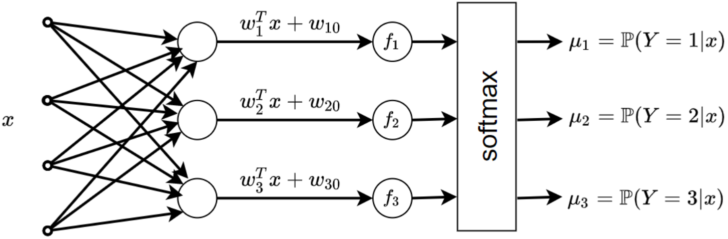
INT6151 Machine Learning
Lecture 5 - MLP

Ta Viet Cuong

VNU-UET

2024

Recap: Logistic Regression


x
$w_1^T x + w_{10}$
$w_2^T x + w_{20}$
$w_3^T x + w_{30}$
f_1
f_2
f_3
softmax
$\mu_1 = \mathbb{P}(Y = 1|x)$
$\mu_2 = \mathbb{P}(Y = 2|x)$
$\mu_3 = \mathbb{P}(Y = 3|x)$

Phân lớp 3 lớp bằng mô hình Hồi quy Logistics với hàm softmax

Formula

$$f_1 = w_1^T x + w_{10} = \mathbf{w}_1^T \mathbf{x}$$

with $\mathbf{w}_1 = \begin{bmatrix} w_1 \\ w_{10} \end{bmatrix} \in \mathbb{R}^{d+1}$ and $\mathbf{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1}$

General

$$f_k = w_k^T x + w_{k0} = \mathbf{w}_k^T \mathbf{x}$$

with $\mathbf{w}_k = \begin{bmatrix} w_k \\ w_{k0} \end{bmatrix} \in \mathbb{R}^{d+1}$

Recap: Logistic Regression

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T \mathbf{x} \\ \mathbf{w}_2^T \mathbf{x} \\ \vdots \\ \mathbf{w}_K^T \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_K^T \end{bmatrix} \mathbf{x} = \mathbf{W}\mathbf{x}$$

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_K \end{bmatrix} = \mathcal{S}(\mathbf{f})$$

$$\ell = -\sum_{k=1}^{K} y_k \log \mu_k = -\mathbf{y}^T \log(\boldsymbol{\mu})$$

Logistic Regression: Matrix formula

$$
\begin{aligned}
\mathbf{f} &= \mathbf{W}\mathbf{x} \\
\boldsymbol{\mu} &= \mathcal{S}(\mathbf{f}) \\
\ell &= -\mathbf{y}^T \log(\boldsymbol{\mu})
\end{aligned}
$$

💡 These formulas are used to program to take advantage of the parallelization of CPU, GPU

💡 The logit value f is the linear function of x so this is a simple model

Non-linearization of Logit Calculation

$$\mathbf{a}_1 = \mathbf{W}_1\mathbf{x}$$
$$\mathbf{f}_1 = \phi_1(\mathbf{a}_1)$$

The non-linear function ϕ_1 is called **activation function**.
Continue:

$$\mathbf{a}_2 = \mathbf{W}_2\mathbf{f}_1$$
$$\mathbf{f}_2 = \phi_2(\mathbf{a}_2)$$

Likewise, we can compute $a_3, f_3, a_4, f_4, \ldots$

Multi-layer perceptrons - Forward propagation

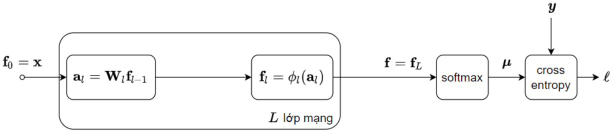
Loop L times, the initial step $\mathbf{f}_0 = \mathbf{x}$ with $l = 1, 2, \dots L$

$$\mathbf{a}_l = \mathbf{W}_l \mathbf{f}_{l-1} \in \mathbb{R}^{p_l}$$
$$\mathbf{f}_l = \phi_l(\mathbf{a}_l) \in \mathbb{R}^{p_l}$$

