Softmax Algorithm and its Extension: A Baby Neural Network

Enzhi Li

October 11, 2018

I Introduction

Softmax is a natural generalization of logistic regression algorithm which is generally used for binary classification. In contrast to logistic regression, softmax is designed for multiple classification. Given a data set of $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_m, y_m)\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{0, 1, ..., N-1\}$, our goal is to train a model that can predict the label of an unknown feature vector $\boldsymbol{x}$. Softmax gives us one such model. In this algorithm, we assume that for a certain data point $(\boldsymbol{x}, y)$, the probability of the observing label y is

$$\begin{aligned}
p(y=0) &= \frac{1}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}}, \qquad (I.1)\\
p(y=k) &= \frac{e^{\theta_k^T x}}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}}, k = 1, 2, ..., N-1.
\end{aligned}$$

Here, $x = (\boldsymbol{x}, 1)^T$ is an extended column vector that represents the feature vector, and $\theta = (\boldsymbol{w}, b)^T$ is a column vector that represents the model parameters. We can rewrite the above probability as

$$\begin{aligned}
p &= \left(\frac{1}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}}\right)^{\delta_{y,0}} \prod_{k=1}^{N-1} \left(\frac{e^{\theta_k^T x}}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}}\right)^{\delta_{y,k}} \qquad (I.2)\\
&= \left(\frac{1}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}}\right)^{\sum_{j=0}^{N-1} \delta_{y,j}} \prod_{k=1}^{N-1} e^{\theta_k^T x \delta_{y,k}}\\
&= \frac{1}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}} \prod_{k=1}^{N-1} e^{\theta_k^T x \delta_{y,k}}
\end{aligned}$$

Using the maximum likelihood estimation (MLE), we can obtain the total probability of observing the data set $\{(\boldsymbol{x}_l, y_l), l = 1, 2, ..., m\}$ as

$$\begin{aligned} P &= \prod_{l=1}^{m} p_l \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (I.3) \\ &= \prod_{l=1}^{m} \left(\frac{1}{1 + \sum_{i=1}^{N-1} e^{\theta_i^T x_l}} \right)^{\delta_{y_l,0}} \prod_{j=1}^{N-1} \left(\frac{e^{\theta_j^T x_l}}{1 + \sum_{i=1}^{N-1} e^{\theta_i^T x_l}} \right)^{\delta_{y_l,j}} \\ &= \prod_{l=1}^{m} \frac{1}{1 + \sum_{i=1}^{N-1} e^{\theta_i^T x_l}} \prod_{j=1}^{N-1} e^{\theta_j^T x_l \delta_{y_l,j}} \end{aligned}$$

With the MLE, we are to find the parameters $\theta_i, i = 1, 2, ..., N-1$ that can maximize the total probability P. This problem can be reformulated as minimizing the loss function which is defined as

$$\begin{aligned} L &= -\log P \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (I.4) \\ &= -\sum_{l=1}^{m} \left(\delta_{y_l,0} \log \frac{1}{1 + \sum_{i=1}^{N-1} e^{\theta_i^T x_l}} + \sum_{j=1}^{N-1} \delta_{y_l,j} \log \frac{e^{\theta_j^T x_l}}{1 + \sum_{i=1}^{N-1} e^{\theta_i^T x_l}} \right) \\ &= \sum_{l=1}^{m} \left(\sum_{j=0}^{N-1} \delta_{y_l,j} \log \left(1 + \sum_{i=1}^{N-1} e^{\theta_i^T x_l} \right) - \sum_{j=1}^{N-1} \delta_{y_l,j} \theta_j^T x_l \right) \\ &= \sum_{l=1}^{m} \left(\log \left(1 + \sum_{i=1}^{N-1} e^{\theta_i^T x_l} \right) - \sum_{j=1}^{N-1} \delta_{y_l,j} \theta_j^T x_l \right) \end{aligned}$$

We need to calculate the gradient of this function to find the optimal parameters that can minimize this loss function. In the next section, we are going to calculate its gradient.

