Wahrscheinlichkeitstheorie und Statistik

Serie 12 - Lösungen

MC 12-1 (Neuronale Netze). Unter welchen Operationen ist die Menge der neuronalen Netze mit fixer Aktivierungsfunktion abgeschlossen? (Mehrere korrekte Antworten sind möglich.)

(a) Linearkombinationen neuronaler Netze sind neuronale Netze.

(b) Das Produkt neuronaler Netze ist ein neuronales Netz.

(c) Die Komposition neuronaler Netze ist ein neuronales Netz.

(d) Das Maximum neuronaler Netze ist ein neuronales Netz.

Lösung:

(a) Wahr. Man verwendet einen linearen Layer, der die Outputs der gegebenen neuronalen Netze kombiniert.

(b) Falsch. Gegenbeispiel: Neuronale Netze mit ReLU Aktivierungsfunktion sind stückweise linear, Produkte stückweise linearer Funktionen sind jedoch stückweise quadratisch.

(c) Wahr. Man stapelt die Layer der beiden Netze, wobei der lineare Output-Layer des einen Netzes mit dem linearen Input-Layer des anderen Netzes durch Matrix-Multiplikation kombiniert wird.

(d) Falsch. Gegenbeispiel: Neuronale Netze mit glatten Aktivierungsfunktionen sind glatte Funktionen, das Maximum glatter Funktionen ist aber i.A. nicht glatt.

Aufgabe 12-2 (Universelle Approximation durch ReLU Netze).

(a) Stellen Sie folgende Dreiecksfunktion als neuronales Netzwerk mit ReLU Aktivierungsfunktion dar:

$$h : \mathbb{R} \to \mathbb{R}, \qquad h(x) = \begin{cases} x+1, & x \in [-1,0], \\ 1-x, & x \in [0,1], \\ 0, & \text{sonst.} \end{cases}$$

(b) Zeigen Sie mit Hilfe von (a), dass jede stetige Funktion $f : [0,1] \to \mathbb{R}$ gleichmässig durch neuronale Netze mit ReLU Aktivierungsfunktion approximiert werden kann.

Hinweis: Für jedes $n \in \mathbb{N}$ sei die Faber–Schauder Approximation f_n definiert als die stückweise lineare Funktion, die an Vielfachen von 2^{-n} mit f übereinstimmt. Da f gleichmässig stetig ist, konvergieren f_n gleichmässig gegen f.

Lösung:

(a) $h(x) = \rho(x+1) - 2\rho(x) + \rho(x-1)$ ist ein neuronales Netz mit ReLU Aktivierungsfunktion.

(b) $f_n(x) = \sum_{k=0}^{2^n} f(k2^{-n})\, h(2^n(x - k2^{-n}))$ ist ein neuronales Netz mit ReLU Aktivierungsfunktion.

Definition (Empirische Risikominimierung). Seien $X_1, \dots, X_n$ unabhängige, identisch verteilte Zufallsvariablen. Empirische Risikominimierung ist ein Optimierungsproblem der Form

$$\inf_\theta \frac{1}{n} \sum_{i=1}^n L(\theta, X_i),$$

für eine Verlustfunktion L.

Aufgabe 12-3 (Maximum-Likelihood Schätzung als empirische Risikominimierung). Beschreiben Sie in folgenden statistischen Modellen Maximum-Likelihood Schätzung als empirische Risikominimierung und identifizieren Sie die Verlustfunktion L:

(a) Der unbekannte Parameter ist eine messbare Funktion $f : \mathbb{R} \to \mathbb{R}$ und die Beobachtungen sind unabhängige identisch verteilte Zufallsvariablen $(X_1, Y_1), \dots, (X_n, Y_n)$, wobei Y_i gegeben X_i normalverteilt mit Mittelwert $f(X_i)$ und Standardabweichung 1 ist.

(b) Der unbekannte Parameter ist eine messbare Funktion $p : \mathbb{R} \to \mathbb{R}$ und die Beobachtungen sind unabhängige identisch verteilte Zufallsvariablen $(X_1, Y_1), \dots, (X_n, Y_n)$, wobei Y_i gegeben X_i Bernoulliverteilt ist mit Erfolgswahrscheinlichkeit $p(X_i)$.

