ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo cái bài thực hành 2

Đề tài: Hồi quy Softmax cho bài toán nhận diện chữ số viết tay

---

Môn học: Nhập môn học máy

Sinh viên thực hiện:

Tô Hữu Danh (23127336)

Nguyễn Đăng Hưng (23127050)

Lê Phú Cường (23127164)

Nguyễn Bá Đăng Khoa (23127392)

Giáo viên hướng dẫn:

Thầy Võ Nhật Tân

Ngày 4 tháng 12 năm 2025

Mục lục

Danh sách bảng

Danh sách hình vẽ

1 Mô tả và tiền xử lý dữ liệu

2 Nền tảng toán học và triển khai mô hình

2.1 Nền tảng toán học

2.1.1 Mô hình Softmax Regression

Mô hình *Softmax Regression* được sử dụng để giải quyết bài toán phân loại đa lớp (*multi-class classification*) trên tập dữ liệu MNIST. Với một mẫu đầu vào được biểu diễn bởi vector đặc trưng $x \in \mathbb{R}^d$ (trong đó d là số chiều của không gian đặc trưng), mô hình trước tiên tính toán điểm số tuyến tính (logits) tương ứng với K lớp thông qua phép biến đổi sau:

$$z = Wx + b.$$

Trong đó:

- $W \in \mathbb{R}^{K \times d}$ là ma trận trọng số;
- $b \in \mathbb{R}^K$ là vector bias;
- $z \in \mathbb{R}^K$ là vector điểm số.

Để chuyển các điểm số tuyến tính z thành một phân phối xác suất hợp lệ trên không gian nhãn, mô hình sử dụng hàm kích hoạt *softmax*. Xác suất để mẫu x thuộc lớp k được xác định bởi:

$$\hat{y}_k = P(y = k|x) = softmax(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

Trong đó, z_k là điểm số ứng với lớp k, và mẫu số $\sum_{j=1}^{K} e^{z_j}$ đảm bảo rằng tổng các xác suất bằng 1, tạo thành một phân phối xác suất chuẩn hóa trên K lớp.

2.1.2 Hàm mất mát (Loss Function)

Để huấn luyện mô hình, ta sử dụng hàm mất mát *Cross-Entropy* nhằm đo lường mức độ khác biệt giữa phân phối xác suất dự đoán và nhãn thực tế. Đồng thời, nhằm hạn chế hiện tượng quá khớp (*overfitting*), một thành phần điều chuẩn L2 (*L2 Regularization*) được bổ sung vào hàm mất mát tổng quát. Cụ thể, hàm mất mát được định nghĩa như sau:

$$J(W, b) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \hat{y}_{ik} + \frac{\lambda}{2} \sum_{l} \sum_{m} W_{lm}^2.$$

Trong đó:

- N là số lượng mẫu trong một batch;
- $y_{i,k}$ là thành phần của vector one-hot label (bằng 1 nếu mẫu i thuộc lớp k, ngược lại bằng 0);
- λ (ký hiệu là `reg` trong mã nguồn) là hệ số điều chuẩn.

Trong biểu thức trên, hạng tử cross-entropy (hạng tử thứ nhất) thúc đẩy mô hình tối đa hóa xác suất dự đoán đúng, trong khi hạng tử điều chuẩn (hạng tử thứ hai) giúp giới hạn độ lớn của các trọng số, từ đó nâng cao khả năng tổng quát hóa của mô hình.