II Gradient and Hessian matrix

The loss function defined in the previous section is a summation of independent functions. The summand function takes the form

$$\begin{aligned} f(\theta; x, y) &= \sum_{j=0}^{N-1} \delta_{y,j} \log \left(1 + \sum_{i=1}^{N-1} e^{\theta_i^T x} \right) - \sum_{j=1}^{N-1} \delta_{y,j} \theta_j^T x \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (II.1) \\ &= \log \left(1 + \sum_{i=1}^{N-1} e^{\theta_i^T x} \right) - \sum_{j=1}^{N-1} \delta_{y,j} \theta_j^T x \end{aligned}$$

This is a function with $\theta = (\theta_1, \theta_2, ..., \theta_{N-1})$ as variables and x, y as parameters. The gradient of this function with respect to $\theta_k, k = 1, 2, ..., N-1$ is

$$\nabla_{\theta_k} f(\theta) = \left(\frac{e^{\theta_k^T x}}{1 + \sum_{i=1}^{N-1} e^{\theta_i^T x}} - \delta_{y,k} \right) x \qquad (II.2)$$

Hessian matrix of this function is

$$\nabla^2_{\theta_{k'}\theta_k} f(\theta) = e^{\theta_k^T x}\frac{1}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}}\left(\delta_{kk'} - \frac{e^{\theta_{k'}^T x}}{1+\sum_{i=1}^{N-1} e^{\theta_i^T x}}\right) x x^T \tag{II.3}$$

In real practice, we will not use Newton's method to solve the equation, and thus Hessian matrix will never be used. Rather, we will use the improved gradient descent method, such as Adam and AdaDelta methods, to optimize our system. A Python program using Adam as optimizer has been implemented and can be found here. I have tested my program against MNIST dataset and achieved an accuracy of 91%. This result can be further improved if I introduce hidden layers into my system. Softmax can be viewed as a neural network without hidden layers. Adding one hidden layer is the topic of next section.

III Softmax with one hidden layer

In this section, I am going to add one hidden layer to my previous softmax program. I use α to denote layer index, and use W, b to denote weight matrices and biases, respectively. From this section on, I will include the biases explicitly. Now, my neural network consists of three layers: the input input layer with layer index $\alpha = 0$, one hidden layer with layer index $\alpha = 1$, and the output layer with layer index $\alpha = 2$. Weight matrix $W^{(0)}$ connects layer 0 to layer 1, and $W^{(1)}$ connects layer 1 to layer 2. The input vector to the whole network is denoted as $\boldsymbol{x}$, which is also the input vector and output vector of layer with index $\alpha = 0$. The net input vector to the hidden layer is

$$\boldsymbol{net}^{(0)} = W^{(0)} \cdot \boldsymbol{x}^{(0)} + \boldsymbol{b}^{(0)} \tag{III.1}$$

Here, $\boldsymbol{x}^{(0)}$ is the output vector of layer $\alpha = 0$. The output vector of layer $\alpha = 1$ is $\boldsymbol{x}^{(1)} = \sigma(net^{(0)})$, where σ is the activation function of the hidden layer, and is applied to vector $\boldsymbol{net}^{(0)}$ in a component-wise manner. The net input vector to layer $\alpha = 2$ is

$$\boldsymbol{net}^{(1)} = W^{(1)} \cdot \boldsymbol{x}^{(1)} + \boldsymbol{b}^{(1)} \tag{III.2}$$

The layer with layer index $\alpha = 2$ is the final output layer. According to the previous results, we know that for an input vector with label y, the loss function is

$$\begin{aligned} E &= \log\left(1 + \sum_{i=1}^{N-1} e^{net_i^{(1)}}\right) - \sum_{i=1}^{N-1} \delta_{y,i} net_i^{(1)} \qquad \text{(III.3)} \\ &:= \log\left(1 + \sum_{i=1}^{N-1} e^{\sum_j W_{ij}^{(1)} x_j^{(1)} + b_i^{(1)}}\right) - \sum_{i=1}^{N-1} \delta_{y,i}\Big(\sum_j W_{ij}^{(1)} x_j^{(1)} + b_i^{(1)}\Big) \end{aligned}$$

