Übungsblatt 2: Gradientenverfahren
Einführung in Deep Learning für Visual Computing

Deadline: **Theoretische Aufgaben** 06.05.2025 - 14:00 via Stud.IP

a) Uneingeschränkte Optimierung ***(10 Punkte)*** Die Rosenbrock Funktion ist eine differenzierbare, nicht konvexe Funktion, die häufig zum Vergleich von unterschiedlichen Optimierungsalgorithmen verwendet wird. Für Parameter a, b ist die Funktion definiert als

$$f(x, y) = (a - x)^2 + b(y - x^2)^2. \tag{1}$$

- Implementieren Sie eine Funktion `rosenbrock(x, y, a, b)`, die sowohl den Funktionswert $f(x, y)$ entsprechend Gleichung 1 für die Parameter a, b zurückgibt, als auch den Wert des Gradienten $\nabla_f(x, y)$.
- Implementieren Sie eine Funktion, die die Rosenbrock Funktion als logarithmisches Konturdiagramm (logarithmisch entlang z-Achse) im Intervall $[-4, 4]^2$ mit Parametern $a = 1$ und $b = 100$ visualisiert. Zusätzlich soll in dem selben Plot das globale Minimum $(x, y) = (1, 1)$ der Funktion markiert sein.

Implementieren Sie jeweils eine Klasse, die jeden der unten gelisteten Optimierungsalgorithmen implementiert. Der Konstruktor Ihrer Klasse soll jeweils die zu optimierenden Parameter und die Hyperparameter des Algorithmus als Argument erwarten. Des Weiteren soll Ihre Klasse die Funktion `step(self, grad)` implementieren, die den Parameter entsprechend des implementierten Algorithmus verändert.

- Implementieren Sie eine Klasse `GradientDescent`, die Gradientenabstieg **ohne** Momentum verwendet um den Parameter optimieren.
- Implementieren Sie eine Klasse `MomentumGradientDescent`, die Gradientenabstieg **mit** Momentum verwendet um den Parameter optimieren.
- Implementieren Sie eine Klasse `RMSPropGradientDescent`, die RMSProp **ohne** Momentum verwendet um den Parameter optimieren.
- Implementieren Sie eine Klasse `AdaMomentumGradientDescent`, die RMSProp **mit** Momentum kombiniert um den Parameter optimieren.

Nachdem Sie die vier Optimierungsalgorithmen implementiert haben, ist nun Ihre Aufgabe die Performance der Verfahren zu vergleichen. Verwenden Sie die Methoden um das globale Minimum der Rosenbrock Funktion zu finden, angefangen von dem Punkt $(x, y) = (-2, -2)$.

Implementieren Sie dazu eine Funktion `optimize(optim, steps)`, die eine Instanz eines Optimierungsverfahrens als Argument nimmt und den Algorithmus `steps`-mal anwendet um das Minimum der Rosenbrock Funktion mit Parameter $a = 1, b = 100$ zu finden. Ihre Evaluierung der Verfahren soll dabei die folgenden Fragestellungen beantworten:

- Ein Vergleich der Methoden in dem alle Verfahren die selben Hyperparameter verwenden und für die gleiche Anzahl an Schritten angewendet werden.
- Ein Vergleich der Methoden in dem alle Verfahren optimiere Hyperparameter verwenden und für eine optimiere Anzahl von Schritten angewendet wird.
- Visualisieren Sie die Rosenbrock Funktion, das globale Minimum der Funktion und den Pfad zum Minimum aller Verfahren in einem Plot (jeweils ein Plot mit gleichen und optimierten Hyperparametern).
- Erläutern Sie Ihre Beobachtungen. Gehen Sie dabei besonders auf die Magnitude der Gradientenwerte, die Rolle der adaptiven Lernrate und des Momentumterm ein.

b) Mini-Batch Optimierung in PyTorch *(10 Punkte)* Ziel dieses Aufgabenteils ist es Sie mit den den Grundlagen der Parameteroptimizerung in PyTorch vertraut zu machen. Der Optimierungsprozess in PyTorch lässt sich in zwei Schritte einteilen:

Forward-Pass In diesem Schritt werden die aktuell geschätzten Parameter verwendet um unser Modell auszuwerten. Der resultierende Wert wird dann verwendet um den Wert der Zielfunktion `loss` zu berechnen. Wichtig ist, dass es sich bei diesem Wert um ein Skalar handelt.

