Training a Feed-Forward Neural Network using SGD for Binary Classification

George Michailidis

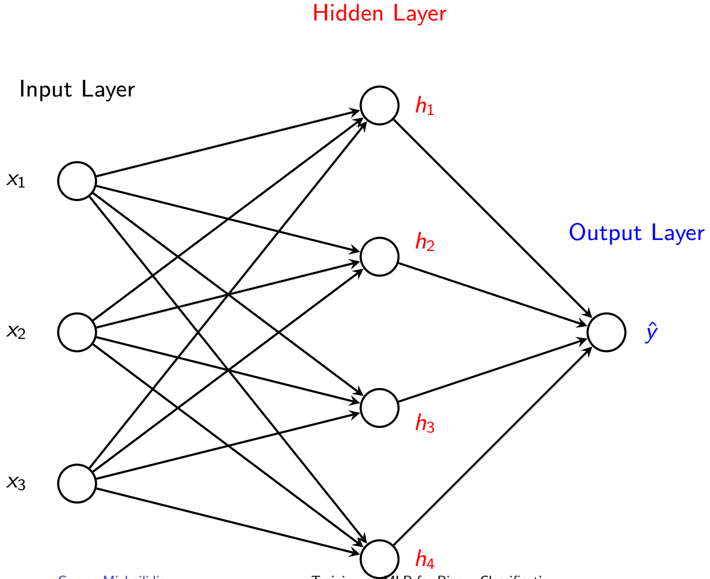
gmichail@ucla.edu

STAT 102B

Binary Classification Problem

- Given dataset $\{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^p$, $y_i \in \{0, 1\}$
- Goal: Train a model $f(x; \theta)$ to predict $\mathbb{P}(y_i = 1|x_i)$
- Popular modeling approaches
  - Logistic regression - refer to the slide deck for Lecture 3.2
  - Multilayer Perceptron (MLP) - this lecture

Visual Illustration of a Single Layer MLP



MLP Architecture

- Input: $x \in \mathbb{R}^p$ (considered as a column vector in subsequent derivations)
- Hidden layer: $h = \sigma(W_1 x + b_1)$, where $W_1 \in \mathbb{R}^{q \times p}$ are the weights and $b_1 \in \mathbb{R}^q$ the biases; hence, $h \in \mathbb{R}^q$
- Output: $\hat{y} = \sigma(W_2 h + b_2)$, $W_2 \in \mathbb{R}^{1 \times q}$, $b_2 \in \mathbb{R}$; hence, $\hat{y} \in \mathbb{R}$
- $\sigma(\cdot)$ is an activation function (e.g., ReLU for hidden layer, sigmoid for output layer)

Objective Function and Model Parameters - I

Recall from Lecture 1.1 (and also Lecture 3.1), that the objective function for the optimization problem for the linear regression problem is given by

$$\text{SSE}(\beta) = \frac{1}{2m}\|y - X\beta\|_2^2 = \frac{1}{2m}\sum_{i=1}^{m}(y_i - x_i^\top\beta)^2$$

Also recall from Lecture 3.1 that the the objective function for the optimization problem for the logistic regression problem is given by

$$-\ell(\beta) = -\frac{1}{m}\sum_{i=1}^{m}\Big[y_i(x_i^\top\beta) - \log\big(1 + \exp(x_i^\top\beta)\big)\Big]$$

Objective Function and Parameters - II

For the binary classification problem, a commonly used loss function is the Binary Cross Entropy, defined as:

$$\mathcal{L}_{\text{BCE}} = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

where:

- $y \in \{0, 1\}$ is the true label
  e.g., an image is that of a cat, $y = 1$, or a dog, $y = 0$
- $\hat{y} \in (0, 1)$ is the predicted probability of the label

Intuition for BCE:

- If $y = 1$, we want $\hat{y}$ close to 1 $\Rightarrow$ loss is small when $\log(\hat{y})$ is close to 0
- If $y = 0$, we want $\hat{y}$ close to 0 $\Rightarrow$ loss is small when $\log(1 - \hat{y})$ is close to 0

Hence, BCE heavily penalizes confident, but incorrect predictions (e.g., predicting $\hat{y} = 0.01$ when $y = 1$)

Objective Function and Parameters - III

- Binary Cross-Entropy Loss:

$$\mathcal{L}_{\text{BCE}}(\hat{y}(\theta), y) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

- Objective function for MLP

$$\min_{\theta} \mathcal{L}_{\text{BCE}}(\theta) = \min_{\theta} -\frac{1}{m} \sum_{i=1}^{m} \Big[y_i \log(\hat{y}(\theta)) + (1 - y) \log(1 - \hat{y}(\theta)) \Big]$$

where for a single layer MLP,

$$\theta = (W_1, b_1, W_2, b_2)$$

The sample size m will be specified later on

Training an MLP (i.e., estimating the parameter θ

The training of an MLP (and most neural networks) requires a forward and a backward pass