Here, N is the number of label categories. My task now is to minimize loss function with respect to weight matrices and biases. The gradients are

$$\begin{aligned}\frac{\partial E}{\partial W_{kl}^{(1)}} &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \frac{\partial net_m^{(1)}}{\partial W_{kl}^{(1)}} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \delta_{km} x_l^{(1)} \\ &= \frac{\partial E}{\partial net_k^{(1)}} x_l^{(1)}\end{aligned} \tag{III.4}$$

$$\begin{aligned}\frac{\partial E}{\partial b_k^{(1)}} &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \frac{\partial net_m^{(1)}}{\partial b_k^{(1)}} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \delta_{mk} \\ &= \frac{\partial E}{\partial net_k^{(1)}}\end{aligned} \tag{III.5}$$

$$\begin{aligned}\frac{\partial E}{\partial W_{kl}^{(0)}} &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \frac{\partial net_m^{(1)}}{\partial W_{kl}^{(0)}} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \sum_n \frac{\partial net_m^{(1)}}{\partial net_n^{(0)}} \frac{\partial net_n^{(0)}}{\partial W_{kl}^{(0)}} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \sum_n W_{mn}^{(1)} \sigma'(net_n^{(0)}) \delta_{kn} x_l^{(0)} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} W_{mk}^{(1)} \sigma'(net_k^{(0)}) x_l^{(0)}\end{aligned} \tag{III.6}$$

$$\begin{aligned}\frac{\partial E}{\partial b_k^{(0)}} &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \frac{\partial net_m^{(1)}}{\partial b_k^{(0)}} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \sum_n \frac{\partial net_m^{(1)}}{\partial net_n^{(0)}} \frac{\partial net_n^{(0)}}{\partial b_k^{(0)}} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} \sum_n W_{mn}^{(1)} \sigma'(net_n^{(0)}) \delta_{nk} \\ &= \sum_{m=1}^{N-1} \frac{\partial E}{\partial net_m^{(1)}} W_{mk}^{(1)} \sigma'(net_k^{(0)})\end{aligned} \tag{III.7}$$

All of these gradients can be rewritten more compactly using abstract matrix notation. They are

$$\begin{aligned}
\frac{\partial E}{\partial W^{(1)}} &= \nabla E \otimes \boldsymbol{x}^{(1)} \\
\frac{\partial E}{\partial b^{(1)}} &= \nabla E \\
\frac{\partial E}{\partial W^{(0)}} &= \nabla E \cdot \left(W^{(1)} \odot \sigma'(net^{(0)}) \right) \otimes \boldsymbol{x}^{(0)} \\
\frac{\partial E}{\partial b^{(0)}} &= \nabla E \cdot \left(W^{(1)} \odot \sigma'(net^{(0)}) \right)
\end{aligned} \tag{III.8}$$

The gradient of E means its gradient with respect to $\boldsymbol{net}^{(1)}$, the explicit expression of which is

$$\frac{\partial E}{\partial net_m^{(1)}} = \frac{e^{net_m^{(1)}}}{1 + \sum_{i=1}^{N-1} e^{net_i^{(1)}}} - \sum_{i=1}^{N-1} \delta_{y,i}\delta_{i,m} \tag{III.9}$$

