Neural Network with Hidden Layers

(Dated: July 23, 2024)

A simple neural network with one hidden layer is illustrated in fig. 1, where in the zeroth layer, i.e. input layer we have the input data

$$A^{(0)} = Z^{(0)}, \tag{1}$$

which is the output of the zeroth layer and is directly fed to the first hidden layer. The signals received over all vertices is

$$Z^{(1)} = W^{(1)} A^{(0)} + b^{(1)} \tag{2}$$

In component form,

$$Z_i^{(1),m} = \sum_j W_{ij}^{(1)} A_j^{(0),m} + b_i^{(1)} \tag{3}$$

where the index $m = 1, 2, \ldots, M$ denotes the sample number, and $i = 1, 2, \ldots, N_1$ indicates the corresponding components of column vector representing the layer number (1), and $j = 1, 2, \ldots, N_0$ indicates the components of column vector representing the layer (0). Therefore, $Z^{(1)}$ is $[N_1 \times M]$, $W^{(1)}$ is $[N_1 \times N_0]$, $A^{(0)}$ is $[N_0 \times M]$. $b^{(1)}$ is in principle a column vector of $[N_1 \times 1]$ but to properly assign values during program execution it needs to be replicated into a $[N_1 \times M]$ matrix to match matrix dimensions. Note that in this way, all the training/testing samples are included in the matrix, so that each matrix operations such as the forward/backward propagation will run for all training/testing sets, and in this vectorized programming parallel computation in high level programming language such as MATLAB or some python packages can be utilized automatically.

With some activation functions, which we choose to be a rectified linear unit, $g(z) = \mathrm{ReLU}(z) = \max(0, z)$, an output value of layer (1) is obtained by

$$A^{(1)} = g(Z^{(1)}), \quad \text{and} \quad A_i^{(1)} = g(Z_i^{(1)}, i) \equiv g(Z_i^{(1)}) \tag{4}$$

Similarly, we can write the forward propagation for other layers

$$Z^{(2)} = W^{(2)} A^{(1)} + b^{(2)}, \quad \text{and} \quad A_i^{(2)} = \sigma(Z^{(2)}, i) \equiv \sigma(Z_i^{(2)}) \tag{5}$$

where in the last (output) layer we used the SoftMax activation function, and simplified the notation in the above format.

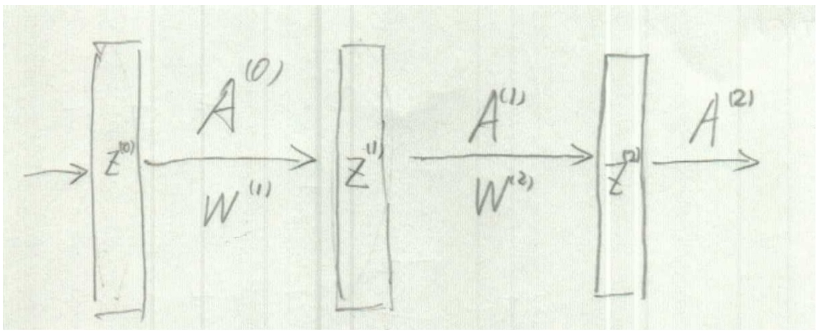

FIG. 1: Diagram illustrating the structure of the neural network.

I. FORWARD PROPAGATION

To summarize, the forward propagation reads:

$$A^{(0)} = Z^{(0)}, \quad \text{size } [N_0 \times M] \tag{6}$$

$$Z^{(1)} = W^{(1)}A^{(0)} + b^{(1)} \tag{7}$$

$$A^{(1)} = g(Z^{(1)}), \quad \text{size } [N_1 \times M] \tag{8}$$

$$Z^{(2)} = W^{(2)}A^{(1)} + b^{(2)} \tag{9}$$

$$A^{(2)} = \sigma(Z^{(2)}), \quad \text{size } [N_2 \times M] \tag{10}$$

For clarification, $Z^{(0)}$ is also the input value (feature value), and $A^{(2)} = Y$ is the output value (target value).
To achieve a well-trained network, we seek to minimize the error (loss function)

$$L = \frac{1}{2}\|A^{(2)} - Y\|^2 = \frac{1}{2M}\sum_m |A^{(2),m} - Y^m|^2, \tag{11}$$

where Y is the actual target value in some training set, and an average over all the training samples is assumed when summation over m is omitted.

