

Bài tập lớn 1 - Mạng Neural Network

August 2018

Tóm tắt nội dung

Trong bài tập này, các bạn sẽ sử dụng kiến thức đã học về neural network để giải quyết bài toán phân lớp (classification). Các bạn sẽ tiến hành cài đặt và thí nghiệm trên hai tập dữ liệu Batman và Fashion MNIST.

1 Giới thiệu

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

- Neural Networks - Fully connected networks là gì và nguyên tắc hoạt động.
- Giải thuật Feedforward và BackPropagation trong bài toán NN.
- Giải thuật gradient descent - Batch and Mini-batch.
- Regularization để tránh overfitting trong NN.

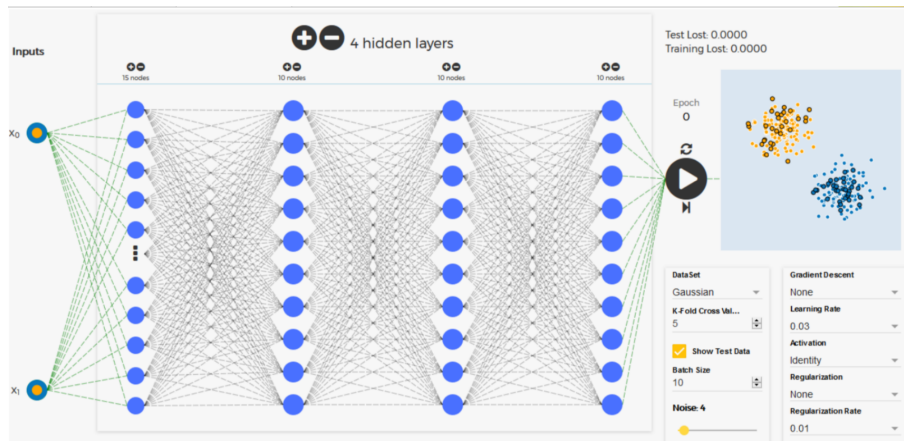
Các bạn có thể tham khảo lại bài giảng của lớp để nắm vững các nội dung này. Ngoài ra, các bạn có thể đặt câu hỏi cho đội ngũ giảng dạy nếu có thắc mắc.

Yêu cầu dành cho các bạn trong bài tập này là giải quyết hai bài toán sau bằng thư viện numpy và TensorFlow:

- Bài 1: phân loại dữ liệu ba lớp dùng neural networks.
- Bài 2: phân loại tập fashion MNIST.

Mục tiêu của bài tập lần này là hiện thực Neural Networks mạng Fully Connected một cách cơ bản trên Numpy và Tensorflow. Một mạng cơ bản sẽ gồm nhiều hidden layers và một lớp softmax tại layer cuối cùng phù hợp cho việc phân loại dữ liệu.

Khi thiết kế một mạng cơ bản số đặc trưng đầu vào cho tầng input (feature feature) sẽ được người dùng lựa chọn dựa vào tính chất dữ liệu và cách mô hình bài toán. Số output sẽ là số lớp mà người dùng muốn phân loại. Ví dụ đối với bài toán fashion MNIST, số feature đầu vào có thể chọn bằng chính bằng số pixel của mỗi ảnh, số node đầu ra sẽ là số lớp cần phân loại (10 lớp). Đối với số lượng hidden layer và số lượng node tương ứng trên mỗi layer, ta có thể tùy chọn. Một chú ý rất quan trọng là số nodes đầu ra của layer trước sẽ là số inputs đầu vào của layers sau đó.



Hình 1: Mạng neural network. Nguồn: graphicsminer.com/neuralnetwork

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.
- **activation_np.py**: Các hàm activation cơ bản cần hiện thực.
- **dnn_np.py**: Sử dụng thư viện Numpy để hiện thực các neural network giải quyết 2 bài toán trên.
- **dnn_tf.py**: Sử dụng Tensorflow để hiện thực các neural network giải quyết 2 bài toán trên.
- **submit.py**: Công cụ để nộp bài tập lên hệ thống chấm bài.
- **bat.dat**: Dữ liệu cho bài tập 1.
- **/fashion-mnist/*.gz**: Dữ liệu fashion MNIST cho bài tập 2.

2.2 Hướng dẫn nộp bài

Sau khi hoàn thành tất cả các mục còn trống trong các bài tập và kiểm tra kỹ kết quả biên dịch và thực thi trên máy, các bạn có thể tiến hành nộp bài theo 2 bước sau:

- Bước 1: Sử dụng lệnh **python submit.py** để nộp và chấm điểm 2 file **activation_np.py** và **dnn_np.py**. Hệ thống sẽ kiểm tra và trả về kết quả. Kết quả được ghi nhận sẽ là điểm số ở lần nộp bài cuối cùng.