Calculate logit, softmax, cross-entropy:

$$\mathbf{f} = \mathbf{f}_L \in \mathbb{R}^C$$
$$\boldsymbol{\mu} = \mathcal{S}(\mathbf{f}) \in \mathbb{R}^C$$
$$\ell = -\mathbf{y}^T \log(\boldsymbol{\mu}) \in \mathbb{R}$$

y is one-hot encoder of label $y \in \{1, 2, \dots, C\}$


$\mathbf{f}_0 = \mathbf{x}$
$\mathbf{a}_l = \mathbf{W}_l \mathbf{f}_{l-1}$
$\mathbf{f}_l = \phi_l(\mathbf{a}_l)$
L lớp mạng
$\mathbf{f} = \mathbf{f}_L$
softmax
$\boldsymbol{\mu}$
$\boldsymbol{y}$
cross entropy
ℓ

Mạng nơ-ron có $L - 1$ lớp ẩn và một lớp đầu ra.

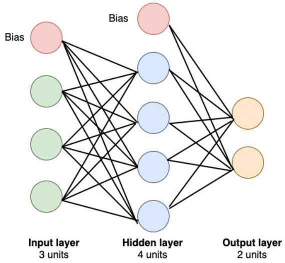
Multi-layer perceptrons - Activations

Choose the activation function

- Linear function $\phi_l(a) = a$, normally use in the last layer
- Sigmoid function $\phi_l(a) = \sigma(a) = \frac{1}{1+e^{-a}}$, normally use in the hidden layers
- Relu $\phi_l(a) = \max(0, a)$
- tanh $\phi_l(a) = 2\sigma(a) - 1$

💡 If choose $\phi_l(a) = a$ for all layers of network, that is equivalent Logistic Regression Network
Classification Rule: Choose the index having the maximum value of $\boldsymbol{f}$ for the class of $\boldsymbol{x}$

Multi-layer perceptrons - Trainable parameters



- Output of the previous layer is the input of the next layer: matrix $\mathbf{W}_l \in \mathbb{R}^{p_l \times p_{l-1}}$ with p_l is the number of output of layer l and p_{l-1} is the number of output layer $l-1$
- Let us denote $p_0 = d + 1$ as the number of inputs
- The last layer: $p_L = C$ is the number of classes of the classification problem

Neural network training

Find parameters $\mathbf{W} = (\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L)$ such that the error function ℓ is minimum.

$$\ell = -\mathbf{y}^T \log(\mathcal{S}(\phi_L(\mathbf{W}_L \phi_{L-1}(\mathbf{W}_{L-1} \ldots \phi_1(\mathbf{W}_1 \mathbf{x})))))$$

Solution: Update parameters using gradients:

$$\delta_{\mathbf{W}_l} = \frac{\partial \ell}{\partial \mathbf{W}_l}, \forall l = 1, 2, \ldots L$$

Composite function derivative

- Univariate function $(f \circ g)(x) = f(g(x))$ have derivative $(f \circ g)'(x) = f'(g(x))g'(x)$
- Multivariate multi-value function $\mathbf{f}(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^n$ have Jacobian derivative matrix:

$$\mathbf{J}_\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_d} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_d} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_d} \end{bmatrix} \in \mathbb{R}^{n \times d}$$

- The nested function: $(\mathbf{f} \circ \mathbf{g})(\mathbf{x}) = \mathbf{f}(\mathbf{g}(\mathbf{x}))$ with $\mathbf{x} \in \mathbb{R}^d, \mathbf{y} = \mathbf{g}(\mathbf{x}) \in \mathbb{R}^n, \mathbf{f}(\mathbf{y}) \in \mathbb{R}^m$

$$\mathbf{J}'_{\mathbf{f} \circ \mathbf{g}}(\mathbf{x}) = \mathbf{J}'_\mathbf{f}(\mathbf{g}(\mathbf{x}))\mathbf{J}'_\mathbf{g}(\mathbf{x})$$

The loss function derivative (Backward propagation)

We have:

$$\ell = -\mathbf{y}^T \log(\boldsymbol{\mu})$$

B1.Calculate derivative $\delta_{\boldsymbol{\mu}}$

$$\delta_{\boldsymbol{\mu}} = -\mathbf{y}^T / \boldsymbol{\mu}^T \in \mathbb{R}^{1\times C}$$