II. GRADIENT CALCULATION, BACK PROPAGATION

To achieve $\min L$, we take the gradient of L and move towards its opposite direction in parameter space, because of the chain rule and layered structure of the network, this is no other than the backpropagation.
We eventually want $\frac{\partial L}{\partial W^{(i)}}$, $\frac{\partial L}{\partial b^{(i)}}$, but it is more convenient to work from the derivatives

$$\frac{\partial L}{\partial Z^{(2)}_\alpha} = \sum_\beta (A^{(2)}_\beta - Y_\beta)\frac{\partial A^{(2)}_\beta}{\partial Z^{(2)}_\alpha}. \tag{12}$$

By virtues of the derivative of the SoftMax function:

$$\frac{\partial A^{(2)}_\beta}{\partial Z^{(2)}_\alpha} = A^{(2)}_\beta(\delta_{\alpha\beta} - A^{(2)}_\alpha). \tag{13}$$

We get:

$$\begin{aligned}\frac{\partial L}{\partial Z^{(2)}_\alpha} &= \sum_\beta (A^{(2)}_\beta - Y_\beta)A^{(2)}_\beta(\delta_{\alpha\beta} - A^{(2)}_\alpha) \\ &= (A^{(2)}_\alpha - Y_\alpha)A^{(2)}_\alpha - A^{(2)}_\alpha \sum_\beta A^{(2)}_\beta(A^{(2)}_\beta - Y_\beta) \\ &= A^{(2)}_\alpha(A^{(2)}_\alpha - Y_\alpha) - A^{(2)}_\alpha \sum_\beta A^{(2)}_\beta(A^{(2)}_\beta - Y_\beta). \end{aligned} \tag{14}$$

where the averaged error term $E = \sum_\beta A^{(2)}_\beta(A^{(2)}_\beta - Y_\beta)$ effectively normalizes the error by spreading it across the output probability distribution $A^{(2)}$, and $E \to 0$ as the network improves during training. Therefore, we neglect the second term and take

$$\frac{\partial L}{\partial Z^{(2)}_\alpha} \approx A^{(2)}_\alpha(A^{(2)}_\alpha - Y_\alpha). \tag{15}$$

Furthermore, as $\sum_\alpha A^{(2)}_\alpha = 1$, the $A^{(2)}$ can be treated as a coefficient in adjusting the gradient weights, for computational efficiency the difference $A^{(2)}_\alpha - Y_\alpha$ is sufficient to guide weight updates. Therefore, we consider the modified gradient terms

$$\frac{\partial L}{\partial Z^{(2)}_\alpha} \approx A^{(2)}_\alpha - Y_\alpha, \tag{16}$$

which still captures the necessary direction for minimizing the loss function.

The relevant derivatives for gradient descent are:

$$\begin{aligned}
\frac{\partial L}{\partial W_{ij}^{(2)}} &= \sum_\alpha \frac{\partial L}{\partial Z_\alpha^{(2)}} \frac{\partial Z_\alpha^{(2)}}{\partial W_{ij}^{(2)}} \approx \sum_\alpha (A_\alpha^{(2)} - Y_\alpha) \frac{\partial(\sum_\beta W_{\alpha\beta}^{(2)} A_\beta^{(1)} + b_\alpha^{(2)})}{\partial W_{ij}^{(2)}} \\
&= \sum_\alpha (A_\alpha^{(2)} - Y_\alpha) \delta_{\alpha i} A_j^{(1)} \\
&= (A_i^{(2)} - Y_i) A_j^{(1)} \\
&= \frac{1}{M} \sum_m (A_i^{(2),m} - Y_i^m) A_j^{(1),m} \\
\frac{\partial L}{\partial W^{(2)}} &= \frac{1}{M} (A^{(2)} - Y) \cdot A^{(1)T},
\end{aligned} \tag{17}$$

where in the last step we wrote the explicit matrix representation. Similarly,

$$\begin{aligned}
\frac{\partial L}{\partial b_j^{(2)}} &= \sum_\alpha \frac{\partial L}{\partial Z_\alpha^{(2)}} \frac{\partial Z_\alpha^{(2)}}{\partial b_j^{(2)}} \\
&\approx \sum_\alpha (A_\alpha^{(2)} - Y_\alpha) \cdot \delta_{j\alpha} \\
&= A_j^{(2)} - Y_j
\end{aligned} \tag{18}$$

$$\frac{\partial L}{\partial b^{(2)}} = \frac{1}{M} \text{ sum } ((A^{(2)} - Y), \quad \text{axis = column }). \tag{19}$$

