Lecture 3: Deep Learning

Pedro Savarese

TTI

2018

Table of Contents

Sigmoid Feedforward Neural Networks

Multiple hidden layers:
Input layer:

$$\mathbf{h}^{(0)} = \mathbf{x}$$

Hidden layers, for $1 \leq k \leq L-1$:

$$\mathbf{z}^{(k)} = \mathbf{W}^{(k)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}$$

$$\mathbf{h}^{(k)} = \sigma(\mathbf{z}^{(k)})$$

Output layer:

$$p(y = 1|\mathbf{x}) = h^{(L)} = \sigma\left(\langle \mathbf{w}^{(L)}, \mathbf{h}^{(L-1)} \rangle + b^{(L)}\right)$$

Sigmoid Feedforward Neural Networks

- So far, we have learned how to separate two classes: **binary classification**
- But what if $y \in \{1, 2, \dots, C\}$?
- Can learn C scores $s_1(\mathbf{x}), s_2(\mathbf{x}), \dots, s_C(\mathbf{x})$ for each class
- Then, map to $p(y = 1|\mathbf{x}), p(y = 2|\mathbf{x}), \dots, p(y = C|\mathbf{x})$
- **Softmax function**: $p(y = i|\mathbf{x}) = \frac{e^{s_i(\mathbf{x})}}{\sum_{j=1}^{C} e^{s_j(\mathbf{x})}}$
- Assures $p(y = i|\mathbf{x})$ is positive (exponentiation) and $\sum_j p(y = j|\mathbf{x}) = 1$ (normalization)

$$\mathbf{p}(y|\mathbf{x}) = \mathbf{h}^{(L)} = softmax\left(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}\right)$$

Sigmoid Feedforward Neural Networks

- Training: gradient ascent/descent:

$$\begin{aligned}\frac{\partial \log \mathcal{L}}{\partial \mathbf{W}^{(1)}} &= \frac{\partial \log \mathcal{L}}{\partial \mathbf{h}^{(L)}} \frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{h}^{(L-1)}} \frac{\partial \mathbf{h}^{(L-1)}}{\partial \mathbf{h}^{(L-2)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(1)}} \\ &= \frac{\partial \log \mathcal{L}}{\partial \mathbf{h}^{(L)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(1)}} \prod_{k=2}^{L} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{h}^{(k-1)}}\end{aligned}$$

But what is $\frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{h}^{(k-1)}}$? Remember that:

$$\mathbf{h}^{(k)} = \sigma(\mathbf{z}^{(k)})$$

So $\frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{h}^{(k-1)}} = \sigma'(\mathbf{z}^{(k)}) \frac{\partial \mathbf{z}^{(k)})}{\partial \mathbf{h}^{(k-1)}}$

Sigmoid Feedforward Neural Networks

- Training: gradient ascent/descent:

$$\frac{\partial \log \mathcal{L}}{\partial \mathbf{W}^{(1)}} = \frac{\partial \log \mathcal{L}}{\partial \mathbf{h}^{(L)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(1)}} \prod_{k=2}^{L} \sigma'(\mathbf{z}^{(k)}) \frac{\partial \mathbf{z}^{(k)})}{\partial \mathbf{h}^{(k-1)}}$$

But $\sigma'(z) \approx 0$ for $z > 5$ or $z < -5$. Having a single $\mathbf{z}^{(k)}$ to be too high/ too low results in $\frac{\partial \log \mathcal{L}}{\partial \mathbf{W}^{(1)}} \approx 0$ and training does not work

Table of Contents

ReLU Feedforward Neural Networks

- Sigmoid function is problematic for gradient ascent/descent
- Alternative: use different function for hidden layers – they do not need to be probabilities anyway
- Commonly: use $ReLU(z) = \max(0, z)$ instead
- $ReLU'(z) = 1\{z > 0\}$

$$\mathbf{h}^{(k)} = ReLU(\mathbf{z}^{(k)}) = \max(0, \mathbf{z}^{(k)})$$

ReLU Feedforward Neural Networks

Extra quick coding session:

- Train network with 3 hiddens layers and sigmoid activation
- Train network with 3 hiddens layers and ReLU activation