Gradient Descent

Congshan Zhang

1 A Deep Neural Network

The training set has feature matrix with N individuals and p features:

$$X = \begin{bmatrix} X_{11} & \cdots & X_{1p} \\ \vdots & \ddots & \vdots \\ X_{N1} & \cdots & X_{Np} \end{bmatrix}. \tag{1}$$

Suppose that this matrix with $N = 100$ and $p = 9$ is the input of the following deep neural network.

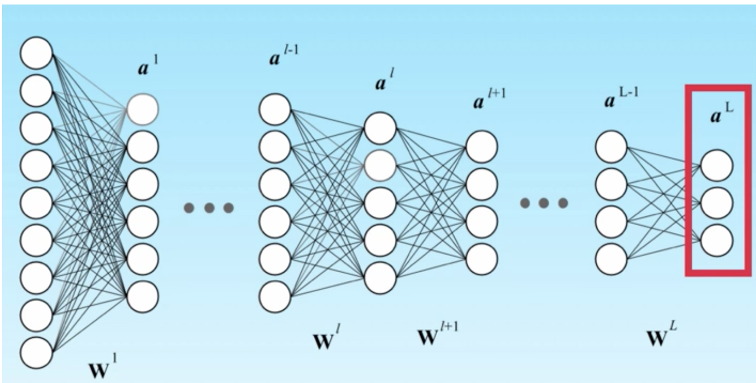

Figure 1: This is an illustration of a deep neural network with $p = 9$ input features.

Each layer l is associated with a weight matrix W^l, which is $(m \times k)$ with m the number of nodes on the $(l-1)$–th layer and k the number of nodes on the current layer. For example,

$$W^1 = \begin{bmatrix} W^1_{11} & \cdots & W^1_{16} \\ \vdots & \ddots & \vdots \\ W^1_{91} & \cdots & W^1_{96} \end{bmatrix}. \tag{2}$$

The feature matrix $X_{(100\times 9)}$ enters into this neural network and is first multiplied by $W^1_{(9\times 6)}$. Upon adding a bias $b^1_{(6\times 1)}$, we transform the original input into a signal which can be used by the first layer of neurons. In general, the input of layer l, which has k nodes, can be written as:

$$o^{l-1}_{N\times m} W^l_{(m\times k)} + \mathbf{1}_{(N\times 1)} \otimes (b^l_{(k\times 1)})^T \tag{3}$$

where $o^{l-1}_{N\times m}$ is the output of the previous layer $l-1$ which has m nodes. The component-wise activation function at layer l then transforms the input into an output:

$$o^l_{N\times k} = \sigma\left(o^{l-1}_{N\times m} W^l_{(m\times k)} + \mathbf{1}_{(N\times 1)} \otimes (b^l_{(k\times 1)})^T\right) \tag{4}$$

At the final layer L, the output is o^L. Usually, o^L represents the estimated choice probability for each class, and we need to choose the class with the largest estimated probability as our prediction.

2 Cross-entropy Loss

After we get our predicions, we want to construct a loss function, which is typically the so-called "cross-entropy loss" or negative log-likelihood.

As discussed above, the neural network estimates choice probabilities for each class $c \in \mathcal{J} \equiv \{1, \ldots, J\}$:

$$f(\boldsymbol{x})_c = p\left(y = c | \boldsymbol{x}, \boldsymbol{\theta}\right) \tag{5}$$

For each datapoint, the likelihood is given by

$$\prod_{c \in \mathcal{J}} p\left(y = c | \boldsymbol{x}, \boldsymbol{\theta}\right)^{1_{\{y=c\}}}. \tag{6}$$

So the negative log-likelihood for one particular data point is

$$l(y|\boldsymbol{x}, \boldsymbol{\theta}) = -\sum_{c \in \mathcal{J}} 1_{\{y=c\}} \log p\left(y = c | \boldsymbol{x}, \boldsymbol{\theta}\right). \tag{7}$$

Recall that o^L is $(N \times J)$, which gives our predictions of all N individuals. We need the likelihood function for the entire sample. It is not hard to observe that the negative log-likelihood for N individuals under *i.i.d.* assumption is given by

$$\mathcal{L}(y|\boldsymbol{x}, \boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c \in \mathcal{J}} 1_{\{y_i=c\}} \log p\left(y_i = c | \boldsymbol{x}_i, \boldsymbol{\theta}\right). \tag{8}$$

where the scaling factor $1/N$ is added on purpose. In particular, in the binary case where $y_i \in \{0, 1\}$, the cross-entropy loss for *each iteration* is

$$\mathcal{L}(y|\boldsymbol{x}, \boldsymbol{\theta}^{(k)}) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log \hat{y}_i^{(k)} + (1 - y_i) \log(1 - \hat{y}_i^{(k)}). \tag{9}$$

The last line is true because from numeric optimization point of view, for a given iteration, the set of parameters $\boldsymbol{\theta}^{(k)}$ is given, and the neural network will generate exactly $\hat{y}_i^{(k)} = p\left(y_i = 1 | \boldsymbol{x}_i, \boldsymbol{\theta}^{(k)}\right)$ and $1 - \hat{y}_i^{(k)} = p\left(y_i = 0 | \boldsymbol{x}_i, \boldsymbol{\theta}^{(k)}\right)$.