- Bước 2: Đối với file **dnn_tf.py**, các bạn đặt trong thư mục và nén ở định dạng zip với tên file zip là họ và tên đầu đủ của bạn, sau đó gửi email về địa chỉ **teaching@vietai.org**, tiêu đề email với cú pháp [Assignment1][HoVaTen]

Hạn cuối nộp bài tập là **11 giờ 59 phút, thứ 6 ngày 7/9/2018**.

3 Hiện thực neural network bằng Numpy

3.1 Dữ liệu Batman

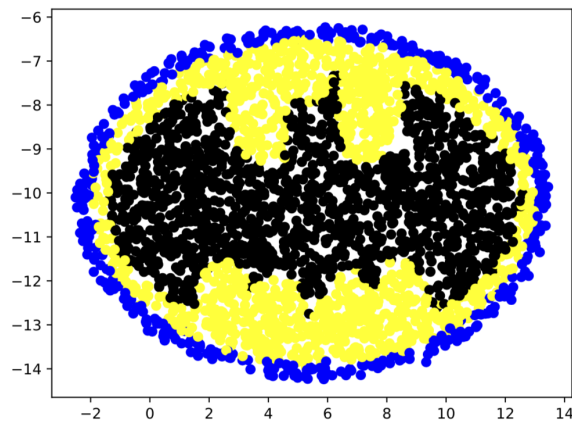
Tập dữ liệu Batman là tập gồm có 3 lớp, được gán nhãn lớp 0, 1 và 2. Ta có thể đọc tập dữ liệu này bằng hàm `get_bat_data()`:

```
# Make sure that bat.dat is in data/  
train_x, train_y, test_x, test_y = get_bat_data()
```

Để hiển thị các điểm dữ liệu huấn luyện, ta có thể dùng đoạn code sau:

```
visualize_point(train_x, train_y, y_hat)  
# Hoặc visualize_point(train_x, train_y, train_y)  
# nếu chưa tính được y_hat
```

Tập dữ liệu được phân bố như trong Hình 2.



Hình 2: Các điểm dữ liệu trong tập Batman. Đen: lớp 0, vàng: lớp 1, xanh: lớp 2.

3.2 Dữ liệu fashion MNIST

Dữ liệu này giống với dữ liệu trong bài tập softmax regression. Bạn có thể sử dụng hàm get data tương ứng để lấy dữ liệu.

3.3 Class Layer và Neural Network

Để tiện cho việc lập trình, trong file có cung cấp sẵn (python) class NeuralNet và class Layer. (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).

Class NeuralNet chứa một list các phần tử kiểu Layer. Trong đó mỗi tầng layer là hidden layer với biến w , chứa tham số mà ta cần tìm khi huấn luyện. Tham số này là một mảng $D^{(l-1)} \times D^{(l)}$, được khởi tạo ngẫu nhiên trong hàm `__init__(w_shape, reg)`. Trong đó $D^{(l-1)}$ là số lượng input từ tầng trước đó và $D^{(l)}$ là số lượng node tại tầng ẩn của ta đang có.

3.4 Các nội dung cần hiện thực

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

3.4.1 Hiện thực các activation functions cơ bản [TODO 1.1]

Hiện thực các hàm activation cơ bản trong mạng Neural Networks tại file `activation_function.py`. Trong file có sẵn hàm sigmoid, relu, tanh và softmax, ta nên thực hiện code trong hàm này. Ngoài ra ta sẽ hiện thực các hàm cơ bản tính gradient của các hàm trên với input của các hàm - `sigmoid_grad`, `reLU_grad`, và `tanh_grad`.

Công thức tính cho các hàm trên đã có trong file giảng dạy của khóa học. Với các hàm tính gradient cơ bản thì người học có thể tự chứng minh lại và thực thi code và hàm tương ứng.

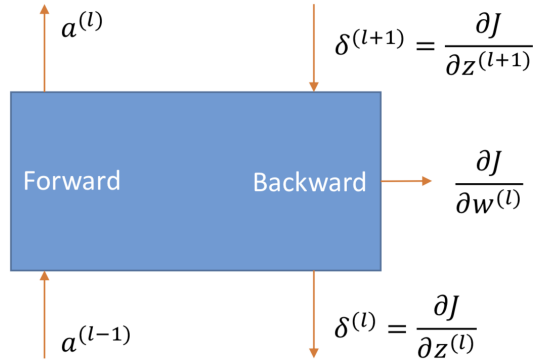
3.4.2 Hiện thực giải thuật Feedforward và Backpropagation cho từng layer [TODO 1.2]

Một số ký hiệu cần chú ý:

- L : tổng số layer (chưa tính input layer).
- l : số thứ tự của layer. $l = 1, 2, \dots, L$.
- $a^{(l-1)}$: output của layer thứ $l - 1$, input của layer thứ l .
- $a^{(0)} = x$: input đầu vào của mạng, và cũng là output của lớp thứ 0.
- $a^{(L)} = \hat{y}$: output của lớp cuối L .

