Deep Learning

Theoretical Exercises – Week 10 – Chapter 8

Exercises on the book ”Deep Learning” written by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Exercises and solutions by T. Méndez and G. Schuster

FS 2024

1 Exercises on Optimization for Training Deep Models

1. What is the differences between learning and pure optimization?

   Solution:
   In pure optimization, the goal itself is to optimize an objective function $J(\theta)$. However, in machine learning, a performance measurement P has to be optimized, which is defined with respect to the test set. That performance measurement P is only optimized indirectly by minimizing a different cost function $J(\theta)$, which is defined with respect to the training set. We hope that by minimizing the cost function $J(\theta)$, the performance measurement P will also be improved. The reason why it is done this way is because the true distribution of the data p_{data} is not known and, thus, has to be approximated with a training set.

2. Why is gradient descent better suited for training neural networks than Newton's method?

   Solution:
   The parameter space of neural networks is generally a very high dimensional space, in which the expected number of saddle points is much larger than the expected number of local minima. Since Newton's method is designed to solve for critical points (points where the gradient is zero) and there are many more saddle points than local minima (or maxima), Newton's method is attracted to saddle points. Gradient descent, however, is designed to move ”downhill' and does not explicitly seek a critical point. Also, gradient descent empirically seems to be able to escape saddle points in many cases. For these reasons gradient descent is better suited for training neural networks than Newton's method.

3. Assume you have a function

$$g(\boldsymbol{x}) = g(x_1, x_2) = \begin{cases} 1, & x_1^2 + x_2^2 < 1 \text{ and } x_1, x_2 \geq 0 \\ 0, & \text{otherwise} \end{cases},$$

which is defined on the two-dimensional random variable $\boldsymbol{x} = [x_1,\ x_2]^T$ with the following probability density function

$$p(\boldsymbol{x}) = p(x_1, x_2) = \begin{cases} 1, & 0 \le x_1, x_2 < 1 \\ 0, & \text{otherwise} \end{cases}.$$

Now, from this function $g(\boldsymbol{x})$ the expected value $\mathbb{E}_\mathrm{x}[g(\boldsymbol{x})]$ has to be calculated using two different methods:

(a) With the definition of the expected value, thus, by calculating the following integral

$$\mathbb{E}_\mathrm{x}[g(\boldsymbol{x})] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x_1, x_2) \cdot g(x_1, x_2)\, dx_1\, dx_2$$

(b) With the Monte Carlo approach, by randomly taking samples $\boldsymbol{x}_i$ according to the probability density function $p(\boldsymbol{x})$ and determining their function value $g(\boldsymbol{x}_i)$. The expected value can then be estimated as

$$\hat{\mathbb{E}}_\mathrm{x}[g(\boldsymbol{x})] = \frac{1}{M}\sum_{i=1}^{M} g(\boldsymbol{x}_i),$$

where M is the number of samples. The estimated expected value $\hat{\mathbb{E}}_\mathrm{x}[g(\boldsymbol{x})]$ converges to the true expected value $\mathbb{E}_\mathrm{x}[g(\boldsymbol{x})]$ from exercise (a), as the number of samples M goes to infinity.

Solution:

(a) The integral of exercise (a) can be calculated as

$$\begin{aligned}
\mathbb{E}_\mathrm{x}[g(\boldsymbol{x})] &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x_1, x_2) \cdot g(x_1, x_2)\, dx_1\, dx_2 \\
&= \int_{0}^{1}\int_{0}^{1} 1 \cdot g(x_1, x_2)\, dx_1\, dx_2 \\
&= \int_{0}^{1}\int_{0}^{\sqrt{1-x_2^2}} 1\, dx_1\, dx_2 \\
&= \int_{0}^{1} \sqrt{1 - x_2^2}\, dx_2 \\
&= \left[\frac{\arcsin(x_2)}{2} + \frac{x_2 \cdot \sqrt{1 - x_2^2}}{2}\right]_0^1 \\
&= \frac{\pi}{4} + 0 - 0 - 0 \\
&= \frac{\pi}{4}.
\end{aligned}$$

Since $\boldsymbol{x}$ is equally distributed and $g(\boldsymbol{x})$ assigns the value 1 to all arguments within the unit circle in the first quadrant and the value 0 to all others, the expected value results, as was to be expected, in the area of a fourth of a circle.

(b) The following code snippet implements the Monte Carlo approach. The error between the estimated expected value and the true expected value gets smaller as the number of samples increases.

```python
import numpy as np

M = 10000
count = 0
for i in range(M):
    x1 = np.random.uniform(low=0.0, high=1.0)
    x2 = np.random.uniform(low=0.0, high=1.0)
    if (x1**2 + x2**2) < 1:
        count += 1

estimate = count / M

true_value = np.pi / 4
error = true_value - estimate
relative_error = np.abs(error) / true_value

print("E[g(x)] =", estimate)
print("Relative error =", 100 * relative_error, "%")
```