To match the matrix dimensions, we replicate the above vectors using python commands

$$\frac{\partial L}{\partial b^{(2)}} \approx \text{np.tile}(\frac{1}{M}\text{np.sum}(A^{(2)} - Y, \text{axis} = 1), (M, 1)).T \tag{20}$$

Backpropagate to the first layer:

$$\begin{aligned}
\frac{\partial L}{\partial W_{ij}^{(1)}} &= \sum_{\alpha,\beta} \frac{\partial L}{\partial Z_\alpha^{(2)}} \frac{\partial Z_\alpha^{(2)}}{\partial A_\beta^{(1)}} \frac{\partial A_\beta^{(1)}}{\partial W_{ij}^{(1)}} \\
&= \sum_{\alpha,\beta} (A_\alpha^{(2)} - Y_\alpha) W_{\alpha\beta}^{(2)} g'(Z_\beta^{(1)}) \delta_{i\beta} A_j^{(0)} \\
&= \sum_\alpha (A_\alpha^{(2)} - Y_\alpha) W_{\alpha i}^{(2)} g'(Z_i^{(1)}) A_j^{(0)}
\end{aligned} \tag{21}$$

In component form and matrix representation:

$$\begin{aligned}
\frac{\partial L}{\partial W_{ij}^{(1)}} &= \frac{1}{M} \sum_{m,\alpha} W_{\alpha i}^{(2)} (A_\alpha^{(2),m} - Y_\alpha^m) g'(Z_i^{(1),m}) A_j^{(0),m} \\
\frac{\partial L}{\partial W^{(1)}} &= \left[W^{(2)T} \cdot (A^{(2)} - Y) * g'(Z^{(1)}) \right] \cdot A^{(0)T}, \\
\text{sizes } [N_1 \times N_0], &\quad [N_2 \times M], [N_1 \times M], [M \times N_0].
\end{aligned} \tag{22}$$

Here $*$ is the element-wise product.

For $\frac{\partial L}{\partial b_j^{(1)}}$:

$$\begin{aligned}
\frac{\partial L}{\partial b_j^{(1)}} &= \sum_\alpha \frac{\partial L}{\partial Z_\alpha^{(2)}} \frac{\partial Z_\alpha^{(2)}}{\partial b_j^{(1)}} \\
&= \sum_{\alpha\beta} (A_\alpha^{(2)} - Y_\alpha) W_{\alpha\beta}^{(2)} \delta_{j\beta} g'(Z_\beta^{(1)}) \\
&= \sum_\alpha (A_\alpha^{(2)} - Y_\alpha) W_{\alpha j}^{(2)} g'(Z_j^{(1)}) \\
\frac{\partial L}{\partial b^{(1)}} &= \frac{1}{M} \text{sum}(\left[W^{(2)T} \cdot (A^{(2)} - Y) * g'(Z^{(1)}) \right], \quad \text{axis} = 1 \ ). \qquad (23)
\end{aligned}$$

For dimension consistency, let

$$\text{pd\_Z1} = W^{(2)T} \cdot (A^{(2)} - Y) * g'(Z^{(1)}) \tag{24}$$

Then,

$$\frac{\partial L}{\partial b^{(1)}} = \text{np.tile} \left(\frac{1}{M} \text{sum(pd\_Z1, axis=1), (M, 1))} \right) .T \tag{25}$$

III. ALGORITHM OUTLINER

With these gradient terms in place, we may update the parameters at each step:

1. Initialization:

$$W^{(1)} \sim \text{rand}(N_1, N_0) - \frac{1}{2}, \quad b^{(1)} \sim \text{rand}(N_1, 1) - \frac{1}{2} \tag{26}$$

$$W^{(2)} \sim \text{rand}(N_2, N_1) - \frac{1}{2}, \quad b^{(2)} \sim \text{rand}(N_2, 1) - \frac{1}{2} \tag{27}$$

2. For a given training set X, compute the input and output of all layers, $Z^{(1)}, A^{(1)}, Z^{(2)}, A^{(2)}$ using the so-called forward propagation.

3. Define a function called backward propagation to compute $\partial_{W^{(i)}} L, \partial_{b^{(i)}} L$.

4. Update the parameters using gradient descent:

$$W^{(i)} = W^{(i)} - \alpha * \partial_{W^{(i)}} L \tag{28}$$

$$b^{(i)} = b^{(i)} - \alpha * \partial_{b^{(i)}} L \tag{29}$$

   according to some learning rate α.

5. Repeat steps 2-4 until a certain accuracy is achieved.