AMA 564 Lecture 10: Appendix of algorithm summary

Jianhui Huang
Department of Applied Mathematics
Hong Kong Polytechnic University

AMA 564, March 24, 2022

Backpropagation for two-layer neural networks

- Above algorithm implies some **forward propagation** by Step 1: $\boldsymbol{x} \longrightarrow z \longrightarrow a \longrightarrow o$.
- Here: $\boldsymbol{x}$: basic input in bottom layer, z: inactivation intermediate variable, a: activation intermediate variable, o: output in last layer.
- In addition, above algorithm also implies some **backward propagation** by Step 2: $\delta^{[2]} \longrightarrow \delta^{[1]}$ from top to bottom layer as well as
- Step 3: $\frac{\partial J}{\partial W^{[2]}}, \frac{\partial J}{\partial b^{[2]}}$ (gradient in top layer) $\longrightarrow \frac{\partial J}{\partial W^{[1]}}, \frac{\partial J}{\partial b^{[1]}}$ (gradient in bottom layer)

Summary of backpropagation for multi-layer network

- Recall multi-layer network for individual training sample $\boldsymbol{x}^{(i)}$

$$
\begin{aligned}
a^{[0]} &= \boldsymbol{x}^{(i)} \\
a^{[1]} &= \mathrm{ReLU}\left(\boldsymbol{W}^{[1]} a^{[0]} + b^{[1]}\right) \\
a^{[2]} &= \mathrm{ReLU}\left(\boldsymbol{W}^{[2]} a^{[1]} + b^{[2]}\right) \\
&\dots \\
a^{[k]} &= \mathrm{ReLU}\underbrace{\left(\boldsymbol{W}^{[k]} a^{[k-1]} + b^{[k]}\right)}_{z^{[k]}} = \mathrm{ReLU}\left(z^{[k]}\right) \\
&\dots \\
a^{[r-1]} &= \mathrm{ReLU}\left(\boldsymbol{W}^{[r-1]} a^{[r-2]} + b^{[r-1]}\right) \\
a^{[r]} &= z^{[r]} = \boldsymbol{W}^{[r]} a^{[r-1]} + b^{[r]}
\end{aligned}
$$

Summary of backpropagation procedure

- To train such multi-layer network, we need learn or estimate the following weight and bias matrix sequences across r layers:

$$\text{weight matrix sequence } \mathcal{W}: \quad \boldsymbol{W}^{[1]}, \boldsymbol{W}^{[2]}, \cdots, \boldsymbol{W}^{[r-1]}, \boldsymbol{W}^{[r]}$$

$$\text{bias matrix sequence } \mathcal{B}: \quad b^{[1]}, b^{[2]}, \cdots, b^{[r-1]}, b^{[r]}$$

- Total weight (parameter) across all layers: $\boldsymbol{\theta} = \{\mathcal{W}, \mathcal{B}\}$.
- To this end, we need first compute the input sequence:

$$\textbf{non-activation} \text{ sequence } \mathcal{Z}: \quad z^{[1]}, z^{[2]}, \cdots, z^{[r-1]}, z^{[r]}$$

$$\textbf{activation} \text{ sequence } \mathcal{A}: \quad \color{red}{a^{[0]}}, a^{[1]}, a^{[2]}, \cdots, a^{[r-1]}, a^{[r]}$$

Summary of backpropagation procedure

- They should be computed using **forward propagation** "$\Uparrow$" from bottom to top layer:

$$\mathcal{Z} \Downarrow: z^{[1]} \longrightarrow z^{[2]} \longrightarrow \dots z^{[r-1]} \longrightarrow z^{[r]},$$

$$\mathcal{A} \Uparrow: a^{[0]} \longrightarrow a^{[1]} \longrightarrow a^{[2]} \longrightarrow \dots a^{[r-1]} \longrightarrow a^{[r]}.$$

- Once done, we may compute the gradients in all layers like $\delta_k = \frac{\partial J}{\partial z^{[k]}}$ and $\frac{\partial J}{\partial b^{[k]}}$ for $k = r, r-1, \cdots, 2, 1$ using **back-propagation** "$\Downarrow$".

- Given gradients in all layers, we may apply the **(stochastic) gradient method** to update with loop manner.

Summary of backpropagation procedure

- The backpropagation procedures provides us an effective way to compute the gradients of cost functional in deep network.
- We can think of backpropagation as a way of computing the gradient of cost functional by systematically applying the chain rule for multi-variable function.

Summary of backpropagation procedure

- Let us explicitly write above procedure in the form of algorithm:
- **Step 1**: input data $\boldsymbol{x}^{(i)}$ and set $a^{[0]} = \boldsymbol{x}^{(i)}$.
- **Step 2**: **forward propagation** pass: compute **inactivation** sequence $z^{[1]}, z^{[2]}, \cdots, z^{[r-1]}, z^{[r]}$ and **activation** sequence $a^{[1]}, a^{[2]}, \cdots, a^{[r-1]}, a^{[r]}$.
- Actually, we have the sequence mixture: $\boldsymbol{x}^{(i)} = a^{[0]} \longrightarrow z^{[1]} \longrightarrow a^{[1]} \longrightarrow z^{[2]} \longrightarrow a^{[2]} \longrightarrow \cdots \longrightarrow z^{[r-1]} \longrightarrow a^{[r-1]} \longrightarrow z^{[r]} = a^{[r]}$.
- **Step 3**: output layer gradient: $\delta^{[r]} \triangleq \frac{\partial J}{\partial z^{[r]}} = \left(z^{[r]} - y\right)$.

- **Step 4**: **backward propagation** pass: compute gradient sequence $\delta^{[r-1]} \longrightarrow \delta^{[r-2]} \longrightarrow \cdots \longrightarrow \delta^{[2]} \longrightarrow \delta^{[1]}$
- **Step 5**: For $k = r, r-1, \cdots, 2, 1$, compute

$$\frac{\partial J}{\partial \boldsymbol{W}^{[k]}} = \frac{\partial J}{\partial z^{[k]}} \cdot a^{[k-1]^\top} = \delta^{[k]} \cdot a^{[k-1]^\top},$$

$$\frac{\partial J}{\partial b^{[k]}} = \delta^{[k]}$$

- **Step 6**: gradient descent updating. Then, we may apply the **(stochastic) gradient method** to update estimate of $\boldsymbol{W}^{[k]}, b^{[k]}$.
- **Loop** for above **Step 2-6** to get $\boldsymbol{\theta} = \{\mathcal{W}, \mathcal{B}\}$ estimation convergence hence complete network training.