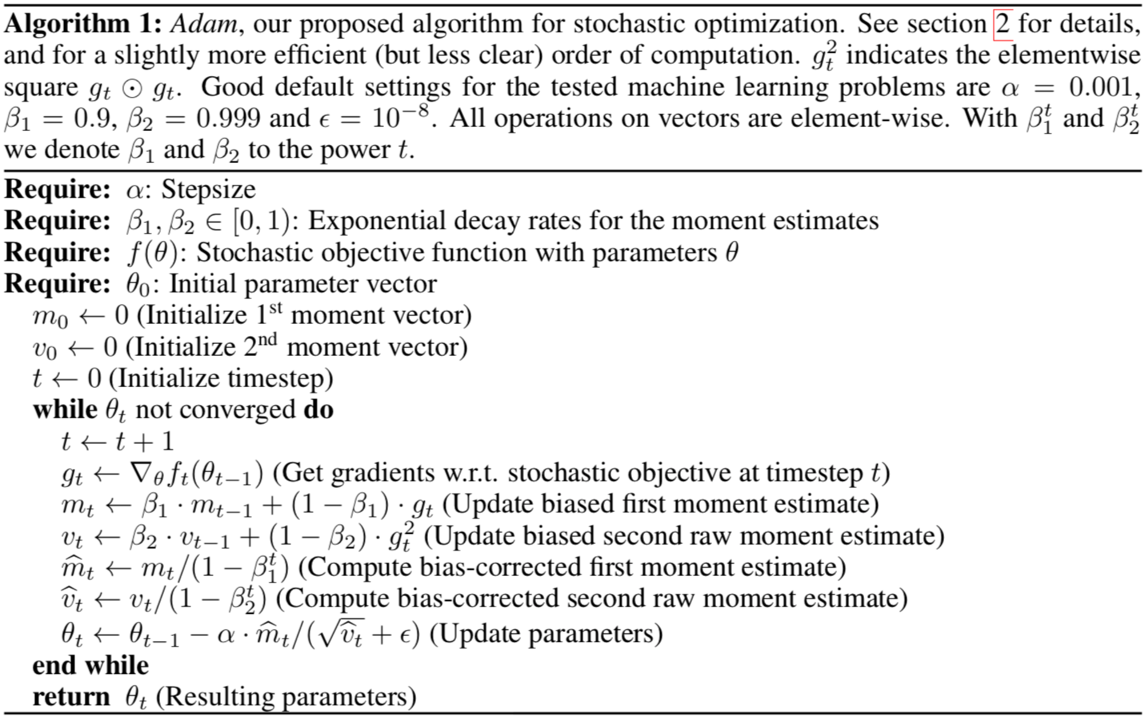
Here, I have used three kinds of matrix-vector products: dot product between vector and matrix which is denoted as $\cdot$, component-wise product between matrix and vector which is denoted as $\odot$, and dyadic product between column vectors which is denoted as $\otimes$. For sake of simplicity, here I list their explicit representations:

$$\begin{aligned}
a = v \cdot B &\leftrightarrow a_j = \sum_i v_i B_{ij} \\
a = u \odot v &\leftrightarrow a_i = u_i v_i \\
A = M \odot v &\leftrightarrow A_{ij} = M_{ij} v_j \\
A = u \otimes v &\leftrightarrow A_{ij} = u_i v_j
\end{aligned} \tag{III.10}$$

The notations used here do not all obey associative laws, and thus brackets are needed in Equation [III.8] to avoid any possible confusion. It is obvious from Equation [III.8] that $W^{(\alpha)} \odot \sigma'(net^{(\alpha-1)})$ is an operational unit, and adding more layers to the network is equivalent to adding this operational unit to the gradient formula. Now that I have calculated all the parameter gradients, I can use gradient descent method and its variants to optimize the loss function. Gradient descent method is

$$\begin{pmatrix} W^{(1)} \\ b^{(1)} \\ W^{(0)} \\ b^{(0)} \end{pmatrix} \leftarrow \begin{pmatrix} W^{(1)} \\ b^{(1)} \\ W^{(0)} \\ b^{(0)} \end{pmatrix} - \eta \begin{pmatrix} \frac{\partial E}{\partial W^{(1)}} \\ \frac{\partial E}{\partial b^{(1)}} \\ \frac{\partial E}{\partial W^{(0)}} \\ \frac{\partial E}{\partial b^{(0)}} \end{pmatrix} \tag{III.11}$$

In reality, we use improved variants of this method, such as Adam and AdaDelta, to accelerate convergence. Adam algorithm as described by its inventors is shown in Fig. [III.1]. My next task is to implement the algorithm derived here. The program can be found here.