2.1.3 Giải thuật tối ưu Gradient Descent

Quá trình huấn luyện mô hình được thực hiện bằng thuật toán *Mini-batch Gradient Descent*. Ở mỗi bước lặp, ta tính gradient của hàm mất mát đối với các tham số W và b. Nhờ đặc tính giải tích thuận lợi của hàm softmax khi kết hợp với hàm mất mát cross-entropy, các biểu thức đạo hàm thu được có dạng đơn giản và hiệu quả như sau:

$$\frac{\partial J}{\partial z} = \hat{y} - y,\ \frac{\partial J}{\partial W} = \frac{1}{N} X^T(\hat{y} - y) + \lambda W,\ \frac{\partial J}{\partial b} = \frac{1}{N} \sum (\hat{y} - y).$$

Trong đó $X \in \mathbb{R}^{N \times d}$ là ma trận dữ liệu đầu vào của mini-batch gồm N mẫu.
Quy tắc cập nhật tham số với tốc độ học α (*learning rate*) được thể hiện như sau:

$$W \leftarrow W - \alpha \frac{\partial J}{\partial W},\ b \leftarrow b - \alpha \frac{\partial J}{\partial b}.$$

Nhờ việc sử dụng mini-batch, quá trình tối ưu vừa duy trì sự ổn định của gradient, vừa đảm bảo hiệu năng tính toán phù hợp cho các tập dữ liệu lớn.
Trước khi đi vào quy trình huấn luyện lặp, ta cần xác định cách khởi tạo tham số và chi tiết các bước tính toán giá trị mất mát cũng như đạo hàm tại mỗi bước lặp. Các hàm này tương ứng với phương thức `__init__` và `compute_loss_and_grads` trong mã nguồn cài đặt.

Algorithm 1 Khởi tạo và Tính toán Loss/Gradient

function INITIALIZE($n\_features, n\_classes, lr, reg$)
  $self.lr \leftarrow lr$
  $self.reg \leftarrow reg$
  ▷ Khởi tạo ngẫu nhiên nhỏ để phá vỡ tính đối xứng
  $W \leftarrow 0.01 \times \mathcal{N}(0, 1, (n\_classes, n\_features))$
  $b \leftarrow \vec{0}_{n\_classes}$
end function

function COMPUTELOSSANDGRADS(X, y)
  $N \leftarrow \text{rows}(X)$
  1. Forward Pass:
  $scores \leftarrow XW^\top + b$
  $probs \leftarrow \text{Softmax}(scores)$
  ▷ Sử dụng max-subtraction để ổn định số học
  2. Compute Loss (Data + Regularization):
  $correct\_logprobs \leftarrow -\log(probs[\text{range}(N), y] + \epsilon)$
  $data\_loss \leftarrow \text{Mean}(correct\_logprobs)$
  $reg\_loss \leftarrow 0.5 \cdot reg \cdot \sum(W \circ W)$
  $loss \leftarrow data\_loss + reg\_loss$
  3. Backward Pass (Gradient):
  $dscores \leftarrow probs$
  $dscores[\text{range}(N), y] \leftarrow dscores[\text{range}(N), y] - 1$
  $dscores \leftarrow dscores/N$
  $\nabla_W \leftarrow dscores^\top X$
  if $reg > 0$ **then**
    $\nabla_W \leftarrow \nabla_W + reg \cdot W$
  end if
  $\nabla_b \leftarrow \text{Sum}(dscores, \text{axis} = 0)$
  return $(loss, \nabla_W, \nabla_b)$
end function

Đoạn mã giả trên mô tả chi tiết hai quá trình:

- **Initialize**: Thiết lập ma trận trọng số W với các giá trị ngẫu nhiên nhỏ theo phân phối chuẩn và bias b bằng 0. Việc này giúp mô hình hội tụ tốt hơn so với khởi tạo bằng 0.
- **ComputeLossAndGrads**: Thực hiện tính toán xuôi (forward) để lấy xác suất dự đoán, sau đó tính toán ngược (backward) để xác định gradient. Lưu ý rằng gradient của trọng số ∇_W bao gồm cả thành phần đạo hàm từ *Regularization* ($reg \cdot W$).

2.1.4 Quy trình huấn luyện mô hình

- Xáo trộn (*shuffle*) tập huấn luyện ở đầu mỗi epoch bằng một hoán vị ngẫu nhiên các chỉ số.