Lösung:

1. Der Maximum-Likelihood Schätzer minimiert die negative log-Likelihood:

$$\inf_f \frac{1}{n} \sum_{i=1}^n \frac{\big(Y_i - f(X_i)\big)^2}{2}$$

Die Verlustfunktion ist quadratisch in y und $f(x)$:

$$L\big(f, (x, y)\big) = \frac{\big(y - f(x)\big)^2}{2}.$$

2. Der Maximum-Likelihood Schätzer minimiert die negative log-Likelihood:

$$\inf_p \; -\sum_{i=1}^n \Big(Y_i \log\big(p(X_i)\big) + (1 - Y_i) \log\big(1 - p(X_i)\big) \Big).$$

Die Verlustfunktion ist die Kreuzentropie von y und $p(x)$:

$$L\big(p, (x, y)\big) = -y \log\big(p(x)\big) - (1 - y) \log\big(1 - p(x)\big).$$

MC 12-4 (Verrauschte Gradienten). Im Kontext von empirischer Risikominimierung definieren wir die Gradienten

$$g_n(\theta) = \frac{1}{n} \sum_{i=1}^n \nabla L(\theta, X_i), \qquad g(\theta) = \mathbb{E}\big(\nabla L(\theta, X_1)\big),$$

wobei wir annehmen, dass $\nabla L(\theta, X_1)$ quadrat-integrierbar ist. Welche Aussagen sind korrekt? (Mehrere richtige Antworten sind möglich.)

(a) $g_n(\theta)$ ist in Erwartung gleich $g(\theta)$.

(b) $g_n(\theta)$ konvergiert in Wahrscheinlichkeit gegen $g(\theta)$.

(c) $g_n(\theta)$ hat Varianz von Ordnung $O(1/n)$.

(d) $\sqrt{n}(g_n(\theta) - g(\theta)$ konvergiert in Verteilung gegen eine Normalverteilung.

Lösung:

(a) Richtig, dank Linearität des Erwartungswerts.

(b) Richtig, dank Gesetz der grossen Zahlen.

(c) Richtig, dank Unabhängigkeit der Beobachtungen.

(d) Richtig, dank zentralem Grenzwertsatz.

Aufgabe 12-5 (Implementierung). Verwenden Sie die Software-Bibliothek Ihrer Wahl, um die Funktion f im statistischen Modell aus Aufgabe 12-3 (a) für folgende Daten (X_i, Y_i), $i \in \{1, \dots, 100\}$, zu lernen:

```
import numpy as np
x = np.random.uniform(-2, 2, 100)
y = np.random.normal(10*np.sin(x**2), 1)
```

Lösung:

```
import numpy as np
import torch
from torch import nn, optim
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt

x = np.random.uniform(-2, 2, 100)
y = np.random.normal(10*np.sin(x**2), 1)

x_tensor = torch.tensor(x, dtype=torch.float32).unsqueeze(1)
y_tensor = torch.tensor(y, dtype=torch.float32).unsqueeze(1)

dataset = TensorDataset(x_tensor, y_tensor)
dataloader = DataLoader(dataset, batch_size=30, shuffle=True)

model = nn.Sequential(
    nn.Linear(1, 1000),
    nn.ReLU(),
    nn.Linear(1000, 1)
)

loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=1e-3)

for epoch in range(1000):
    total_loss = 0.0
```

```
    for batch_x, batch_y in dataloader:
        pred = model(batch_x)
        loss = loss_fn(pred, batch_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

model.eval()
with torch.no_grad():
    x_plot = np.linspace(-2, 2, 300)
    x_plot_tensor = torch.tensor(x_plot, dtype=torch.float32).unsqueeze(1)
    y_pred = model(x_plot_tensor).squeeze().numpy()
    y_true = 10*np.sin(x_plot**2)

plt.figure(figsize=(8, 5))
plt.plot(x_plot, y_true, label='True function f(x) = 10*sin(x**2)', color='green')
plt.plot(x_plot, y_pred, label='Model prediction', color='blue', linestyle='--')
plt.scatter(x, y, label='Training data', color='red', alpha=0.6)
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```