Forward Pass

- Computes predictions from inputs
- Data flows from input $\rightarrow$ hidden layers $\rightarrow$ output
- Loss is computed by comparing predictions with targets

Backward Pass (Backpropagation of the Gradient)

- Computes gradients of the loss function with respect to θ (using the chain rule)
- Gradients are used to update weights via an iterative optimization algorithm (usually SGD, or an enhanced momentum method like ADAM)

Forward Pass for a single layer MLP

Remark: for a single observation i in the training data set with the index i omitted

1. **Hidden Layer Computation**

$$h = \sigma(a), \quad \text{where } a = W_1 x + b_1$$
$$\sigma(\cdot) \;:\; \text{ReLU activation function; } \sigma(t) = \max(0, t)$$

2. **Output Layer Computation**

$$z = W_2 h + b_2$$
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

3. **Binary Cross-Entropy Loss**

$$\mathcal{L}_{\text{BCE}} = -\big(y \log \hat{y} + (1 - y) \log(1 - \hat{y})\big)$$

Backward Pass

To apply GD or its variants, we need to calculate

$$\nabla_{\theta} \mathcal{L}_{\text{BCE}}$$

i.e., the gradient of the BCE loss function with respect to the model parameter θ

Specifically, we need to calculate

1. $\nabla_{W_1} \mathcal{L}_{\text{BCE}}$
2. $\nabla_{b_1} \mathcal{L}_{\text{BCE}}$
3. $\nabla_{W_2} \mathcal{L}_{\text{BCE}}$
4. $\nabla_{b_2} \mathcal{L}_{\text{BCE}}$

It can be seen that these gradients are functions of other intermediate parameters

We will calculate these gradients from the output layer backwards; i.e., from the output layer back to the input layer
Hence, the name backward pass

Gradient of BCE Loss with respect to Output Activation

Notation abbreviation: $\mathcal{L} \equiv \mathcal{L}_{\text{BCE}}$
Calculations for a single observation i in the training data set with the index i omitted

The derivative of the BCE loss function with respect to the predicted output $\hat{y}$ is:

$$\frac{d\mathcal{L}}{d\hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

Since $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$, we have after some algebra:

$$\frac{d\hat{y}}{dz} = \hat{y}(1-\hat{y})$$

Therefore, an application of the chain rule gives:

$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{d\hat{y}} \cdot \frac{d\hat{y}}{dz} = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \cdot \hat{y}(1-\hat{y})$$

This simplifies to:

$$\frac{d\mathcal{L}}{dz} = (\hat{y} - y) \tag{1}$$

Backward Pass: Gradients with respect to W_2 and b_2

Recall that

- $z = W_2 h + b_2$
- $\hat{y} = \sigma(z)$
- h contains the "data" in the hidden layer

Based on another application of the chain rule and the result in (1) we obtain:

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{d\mathcal{L}}{dz} \cdot \frac{\partial z}{\partial W_2} = (\hat{y} - y) \cdot h^\top \tag{2}$$

$$\frac{d\mathcal{L}}{db_2} = \frac{d\mathcal{L}}{dz} \cdot \frac{dz}{db_2} \cdot 1 = \frac{d\mathcal{L}}{dz} = \hat{y} - y \tag{3}$$

Backward Pass: Gradient with respect to h

We use the chain rule:

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{d\mathcal{L}}{dz} \cdot \frac{\partial z}{\partial h}$$

Since $z = W_2 h + b_2$, we have:

$$\frac{\partial z}{\partial h} = W_2^\top$$

which combined with (1) yields

$$\frac{\partial \mathcal{L}}{\partial h} = (\hat{y} - y) \cdot W_2^\top \tag{4}$$

Backward Pass: Gradient with respect to a

Recall that

$$h = \sigma(a); \quad a = W_1 x + b_1; \quad \sigma(\cdot) = \text{ReLU activation function}$$

Note that $a \in \mathbb{R}^q, h \in \mathbb{R}^q$ (both considered as column vectors)

Since the hidden layer activation is $h = \text{ReLU}(a)$, the derivative is given element-wise by:

$$\frac{\partial h}{\partial a} = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases} \Rightarrow \mathbf{1}_{a>0},$$

where $\mathbf{1}_{a>0}$ denotes the element-wise derivative (a column vector of ones for $a_j > 0, \; j = 1, \cdots, q$ and zeros otherwise)