Algorithm 2 Hàm huấn luyện `fit`

function FIT($X_{train}, y_{train}, X_{val}, y_{val}, epochs, batch\_size$)
  $N \leftarrow \text{rows}(X_{train})$
  $history.train\_loss \leftarrow [\,]$
  $history.val\_acc \leftarrow [\,]$
  for $epoch \leftarrow 1$ to $epochs$ **do**
    $indices \leftarrow \text{RandomPermutation}(\{0, \ldots, N-1\})$
    $X_{train} \leftarrow X_{train}[indices]$
    $y_{train} \leftarrow y_{train}[indices]$
    $epoch\_loss \leftarrow 0$
    $n\_batches \leftarrow 0$
    for $start \leftarrow 0$ to N step $batch\_size$ **do**
      $end \leftarrow start + batch\_size$
      $X_{batch} \leftarrow X_{train}[start : end]$
      $y_{batch} \leftarrow y_{train}[start : end]$
      $(L, \nabla_W, \nabla_b) \leftarrow \text{ComputeLossAndGrads}(X_{batch}, y_{batch})$
      $epoch\_loss \leftarrow epoch\_loss + L$
      $n\_batches \leftarrow n\_batches + 1$
      $W \leftarrow W - lr \cdot \nabla_W$
      $b \leftarrow b - lr \cdot \nabla_b$
    end for
    $avg\_loss \leftarrow epoch\_loss / \max(n\_batches, 1)$
    $history.train\_loss.\text{append}(avg\_loss)$
    if $X_{val} \neq \varnothing$ and $y_{val} \neq \varnothing$ **then**
      $\hat{y}_{val} \leftarrow \text{Predict}(X_{val})$
      $val\_acc \leftarrow \text{Mean}(\hat{y}_{val} = y_{val})$
      $history.val\_acc.\text{append}(val\_acc)$
    end if
  end for
  return *history*
end function

- Chia dữ liệu đã xáo trộn thành các mini-batch có kích thước `batch_size`.
- Với mỗi mini-batch, gọi `ComputeLossAndGrads` để tính giá trị hàm mất mát và gradient theo W, b.
- Cập nhật tham số theo quy tắc gradient descent: $W \leftarrow W - \text{lr} \cdot \nabla_W$, $b \leftarrow b - \text{lr} \cdot \nabla_b$.
- Tính loss trung bình của toàn bộ các mini-batch trong một epoch và lưu vào `history.train_loss`.
- Nếu có tập validation, tính độ chính xác trên X_{val} và lưu vào `history.val_acc`.

2.1.5 Quy trình suy diễn và dự đoán

Algorithm 3 Hàm suy diễn `predict_proba` và `predict`

function PREDICT_PROBA(X)
  $Z \leftarrow XW^\top + b$
  $Z' \leftarrow Z - \max(Z \text{ theo từng hàng})$
  $expZ \leftarrow \exp(Z')$
  $P \leftarrow expZ / \sum(expZ \text{ theo từng hàng})$
  return P
end function

function PREDICT(X)
  $P \leftarrow \text{predict\_proba}(X)$
  $\hat{y} \leftarrow \arg\max(P \text{ theo từng hàng})$
  return $\hat{y}$
end function

- `predict_proba`:
  - Tính điểm số tuyến tính $Z = XW^\top + b$ cho tất cả mẫu và tất cả lớp.
  - Chuẩn hoá Z theo từng hàng để ổn định số học, sau đó áp dụng hàm softmax để thu được ma trận xác suất P.
- `predict`:
  - Gọi lại `predict_proba` để lấy phân phối xác suất trên các lớp cho từng mẫu.
  - Chọn lớp có xác suất lớn nhất bằng toán tử $\arg\max$ theo chiều lớp, thu được vector nhãn dự đoán $\hat{y}$.

3 Thử nghiệm thiết kế thuộc tính

4 Đánh giá và phân tích kết quả

5 Mô tả ứng dụng

6 Kết luận và nhận định

7 Tài liệu tham khảo