Lecture 3: Deep Learning

Pedro Savarese

TTI

2018
Table of Contents

1. Sigmoid Feedforward Neural Networks

2. ReLU Feedforward Neural Network
Multiple hidden layers:
Input layer:
\[ h^{(0)} = x \]

Hidden layers, for \( 1 \leq k \leq L - 1 \):
\[
\begin{align*}
\mathbf{z}^{(k)} &= \mathbf{W}^{(k)} h^{(k-1)} + \mathbf{b}^{(k)} \\
\mathbf{h}^{(k)} &= \sigma(\mathbf{z}^{(k)})
\end{align*}
\]

Output layer:
\[
p(y = 1|\mathbf{x}) = h^{(L)} = \sigma(\langle \mathbf{w}^{(L)}, \mathbf{h}^{(L-1)} \rangle + b^{(L)})
\]
So far, we have learned how to separate two classes: **binary classification**

But what if $y \in \{1, 2, \ldots, C\}$?

Can learn $C$ scores $s_1(x), s_2(x), \ldots, s_C(x)$ for each class

Then, map to $p(y = 1|x), p(y = 2|x), \ldots, p(y = C|x)$

**Softmax function**: $p(y = i|x) = \frac{e^{s_i(x)}}{\sum_{j=1}^{C} e^{s_j(x)}}$

Assures $p(y = i|x)$ is positive (exponentiation) and $\sum_j p(y = j|x) = 1$ (normalization)

$$p(y|x) = h^{(L)} = \text{softmax}(W^{(L)}h^{(L-1)} + b^{(L)})$$
Sigmoid Feedforward Neural Networks

Training: gradient ascent/descent:

\[
\frac{\partial \log \mathcal{L}}{\partial W^{(1)}} = \frac{\partial \log \mathcal{L}}{\partial h^{(L)}} \frac{\partial h^{(L)}}{\partial h^{(L-1)}} \frac{\partial h^{(L-1)}}{\partial h^{(L-2)}} \cdots \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial W^{(1)}}
\]

\[
= \frac{\partial \log \mathcal{L}}{\partial h^{(L)}} \frac{\partial h^{(1)}}{\partial W^{(1)}} \prod_{k=2}^{L} \frac{\partial h^{(k)}}{\partial h^{(k-1)}}
\]

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

\[
h^{(k)} = \sigma(z^{(k)})
\]

So \( \frac{\partial h^{(k)}}{\partial h^{(k-1)}} = \sigma'(z^{(k)}) \frac{\partial z^{(k)}}{\partial h^{(k-1)}} \)
Sigmoid Feedforward Neural Networks

Training: gradient ascent/descent:

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

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

1. Sigmoid Feedforward Neural Networks

2. ReLU Feedforward Neural Network
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
$ReLU'(z) = 1\{z > 0\}$

$$h^{(k)} = ReLU(z^{(k)}) = \max(0, z^{(k)})$$
Extra quick coding session:

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