- j : chỉ mục cho các input feature tại layer thứ l : $a_j^{(l-1)}$. Trong đó $j = 0, 1, \dots, D^{(l-1)} - 1$.
- k : chỉ mục cho các output feature tại layer thứ l : $a_k^{(l)}$. Trong đó $k = 0, 1, \dots, D^{(l)} - 1$.

Như vậy, tại layer thứ l , trọng số w sẽ có dạng $D^{(l-1)} \times D^{(l)}$. Ví dụ, nếu ta ký hiệu $w_{jk}^{(l)}$ thì nghĩa là trọng số này được nhân với input feature thứ j để sinh ra output thứ k .



Hình 3: Cách tiếp cận modular back propagation. Minh họa tại layer thứ l .

Với mỗi layer thì ta có thể coi như 1 module như trong Hình 3. Khi hiện thực feedforward, ta cần tính output của layer thứ l từ output của layer trước $a^{(l-1)}$ theo công thức như sau:

$$z^{(l)} = a^{(l-1)}w \quad (1)$$

$$a^{(l)} = \sigma(z^{(l)}) \quad (2)$$

Trong đó, $\sigma(z)$ là một trong các hàm activation kể trên.

Sau khi đã tính được các output cho tất cả các layer và giá trị hàm loss J , ta cần tính đạo hàm sau cho layer cuối cùng:

$$\delta^{(L)} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} = \nabla_{a^{(L)}} J \odot \sigma'(z^{(L)}) \quad (3)$$

\odot là tích Hadamard.

Sau đó, ta sẽ tiến hành lặp qua từng layer, từ L đến 1 để tính các đạo hàm sau:

- Tính $\delta^{(l)} = (\delta^{(l+1)}w^{(l+1)T}) \odot \sigma'(z^{(l)})$ (Bỏ bước này nếu đang ở layer L).

- Dựa vào $\delta_k^{(l)}$ để tính đạo hàm của J theo $w_{jk}^{(l)}$:

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = a_j^{(l-1)} \delta_k^{(l)} \quad (4)$$

Viết dưới dạng ma trận:

$$\frac{\partial J}{\partial w^{(l)}} = a^{(l-1)T} \delta^{(l)} \quad (5)$$

Sau khi thực thi xong các hàm forward và backward của class Layer. Người học có thể chạy `unit_test` của nhằm kiểm tra kết quả của đoạn mã mình đã lập trình với giá trị của gradient thông qua phương pháp số. Giải thuật numerical gradient đã được lập trình trước theo công thức sau:

$$\frac{\partial J}{\partial w_{jk}} = \frac{J(w_{jk} + \epsilon) - J(w_{jk} - \epsilon)}{2 \times \epsilon} \quad (6)$$

Nếu kết quả của relative error giữa đoạn chương trình của người học với numerical gradient kể trên $< 1e - 4$ thì người học đã thực thi đúng giải thuật back propagation cho một layer. Trong trường hợp relative error $> 1e - 2$ thì giải thuật thực thi chưa đúng và người học nhiều khả năng sẽ không chạy được toàn mạng NN cho bài toán của mình. Bước kiểm thử này người học chỉ cần thực hiện giải thuật, dữ liệu sẽ tự động sinh và giải thuật gradient checking sẽ đưa ra con số về relative error cho người học.

Trong phần code của chương trình đoạn code sau thực thi quá trình check kể trên:

```
your_layer = Layer((60, 100), 'sigmoid')
unit_test_layer(your_layer)
```

Người học có thể thay tham số (60, 100) và 'sigmoid' vào tùy ý theo mong muốn tạo ra 1 hidden layer cơ bản. Sau khi chạy hàm `unit_test_layer` - kết quả nên nhỏ hơn $1e-4$.

3.4.3 Tính độ lỗi trong hàm `compute_loss` [TODO 1.3]

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

$$J = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K (y^{(i)} = k) \log \hat{y}_k^{(i)} \quad (7)$$

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`.