💡 row vector

Logits derivative

$$\begin{aligned}\boldsymbol{\mu} &= \mathcal{S}(\mathbf{f}) \\ \mu_i &= \frac{e^{f_i}}{\sum_{c=1}^{C} e^{f_c}}\end{aligned} \tag{1}$$

B2. Calculate derivative $\mathbf{J}_{\boldsymbol{\mu}}(\mathbf{f})$

$$\begin{aligned}\frac{\partial \mu_i}{\partial f_j} &= \frac{u'v - v'u}{v^2} = \frac{\mathbb{I}(i=j)e^{f_j}\sum_{c=1}^{C} e^{f_c} - e^{f_j}e^{f_i}}{(\sum_{c=1}^{C} e^{f_c})^2} \\ &= \mathbb{I}(i=j)\mu_j - \mu_i\mu_j = \begin{cases}(1-\mu_i)\mu_i & i=j \\ -\mu_i\mu_j & i \neq j\end{cases} \\ &= \begin{bmatrix}(1-\mu_1)\mu_1 & -\mu_2\mu_1 & \cdots & -\mu_K\mu_1 \\ -\mu_1\mu_2 & (1-\mu_2)\mu_2 & \cdots & -\mu_K\mu_2 \\ \vdots & \vdots & & \vdots \\ -\mu_1\mu_K & -\mu_2\mu_K & \cdots & (1-\mu_K)\mu_K\end{bmatrix} \in \mathbb{R}^{C\times C}\end{aligned}$$

Logits derivative

B3. Calculate derivative $\delta_{\mathbf{f}}$

$$\delta_{\mathbf{f}_L} = \delta_{\mathbf{f}} = \delta_{\boldsymbol{\mu}} \mathbf{J}_{\boldsymbol{\mu}}(\mathbf{f}) \in \mathbb{R}^{1 \times C}$$

In case of softmax and cross-entropy: $\delta_{\mathbf{f}} = \boldsymbol{\mu}^T - \mathbf{y}^T$

B4. Calculate derivative $\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l)$

- Linear function $\phi_l(a) = a$ then $\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l) = \mathbf{I} \in \mathbb{R}^{p_l \times p_l}$
- Sigmoid function $\phi_l(a) = \sigma(a)$ then $\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l) = \mathrm{diag}(f_{l1}(1 - f_{l1}), f_{l2}(1 - f_{l2}), \ldots, f_{lp_l}(1 - f_{lp_l}))$

💡 Since the activation function is computed on each element of $\mathbf{a}_l$, the matrix $\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l)$ is a diagonal matrix

Layer derivative

B5. Calculate derivative $\mathbf{J}_{\mathbf{a}_l}(\mathbf{f}_{l-1})$

$$\mathbf{a}_l = \mathbf{W}_l \mathbf{f}_{l-1}$$

$$\mathbf{J}_{\mathbf{a}_l}(\mathbf{f}_{l-1}) = \mathbf{W}_l$$

B6. Calculate derivative of a_l with W_l:

$$\frac{\partial a_{li}}{\partial \mathbf{W}_l} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ - & \mathbf{f}_{l-1}^T & - & - \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{p_l \times p_{l-1}}$$

The matrix consists of all zero rows, only the *i*th row is the input $\mathbf{f}_{l-1}^T$.

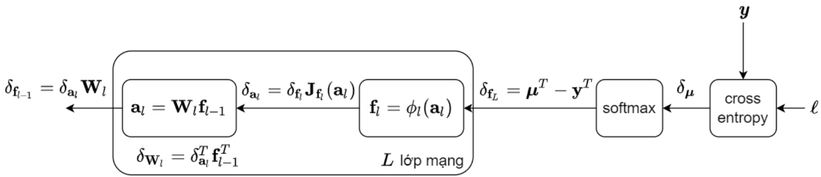
Layer derivative

B7. Calculate derivative $\delta_{\mathbf{a}_l}, \delta_{\mathbf{W}_l}, \delta_{\mathbf{f}_{l-1}}$

$$\begin{aligned}
\delta_{\mathbf{a}_l} &= \delta_{\mathbf{f}_l} \mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l) \in \mathbb{R}^{1 \times p_l} \\
\delta_{\mathbf{W}_l} &= \delta_{\mathbf{a}_l}^T \mathbf{f}_{l-1}^T \in \mathbb{R}^{p_l \times p_{l-1}} \\
\delta_{\mathbf{f}_{l-1}} &= \delta_{\mathbf{a}_l} \mathbf{W}_l \in \mathbb{R}^{1 \times p_{l-1}}
\end{aligned}$$