Then, by the chain rule, the gradient of the loss with respect to a is:

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial h} \odot \mathbf{1}_{\text{ReLU}}(a), \tag{5}$$

where $\odot$ denotes the element-wise (Hadamard) product of two vectors

Backward Pass: Gradients with respect to input layer parameters:

Recall that

$$a = W_1 x + b_1$$

Another application of the chain rule yields:

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} \cdot x^\top \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial a} \tag{7}$$

These gradients (i.e., all the expressions in (1)-(7)) are used to update the input layer weights and biases during training using gradient descent or its variants

Recap - I

Assume input $x \in \mathbb{R}^{p \times 1}$, hidden units column vector $h \in \mathbb{R}^{q \times 1}$, and a scalar output $\hat{y}$

Forward Pass

for a single observation i in the training data set with the index i omitted

Computation	**Dimension of output**
$a = W_1 x + b_1$	$(q \times 1)$
$h = \text{ReLU}(a)$	$(q \times 1)$
$z = W_2 h + b_2$	(1×1)
$\hat{y} = \sigma(z)$	(1×1)
$\mathcal{L} = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$	(1×1)

Recap - II

Backward Pass

for a single observation i in the training data set with the index i omitted

Output Layer:

Computation	Dimension of output
$\frac{d\mathcal{L}}{dz} = \hat{y} - y$	(1×1)
$\frac{\partial\mathcal{L}}{\partial W_2} = (\hat{y} - y)h^\top$	$(1 \times q)$
$\frac{d\mathcal{L}}{\partial b_2} = \hat{y} - y$	(1×1)

Recap - III

Hidden Layer:

Computation	**Dimension of output**
$\frac{\partial \mathcal{L}}{\partial h} = W_2^\top(\hat{y} - y)$	$(q \times 1)$
$\frac{\partial h}{\partial a} = \mathbf{1}_{a>0}$	$(q \times 1)$
$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial h} \odot \mathbf{1}_{a>0}$	$(q \times 1)$
$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} \cdot x^\top$	$(q \times p)$
$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial a}$	$(q \times 1)$

Stochastic Gradient Descent (SGD)

Parameter update based on a mini-batch $s = 1$ and constant step size

$$W_2^{k+1} = W_2^k - \eta \cdot \frac{\partial \mathcal{L}}{\partial W_2}(W_2^k)$$

$$b_2^{k+1} = b_2^k - \eta \cdot \frac{\partial \mathcal{L}}{\partial b_2}(b_2^k)$$

$$W_1^{k+1} = W_1^k - \eta \cdot \frac{\partial \mathcal{L}}{\partial W_1}(W_1^k)$$

$$b_1^{k+1} = b_1^k - \eta \cdot \frac{\partial \mathcal{L}}{\partial b_1}(b_1^k)$$

SGD with mini-batch size $s > 1$ - I

Forward Pass

For $i = 1 : s$ calculate:

$$
\begin{aligned}
a_i &= W_1 \mathbf{x}^{(i)} + b_1 \\
h_i &= \mathrm{ReLU}(a_i) \\
z_i &= W_2 h_i + b_2 \\
\hat{y}_i &= \sigma(z_i) = \frac{1}{1 + e^{-z_i}}
\end{aligned}
$$

end for

Calculate BCE loss for the mini-batch:

$$
\mathcal{L} = \frac{1}{s} \sum_{i=1}^{s} [-y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i)]
$$

SGD with mini-batch size $s > 1$ - II

Backward Pass: Gradients averaged over the mini-batch;
i.e., calculate gradient based on (1)-(7) for a single observation i, then sum them up and average them

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{1}{s} \sum_{i=1}^{s} (\hat{y}_i - y_i) \cdot h_i^\top$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{1}{s} \sum_{i=1}^{s} (\hat{y}^{(i)} - y^{(i)})$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{1}{s} \sum_{i=1}^{s} \left[\left((W_2^\top (\hat{y}_i - y_i)) \odot \mathbf{1}_{a_i > 0} \right) \cdot x_i^\top \right]$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{1}{s} \sum_{i=1}^{m} \left[(W_2^\top (\hat{y}_i - y_i) \odot \mathbf{1}_{a_i > 0} \right]$$

and then update the model parameters as in slide 19

Forward and Backward Pass in Matrix Notation - I

For a mini-batch of size s it is very inefficient to execute loops over the observations in the mini-batch to calculate the average gradient, especially in intepreted based languages, such as `R` or `Python`

