.py:

```
# Lưu ý: cần có file data/logistic_unittest.npy python unit_test.py
logistic
```

Hoặc lệnh sau để kiểm tra file softmax_np.py:

```
# Lưu ý: cần có file data/softmax_unittest.npy python unit_test.py
softmax
```

Ngoài ra, bạn có thể chỉ dùng:

```
python unit_test.py
```

Đoạn script unit test sẽ yêu cầu bạn nhập 0 (logistic) hay 1 (softmax) để chạy unit test tương ứng.

6 Chấm điểm

Điểm của bài tập về nhà này được phân bố như sau:

Bài 1 (numpy)	
TODO 1.1 - Chuẩn hóa dữ liệu TODO 1.2 - Chuẩn hóa dữ liệu TODO 1.3 - Reshape dữ liệu TODO 1.4 - Thêm đặc trưng TODO 1.5 - Tính các giá trị phân loại TODO 1.6 - Tính độ lỗi TODO 1.7 - Tính đạo hàm TODO 1.8 - Cập nhật w TODO 1.9 - Cập nhật w với momentum TODO 1.10 - Đánh giá mô hình phân loại	
Bài 1 (TensorFlow)	
TODO 1.11 - Tạo TF place holders TODO 1.12 - Tạo TF variable cho w TODO 1.13 - Tạo toán tử feed forward TODO 1.14 - Tính độ lỗi TODO 1.15 - Tạo đối tượng SGD optimizer TODO 1.16 - Tính cost và cập nhật w	
Bài 2 (numpy)	
TODO 2.1 - Chuẩn hóa dữ liệu TODO 2.2 - Tạo ma trận one-hot TODO 2.3 - Tính các giá trị phân loại TODO 2.4 - Tính độ lỗi TODO 2.5 - Tính đạo hàm TODO 2.6 - Đề xuất điều kiện dừng TODO 2.7 - Đánh giá mô hình phân loại	
Bài 2 (TensorFlow)	
TODO 2.8 - Tạo TF place holders, variables và SGD optimizer, cập nhật w TODO 2.9 - Tạo toán tử feed forward TODO 2.10 - Tính độ lỗi TODO 2.11 - Tạo điều kiện dừng	
Tổng điểm	

TÀI LIỆU

Tài liệu

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.