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure III.1: ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, Diederik P. Kingma, and Jimmy Lei Ba

IV Adding more hidden layers

Adding more hidden layers is equivalent to adding more operational units to Equation [III.8], the operational unit being $W \odot \sigma'$. Assume now I have L layers in total. Then there will be $L-2$ hidden layers, the other two being the input layer and output layer. Input layer has layer index 0, and output layer has layer index $L-1$. The total number of weight matrices is $L-1$, and the number of bias terms is equal to the number of weight matrices. Weight matrix $W^{(\alpha)}, \alpha = 0, 1, 2, ..., L-2$ connects layer α and layer $\alpha+1$. Vector $\boldsymbol{x}^{(\alpha)}, \alpha = 0, 1, ..., L-1$ is the output vector of layer α. $\boldsymbol{x}^{(0)}$ is also the input vector to the whole model, and $\boldsymbol{x}^{(L-1)}$ is the output vector of the whole model. Here, I assume that all the activation functions for each hidden layer and for the input layer are σ, which is required to be differentiable. $\boldsymbol{net}^{(\alpha)} = W^{(\alpha)} \cdot \boldsymbol{x}^{(\alpha)} + b^{(\alpha)}$ is the net input to layer $\alpha + 1$. $\boldsymbol{x}^{(\alpha)} = \sigma(\boldsymbol{net}^{(\alpha-1)})$ is the output vector for layer α, where σ is applied to net input vector $\boldsymbol{net}^{(\alpha-1)}$

in a component-wise manner. The gradient formulae are

$$
\begin{aligned}
&\frac{\partial E}{\partial W^{(L-2)}} = \nabla E \otimes \boldsymbol{x}^{(L-2)} \\
&\frac{\partial E}{\partial b^{(L-2)}} = \nabla E \\
&\frac{\partial E}{\partial W^{(L-3)}} = \nabla E \cdot \Big(W^{(L-2)} \odot \sigma'(net^{(L-3)})\Big) \otimes \boldsymbol{x}^{(L-3)} \\
&\frac{\partial E}{\partial b^{(L-3)}} = \nabla E \cdot \Big(W^{(L-2)} \odot \sigma'(net^{(L-3)})\Big) \\
&... \\
&\frac{\partial E}{\partial W^{(1)}} = \nabla E \cdot \Big(W^{(L-2)} \odot \sigma'(net^{(L-3)})\Big) \cdot \Big(W^{(L-3)} \odot \sigma'(net^{(L-4)})\Big) \cdot ... \cdot \Big(W^{(2)} \odot \sigma'(net^{(1)})\Big) \otimes \boldsymbol{x}^{(1)} \\
&\frac{\partial E}{\partial b^{(1)}} = \nabla E \cdot \Big(W^{(L-2)} \odot \sigma'(net^{(L-3)})\Big) \cdot \Big(W^{(L-3)} \odot \sigma'(net^{(L-4)})\Big) \cdot ... \cdot \Big(W^{(2)} \odot \sigma'(net^{(1)})\Big) \\
&\frac{\partial E}{\partial W^{(0)}} = \nabla E \cdot \Big(W^{(L-2)} \odot \sigma'(net^{(L-3)})\Big) \cdot \Big(W^{(L-3)} \odot \sigma'(net^{(L-4)})\Big) \cdot ... \cdot \Big(W^{(1)} \odot \sigma'(net^{(0)})\Big) \otimes \boldsymbol{x}^{(0)} \\
&\frac{\partial E}{\partial b^{(0)}} = \nabla E \cdot \Big(W^{(L-2)} \odot \sigma'(net^{(L-3)})\Big) \cdot \Big(W^{(L-3)} \odot \sigma'(net^{(L-4)})\Big) \cdot ... \cdot \Big(W^{(1)} \odot \sigma'(net^{(0)})\Big)
\end{aligned}
\tag{IV.1}
$$

With the gradient formulae, I can optimize the loss function with respect to model parameters. Program implementation is on its way.