Machine Learning 3
Refugeeks / intoCODE
2022

Torben Laue, M.Sc.
Florian Kluger, M.Sc.

Gradientenabstieg

Für die folgenden Aufgaben haben Sie eindimensionale (1D) Eingabe-Daten $\mathbf{X}$ und Ground-Truth Ausgabe-Daten $\mathbf{y}$ gegeben. Diese Daten sind in Abbildung 1 als schwarze Kreuze dargestellt. Es sind um $N = 10$ Punkte.

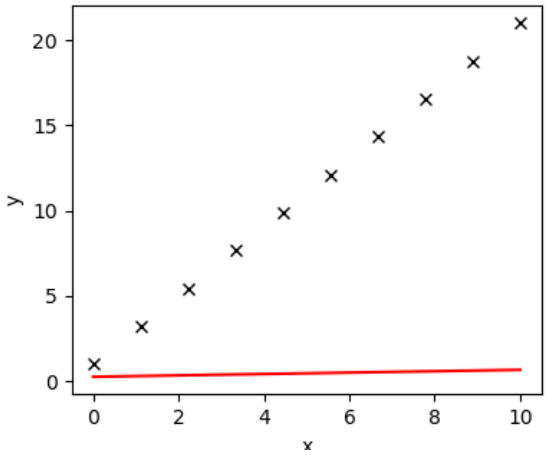

Abbildung 1: Daten (schwarze Kreuze) und initiale Schätzung der Geraden (rote Linie).

Diese Daten können mit einer Geraden beschrieben werden, welche die Gleichung $y = a \cdot x + b$ hat. Ihr Ziel ist es nun, die Parameter a und b zu bestimmen.

Dafür beginnen Sie mit (schlechten) Anfangswerten für a und b, welche die rote Linie $y_e = a \cdot x + b$ in Abbildung 1 ergeben.

Mithilfe des Gradientenabstieg-Verfahrens sollen Sie die folgende Loss-Funktion minimieren:

$$\ell = \frac{1}{N} \sum_{i=1}^{N} (y - y_e)^2 \,. \tag{1}$$

Dies ist der *Mean Squared Error* (MSE, mittlerer quadratischer Fehler).

1 Manueller Gradientenabstieg

Sie sollen nun den Gradientenabstieg selbst implementieren. Verwenden Sie dafür die Datei `simple_gradient_descent.py`

Die grundlegende Struktur des Codes ist vorgegeben. Die Parameter $\mathbf{w} = [a, b]$ sind im Tensor `weights` gespeichert und werden zufällig initialisiert. Schauen Sie sich den Code und die Kommentare zunächst sorgfältig an.

Implementieren Sie dann folgende Schritte innerhalb der for-Schleife:

(a) Berechnen Sie $y_e = a \cdot x + b$.

(b) Berechnen Sie die Loss-Funktion ℓ.

(c) Berechnen Sie mit dem `backward()` Befehl die Gradienten $\frac{\partial \ell}{\partial \mathbf{w}}$.

(d) Führen Sie einen Schritt des Gradientenabstiegs $\mathbf{w} \leftarrow \mathbf{w} - \lambda \cdot \frac{\partial \ell}{\partial \mathbf{w}}$ durch. Verwenden Sie als Lernrate λ die Variable `learning_rate`.

Wenn Sie alles richtig gemacht haben, sollte das Ergebnis etwa so aussehen:

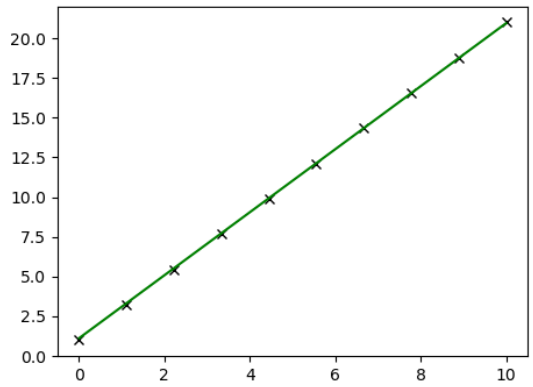

Abbildung 2: Daten (schwarze Kreuze) und optimierte Gerade (grüne Linie).

2 Gradientenabstieg mit Optimizer

Sie sollen nun den Gradientenabstieg mit dem *Adam*-Optimizer implementieren. Verwenden Sie dafür die Datei `gradient_descent_with_optimizer.py`

Die Struktur des Codes ist so ähnlich wie in der vorherigen Aufgabe.

Implementieren Sie folgende Schritte:

(a) Erstellen Sie *vor der for-Schleife* ein Objekt der Adam-Klasse. (`https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`)

(b) Berechnen Sie $y_e = a \cdot x + b$.

(c) Berechnen Sie die Loss-Funktion ℓ.

(d) Berechnen Sie mit dem `backward()` Befehl die Gradienten.

(e) Führen Sie mit dem `step()`-Befehl des Optimizers einen Schritt des Gradientenabstiegs durch.

Das Ergebnis sollte so ähnlich wie in Aufgabe 1 aussehen.