- The first formula have diagonal matrix
- The second obtained is based on $\frac{\partial \ell}{\partial \mathbf{W}_l} = \sum_{i=1}^{p_l} \frac{\partial \ell}{\partial a_{li}} \frac{\partial a_{li}}{\partial \mathbf{W}_l}$
- loss.backward() matrix

Back propagation (1)

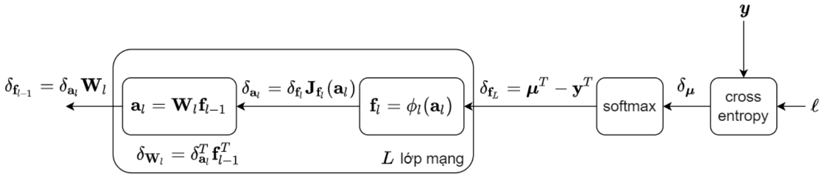

$\delta_{\mathbf{f}_{l-1}} = \delta_{\mathbf{a}_l}\mathbf{W}_l$
$\mathbf{a}_l = \mathbf{W}_l\mathbf{f}_{l-1}$
$\delta_{\mathbf{a}_l} = \delta_{\mathbf{f}_l}\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l)$
$\mathbf{f}_l = \phi_l(\mathbf{a}_l)$
$\delta_{\mathbf{f}_L} = \boldsymbol{\mu}^T - \mathbf{y}^T$
softmax
$\delta_{\boldsymbol{\mu}}$
$\boldsymbol{y}$
cross entropy
ℓ
$\delta_{\mathbf{W}_l} = \delta_{\mathbf{a}_l}^T\mathbf{f}_{l-1}^T$
L lớp mạng

Quá trình lan truyền ngược đạo hàm

1. (Forward propagation) Loop L time, convention $f_0 = x$ with $l = 1, 2, \dots L$

$$\mathbf{a}_l = \mathbf{W}_l\mathbf{f}_{l-1} \in \mathbb{R}^{p_l}$$
$$\mathbf{f}_l = \phi_l(\mathbf{a}_l) \in \mathbb{R}^{p_l}$$

Back propagation (2)

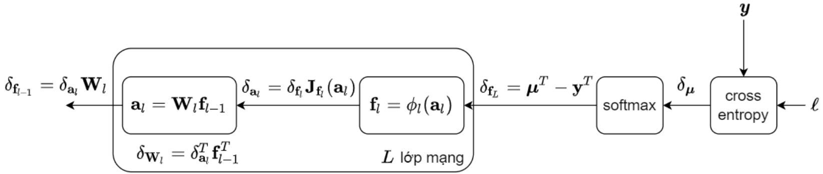

$\delta_{\mathbf{f}_{l-1}} = \delta_{\mathbf{a}_l}\mathbf{W}_l$
$\mathbf{a}_l = \mathbf{W}_l\mathbf{f}_{l-1}$
$\delta_{\mathbf{a}_l} = \delta_{\mathbf{f}_l}\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l)$
$\mathbf{f}_l = \phi_l(\mathbf{a}_l)$
$\delta_{\mathbf{f}_L} = \boldsymbol{\mu}^T - \mathbf{y}^T$
softmax
$\delta_{\boldsymbol{\mu}}$
cross entropy
$\boldsymbol{y}$
ℓ
$\delta_{\mathbf{W}_l} = \delta_{\mathbf{a}_l}^T\mathbf{f}_{l-1}^T$
L lớp mạng

Quá trình lan truyền ngược đạo hàm

2. (Output) Final Logits, probability (softmax) and loss

$$\mathbf{f} = \mathbf{f}_L \in \mathbb{R}^C$$
$$\boldsymbol{\mu} = \mathcal{S}(\mathbf{f}) \in \mathbb{R}^C$$
$$\ell = -\mathbf{y}^T\log(\boldsymbol{\mu}) \in \mathbb{R}$$