Why will the neural network generate exactly the choice probabilities? This comes from multinomial logistic model which we do not discuss in detail here. Recall the softmax activation function: $\sigma : \mathbb{R}^J \to [0, 1]^J$

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{J} e^{z_k}}, \text{ for } j = 1, \ldots, J. \tag{10}$$

In artificial neural network,

$$p(y_i = c | \boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{w}_c}}{\sum_{k=1}^{J} e^{\boldsymbol{x}_i^T \boldsymbol{w}_k}}, \text{ for } c = 1, \ldots, J. \tag{11}$$

Note that in the last layer, the weight matrix $W^L = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_J]$.

3 Backpropagation

Suppose at each layer, z is used to denote the input and o is used to denote the output, and let $\delta^l \equiv \frac{\partial \mathcal{L}}{\partial z^l}$. Then, at the last layer L, we have:

$$\underset{(J\times N)}{\delta^L} = \underset{(J\times N)}{\nabla_{o^L}\mathcal{L}} \odot \underset{(J\times N)}{\sigma'(z^L)} \tag{12}$$

Note that if the last layer has J nodes (which means we have J classes), then both $\Delta_{o^L}\mathcal{L}$ and $\sigma'(z^L)$ are $(J \times N)$ matrices. The $\odot$ is the component wise Hadamard product.

In an inner layer l, we have

$$\underset{(m\times N)}{\delta^l} = \underset{(m\times k)}{\left(W^{l+1}\right)} \underset{(k\times N)}{\delta^{l+1}} \odot \underset{(m\times N)}{\sigma'(z^l)} \tag{13}$$

where we assume there are m nodes in the l−th layer and k nodes in the $(l+1)$−th layer.

Finally, to update weights and biases, we calculate

$$\underset{(m\times q)}{\frac{\partial \mathcal{L}}{\partial W^l}} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial z_i^l} \left(\frac{\partial z_i^l}{\partial W^l}\right)^T = \sum_{i=1}^{N} \delta^l_{(\cdot,i)} o^l_{(i,\cdot)} = \underset{(m\times N)}{\delta^l} \underset{(N\times q)}{o^{l-1}} \tag{14}$$

$$\underset{(m\times 1)}{\frac{\partial \mathcal{L}}{\partial b^l}} = \sum_{i=1}^{N} \left(\frac{\partial z_i^l}{\partial b^l}\right) \frac{\partial \mathcal{L}}{\partial z_i^l} = \sum_{i=1}^{N} I_{m\times m}\delta^l_{(\cdot,i)} = \underset{(m\times N)}{\delta^l} \mathbf{1}_{N\times 1} \tag{15}$$

where q is the number of nodes in the $(l-1)$−th layer. This calculation follows from *the chain rule for tensors* (see (6.47) on page 207 in [1]).

3.1 Algorithm

step 1) Forward propagation: get all o^l and $\hat{y}_i$.
step 2) Error in the last layer: $\underset{(J\times N)}{\delta^L} = \underset{(J\times N)}{\nabla_{o^L}\mathcal{L}} \odot \underset{(J\times N)}{\sigma'(z^L)}$.
step 3) Backward propagation: calculate $\underset{(m\times N)}{\delta^l} = \underset{(m\times k)}{(W^{l+1})} \underset{(k\times N)}{\delta^{l+1}} \odot \underset{(m\times N)}{\sigma'(z^l)}$
step 4) Derivatives of cost with respect to weight and bias: $\underset{(m\times q)}{\frac{\partial \mathcal{L}}{\partial W^l}} = \underset{(m\times N)}{\delta^l} \underset{(N\times q)}{o^{l-1}}$, and $\underset{(m\times 1)}{\frac{\partial \mathcal{L}}{\partial b^l}} = \underset{(m\times N)}{\delta^l} \mathbf{1}_{N\times 1}$.
step 5) Update weight and bias: $w \to w - \alpha \left(\frac{\partial \mathcal{L}}{\partial W^l}\right)^T$, and $b \to b - \alpha \left(\frac{\partial \mathcal{L}}{\partial b^l}\right)^T$.

References

[1] Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT press.