- $\hat{y}_k^{(i)}$ là phần tử thứ k trong hàng thứ i của ma trận \hat{y} .
- m là tổng số mẫu huấn luyện.
- K là số lớp phân loại

Khi sử dụng hàm lỗi này, thì giá trị $\delta^{(L)}$ cần tính là:

$$\delta^{(L)} = \hat{y} - y \quad (8)$$

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.

Trong file chương trình thì hàm `compute_loss()` tại class `NeuralNet` sẽ thực thi công thức toán ở phía trên.

3.4.4 Áp dụng regularization vào mạng NN [TODO 1.4]

Trong thực tế khi train mạng NN chúng ta thường hay gặp phải trường hợp mạng học quá tốt nhưng lại không có khả năng phân loại những mẫu test. Như vậy chúng ta cần có phương pháp nhằm tránh hiện tượng này. Và L2 regularization là một trong số phương pháp hay dùng trong NN. Để sử dụng L2 regularization, ta sửa hàm lỗi thành như sau:

$$J(w) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K (y^{(i)} = k) \log \hat{y}_k^{(i)} + \frac{1}{2} \lambda \sum_{l=1}^L w^{(l)2} \quad (9)$$

Trong đó λ là giá trị của thông số regularization. Người học sẽ thiết lập hyperparameter này thông qua biến `reg` cho mỗi layer, dựa trên kinh nghiệm với những giá trị như sau: 1e-5, .., 1e-3, .., 1e-1. etc.

Việc thay đổi hàm loss cũng sẽ dẫn đến sự thay đổi giá trị gradient của hàm loss với tham số từng hàm. Quá trình update w của từng layers sẽ có thêm thành phần regularization như sau:

$$w = w - \alpha \times \left(\frac{\partial J(w)}{\partial w} + \lambda w \right) \quad (10)$$

3.5 Hiện thực giải thuật backpropagation cho class NeuralNet [TODO 1.5]

Sau khi đã thực thi forward và backward cho từng layer tại class `Layer`. Mục tiêu của người học là ghép các layer đó vào trong class `NeuralNet` nhằm thực hiện đầy đủ giải thuật backpropagation cho quá trình chạy cho toàn mạng. Quá trình forward cho mạng NN đã được thực thi tại class `NeuralNet` với hàm `forward`. Nhiệm vụ của người học là thực hiện hàm `backward` của class `NeuralNet` để tính gradient của loss tương ứng với weight của từng layers. Đoạn code sau thực hiện mẫu trước việc đó và người học sẽ implement code vào mục tương ứng.

```

def backward(self, y, all_x):
    """backward
    TODO:
    :param y: samples y = [0 0 0 1 1 1 2 2 2]
    :param all_x: input data and activation from every layer
    """

    # [TODO 1.5] Compute delta_last factor from the output
    delta_last = 0
    delta_last /= y.shape[0]

    # [TODO 1.5] Compute gradient of the loss function
    # with respect to w of softmax layer, use delta_last
    grad_last = 0

    grad_list = []
    grad_list.append(grad_last)

    for i in range(len(self.layers) - 1)[::-1]:
        prev_layer = self.layers[i+1]
        layer = self.layers[i]
        x = all_x[i]
        # [TODO 1.5] Compute delta for previous layer
        delta_prev = 0

        grad_w, delta = layer.backward(x, delta_prev)
        grad_list.append(grad_w.copy())

    grad_list = grad_list[::-1]
    return grad_list

```

3.5.1 Cập nhật w theo Batch Gradient Descent và Mini-batch Gradient Descent [TODO 1.6]

Trước tiên, để cập nhật được w sau khi có gradient, ta cần hiện thực lại các hàm `update_weight` và `update_weight_momentum`. Công thức hiện thực tương tự bài tập về nhà 1.

Kế đến, mục tiêu chính của phần này là hiện thực hàm train NN với một trong hai cách: Batch GD và Mini-batch GD.

- Với Batch Gradient Descent, việc tính toán hàm loss cũng như update giá trị w theo gradient sẽ cần với tất cả m điểm dữ liệu.
- Với giải thuật Mini-batch GD thì số điểm dữ liệu tham gia vào quá trình tính toán kể trên sẽ nhỏ hơn - 100, 200 hoặc 500. Đây là 1 hyperparameter

người học có thể tweak nhằm tạo quá trình train tốt nhất (lưu trong biến `batch_size`).

Một số định nghĩa thường gặp khi huấn luyện bằng mini-batch:

- Batch: một tập các mẫu huấn luyện.
- Batch size: số lượng mẫu được dùng để tính feedforward, backpropagation và cập nhật mạng trong mỗi vòng lặp.
- Iteration: một lần feedforward, backpropagation và cập nhật trọng số w trên một batch.
- Epoch: vòng lặp lớn. Mỗi epoch bao gồm các iteration nhỏ, mỗi iteration sẽ cho NN học qua các batch khác nhau. Khi tất cả các iteration đã phủ hết toàn bộ tất cả các mẫu trong tập huấn luyện m thì coi như kết thúc một epoch.