Back propagation (3)


$\boldsymbol{y}$
$\delta_{\mathbf{f}_{l-1}} = \delta_{\mathbf{a}_l}\mathbf{W}_l$
$\mathbf{a}_l = \mathbf{W}_l\mathbf{f}_{l-1}$
$\delta_{\mathbf{a}_l} = \delta_{\mathbf{f}_l}\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l)$
$\mathbf{f}_l = \phi_l(\mathbf{a}_l)$
$\delta_{\mathbf{f}_L} = \boldsymbol{\mu}^T - \mathbf{y}^T$
softmax
$\delta_{\boldsymbol{\mu}}$
cross entropy
ℓ
$\delta_{\mathbf{W}_l} = \delta_{\mathbf{a}_l}^T\mathbf{f}_{l-1}^T$
L lớp mạng

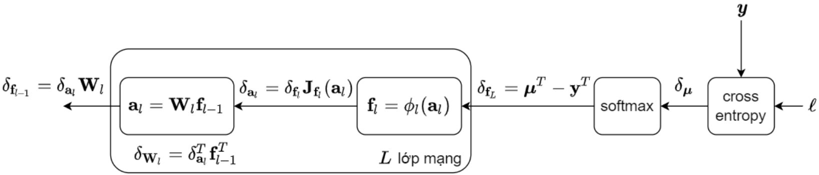
Quá trình lan truyền ngược đạo hàm

3. (Output derivative) Use derivative of loss function (the last layer)

$$\delta_{\mathbf{f}_L=}\delta_{\mathbf{f}} = \delta_{\boldsymbol{\mu}}\mathbf{J}_{\boldsymbol{\mu}}(\mathbf{f}) \in \mathbb{R}^{1\times C}$$

In case of soft-max and cross-entropy $\delta_{\mathbf{f}} = \boldsymbol{\mu}^T - \mathbf{y}^T$

Back propagation BP


$\boldsymbol{y}$
$\delta_{\mathbf{f}_{l-1}} = \delta_{\mathbf{a}_l}\mathbf{W}_l$
$\mathbf{a}_l = \mathbf{W}_l\mathbf{f}_{l-1}$
$\delta_{\mathbf{a}_l} = \delta_{\mathbf{f}_l}\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l)$
$\mathbf{f}_l = \phi_l(\mathbf{a}_l)$
$\delta_{\mathbf{f}_L} = \boldsymbol{\mu}^T - \mathbf{y}^T$
softmax
$\delta_{\boldsymbol{\mu}}$
cross entropy
ℓ
$\delta_{\mathbf{W}_l} = \delta_{\mathbf{a}_l}^T\mathbf{f}_{l-1}^T$
L lớp mạng

Quá trình lan truyền ngược đạo hàm

4. (Weight derivative - back propagation) Loop L times with $l = L, L-1, \ldots, 1$

$$\delta_{\mathbf{a}_l} = \delta_{\mathbf{f}_l}\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l) \in \mathbb{R}^{1\times p_l}$$
$$\delta_{\mathbf{W}_l} = \delta_{\mathbf{a}_l}^T\mathbf{f}_{l-1}^T \in \mathbb{R}^{p_l\times p_{l-1}}$$
$$\delta_{\mathbf{f}_{l-1}} = \delta_{\mathbf{a}_l}\mathbf{W}_l \in \mathbb{R}^{1\times p_{l-1}}$$

The results: $\delta_{\mathbf{a}_l}, \delta_{\mathbf{W}_l}, \delta_{\mathbf{f}_l}$ with $l = 1, 2, \ldots L$, specially $\delta_{\mathbf{x}} = \delta_{\mathbf{f}_0}$

How to use the results of BP

- $\delta_{\mathbf{W}_l}$: use for training
- $\delta_{\mathbf{x}}$: use for generate data with constraint
- $\delta_{\mathbf{f}_l}, \delta_{\mathbf{a}_l}$ for debugging, interpreting results, and visualizing the output of the network and its layers

Read more

- Yes you should understand backprop - Andrej Karpathy
- Cs231 understanding backprop
- Vector, Matrix, and Tensor Derivatives