It is more efficient to use matrix-vector operations

Forward and Backward Pass in Matrix Notation - II

- Mini-batch size: s
- Hidden layer dimension: q
- Input batch: $X \in \mathbb{R}^{s \times p}$
- True labels: $Y \in \mathbb{R}^{s \times 1}$
- Weights and biases - note that the dimensions of W_1 and W_2 have been modified to enable efficient matrix-vector computations:

$$W_1 \in \mathbb{R}^{p \times q}, \quad b_1 \in \mathbb{R}^q$$

$$W_2 \in \mathbb{R}^{q \times 1}, \quad b_2 \in \mathbb{R}$$

- Hidden activation: ReLU, denoted $\mathrm{ReLU}(\cdot)$
- Output activation: sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

Forward and Backward Pass in Matrix Notation - III

Forward Pass Computations

$$
\begin{aligned}
A &= XW_1 + \mathbf{1}_s b_1^\top && \in \mathbb{R}^{s\times q} \\
H &= \mathrm{ReLU}(A) && \in \mathbb{R}^{s\times q} \\
Z &= HW_2 + \mathbf{1}_s b_2 && \in \mathbb{R}^{s\times 1} \\
\hat{Y} &= \sigma(Z) && \in \mathbb{R}^{s\times 1}
\end{aligned}
$$

where $\mathbf{1}_s$ denotes a column vector of size s comprising of all ones

Forward and Backward Pass in Matrix Notation - III

Backward Pass Computations

$$\delta_2 = \hat{Y} - Y \qquad \in \mathbb{R}^{s \times 1}$$

$$\nabla_{W_2} \mathcal{L} = \frac{1}{s} H^\top \delta_2 \qquad \in \mathbb{R}^{q \times 1}$$

$$\nabla_{b_2} \mathcal{L} = \frac{1}{s} \sum_{i=1}^{s} \delta_2(i) \qquad \in \mathbb{R}^{1 \times 1}$$

$$\delta_1 = (\delta_2 W_2^\top) \odot \mathbf{1}_{H>0} \qquad \in \mathbb{R}^{s \times q}$$

$$\nabla_{W_1} \mathcal{L} = \frac{1}{s} X^\top \delta_1 \qquad \in \mathbb{R}^{p \times q}$$

$$\nabla_{b_1} \mathcal{L} = \frac{1}{s} \sum_{i=1}^{s} \delta_1(i) \qquad \in \mathbb{R}^{s \times 1}$$

Initialization Methods

- W_1, W_2
  - Xavier (Glorot) Initialization:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{d_{in} + d_{out}}}, \sqrt{\frac{6}{d_{in} + d_{out}}}\right)$$

    where
    - d_{in} the input dimension of a hidden layer
    - d_{out} the output dimension of a hidden layer
  - He initialization (preferable for ReLU):

$$W \sim \mathcal{N}\left(0, \frac{2}{d_{in}}\right)$$

  - Helps prevent vanishing/exploding gradients
- $b_1 = 0, b_2 = 0$

What is d_{in}, d_{out} for the MLP under consideration

Layer	**Weight Matrix**	**Dimension**	d_{in}	d_{out}
Input $\rightarrow$ Hidden	W_1	$\mathbb{R}^{d_h \times d_x}$	$d_{\text{input}} = p$	$d_{\text{hidden}} = q$
Hidden $\rightarrow$ Output	W_2	$\mathbb{R}^{1 \times d_h}$	$d_{\text{hidden}} = q$	1

Hence,

- for Xavier initialization the entries of W_1 are uniformly distributed and $d_{in} + d_{out} = q + p$
  for W_2, $d_{in} + d_{out} = 1 + q$
- for He initialization the entries of W_1 are normally distributed with $d_{in} = p$ and for W_2, $d_{in} = q$

Terminology: Epoch

In machine learning—especially in training neural networks—an epoch is a standard unit of measure that refers to one complete pass through the entire training data set by the optimization algorithm

In the case of SGD,
Epoch = one full pass through all mini-batches in the training set

How do epochs relate to gradient iterations

- GD: 1 epoch = 1 gradient iteration
- SGD: the training data set is divided in $r = m/s$ mini-batches
  Then, 1 epoch = r mini-batch gradient updates

It is customary when training neural networks to specify the number of epochs, instead of a tolerance level for the stopping criterion and monitor the objective function over epochs

Hence, if one specifies 100 epochs, then SGD performs $100r$ iterations; i.e., it updates the model parameters $100r$ times

Data Splits

- **Training Set**: Used to fit model parameters
  the size of the training set is m
- **Validation Set**: Tune hyperparameters, prevent overfitting
  for the single layer MLP under consideration, the hyperparameters are η (step size) and q (hidden layer dimension)
- **Test Set**: Evaluate generalization performance
- Typical split: 70% train, 15% val, 15% test