Ví dụ trong tập dữ liệu Batman có tất cả $m = 12000$:

- Nếu chọn `batch_size=12000`, thì một epoch bao gồm 1 iteration duy nhất.
- Nếu chọn `batch_size=120`, thì một epoch bao gồm $12000/120 = 100$ iteration.

3.5.2 Đánh giá mô hình phân loại

Để đánh giá NN đã huấn luyện, ta có thể sử dụng confusion matrix như bài tập softmax regression.

3.5.3 Huấn luyện neural network

Code load data và khởi tạo neural network cho tập dữ liệu Batman được đặt trong hàm `bat_classification`. Tương tự, code cho data mnist được đặt trong `mnist_classification`.

Để điều chỉnh quá trình huấn luyện, bạn có thể thay đổi các thuộc tính của biến `cfg`:

- `num_epoch`: số lượng vòng lặp cho quá trình huấn luyện.
- `batch_size`: số lượng mẫu trong một batch.
- `learning_rate`: hệ số học.
- `momentum_rate`: hệ số động lực γ .
- `num_train`: số lượng mẫu sẽ dùng để train trong `train_x`.
- `reg`: hệ số λ cho regularization.

- `visualize`: `True` hoặc `False`, xác định xem có vẽ đồ thị lúc huấn luyện hay không.
- `epochs_to_draw`: số epoch cần trải qua để vẽ đồ thị.

Bạn có thể đặt các tham số trên ngay lúc khởi tạo `cfg`:

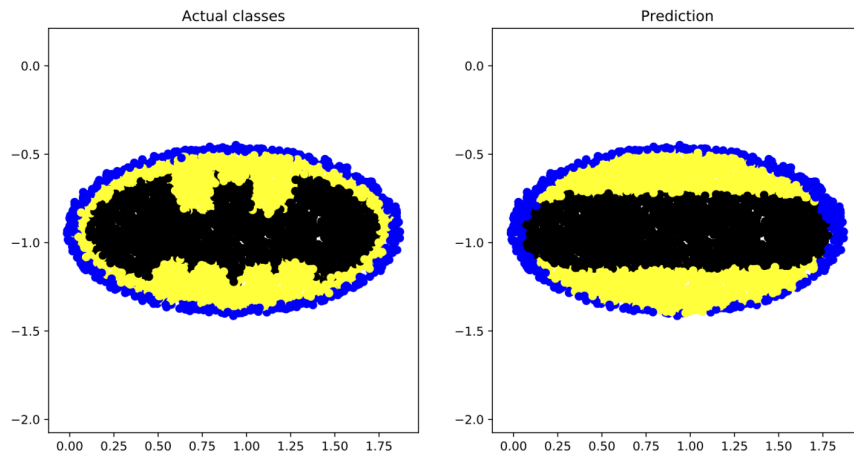
```
# Các tham số khác sẽ được gán giá trị mặc định
# Tham khảo class Config để biết thêm chi tiết
cfg= Config(learning_rate=0.001, batch_size = 120)
```

Hoặc chỉnh sửa tham số sau khi khởi tạo:

```
cfg.num_train = 1000
```

3.5.4 Các hàm hỗ trợ

Nhằm hỗ trợ việc lập trình, hai hàm hỗ trợ đã được hoàn tất sẵn gồm có:



Hình 4: Hiển thị dữ liệu dựa theo nhãn bằng hàm `visualize_point`. Dữ liệu test cho bài tập 1

- `visualize_point`: ta có thể truyền vào hàm này `train_x`, `train_y` và `y_hat` để quan sát khả năng hoạt động của bộ phân loại trên chính tập dữ liệu huấn luyện (Hình 2). Bạn có thể điều khiển tần số hiển thị bằng biến `epochs_to_draw`.

4 Sử dụng Tensorflow để huấn luyện mạng NN [TODO 1.7]

Trong bài tập này, thay vì sử dụng xây dựng lại Tensorflow graph, ta có thể sử dụng estimator của Tensorflow. Ngoài việc giúp xây dựng đồ thị, nó còn hỗ trợ việc huấn luyện NN bằng vòng lặp, tự tạo giúp session khi tính toán, v.v. Để hiện thực phần này, bạn tham khảo file `dnn_tf.py` để xem hướng dẫn cụ thể cách dùng estimator.