Backward-pass Während des Forward-Pass konstruiert PyTorch einen gerichteten azyklischen Graphen (DAG, directed acyclic graph) mit allen Operationen die ausgeführt wurden. Mit Hilfe des DAG kann PyTorch den Gradient unserer Zielfunktion und Modells automatisch berechnen. Um den Forward-Pass "zu beenden" und den Gradienten berechnen zu lassen muss lediglich `loss.backward()` aufgerufen werden. Anschließend können wir die Parameter unseres Modells optimieren indem wir `optimizer.step()` aufrufen.

Im Gegensatz zu unserer Implementierung in Aufgabenteil *a*) müssen wir den berechneten Wert des Gradienten weder selbst speichern noch an die `step` Funktion des Optimierers übergeben. Das Framework abstrahiert dies in dem es den Gradienten eines Tensor (für den ein Gradient berechnet werden soll) als sein Attribut speichert. Wir können auf den Gradienten eines Tensor `tensor` mit `tensor.grad` zugreifen. Zum Lösen dieser Aufgabe ist ein manueller Zugriff auf den Tensor allerdings nicht notwendig. Wichtig ist, dass PyTorch uns die Entscheidung überlässt den Gradienten mit `optimizer.zero_grad()` auf null zurückzusetzen. Verwenden Sie nun die oben beschrieben Funktionen von PyTorch um das Minimum der Rosenbrock Funktion zu finden:

- Erstellen Sie einen Tensor, der für $x, y = (-1, -1)$ speichert und stellen Sie sicher, dass für diesen ein Gradient berechnet wird.
- Wählen Sie einen Optimierungsalgorithmus aus dem torch.optim Submodul aus und erzeugen Sie eine Instanz dieses Optimierers bei dem Sie den zuvor erzeugten Tensor als Parameter übergeben.
- Implementieren Sie eine Funktion `rosenbrock_th(x,y,a,b)`, die nur Gleichung 1 berechnet (die Funktion muss nicht den Wert des Gradienten zurückgeben).
- Optimieren Sie den Parameter für 1000 Schritte. Achten Sie darauf, dass in der Schleife Ihres Programms, `optimizer.zero_grad()`, `loss.backward()` und `optimizer.step()` in der richtigen Reihenfolge aufgerufen werden.
- Visualisieren Sie die Rosenbrock Funktion, ihr Minimum und die Punkte, die während der Optimierung erzeugt wurden, in einem Plot.

Bisher haben wir direkt die Argumente x, y der Rosenbrock Funktion optimiert um ein Minimum zu finden. In Deep Learning Anwendungen ist das Optimierungsproblem allerdings in der Regel etwas anders formuliert: Wir haben eine Menge von Observationen und wollen den Parameter unseres Modell optimieren, so dass der Fehler zwischen den Observationen und der Vorhersage unseres Modell minimiert wird. Um ein solches Optimierungsproblem mit der Rosenbrock Funktion nachzustellen, nehmen wir an, dass wir eine Menge aus Observationen $\{(x, y, z)\}$ wobei $z = f(x, y)$ haben, aber die Parameter a, b unbekannt sind. Unser Ziel ist es nun die Parameter a', b' mit Hilfe von Gradientenabstieg zu optimieren, so dass wir den Fehler L zwischen unsere Schätzung und den tatsächlichen Werten minimieren:

$$\arg\min_{a',b'} L(f'(x, y), z)$$

1. Implementieren Sie eine Klasse, die von `torch.nn.Module` erbt. Stellen Sie sicher, dass im Konstruktor Ihrer Klasse, der Konstruktor der Elternklasse aufgerufen wird. Darüber hinaus soll der Konstruktor Ihrer Klasse die Parameter a', b' als Membervariablen hinzufügen und mit zufälligen Werten initialisieren. Zusätzlich soll Ihre Klasse noch die Methode `forward(self, xy)` implementieren, die den Funktionswert der Rosenbrock Funktion für die Argumente x, y mit den Parametern a', b' zurückgibt.
2. Erzeugen Sie eine Instanz Ihrer Klasse und verwenden Sie den mini-batch Ansatz aus der Vorlesung zusammen mit den oben beschrieben Techniken um den erwarteten L_1-Fehler zwischen der Schätzung Ihrer Klasse $f'(x, y)$ und der tatsächlichen Werten $z = f(x, y)$ zu minimieren:

   $$\mathbb{E}_{x,y\sim\mathcal{U}_{[-1,1]}}\left[|f'(x, y) - z|\right].$$

   Erzeugen Sie die Observationen in dem Sie x, y gleich verteilt aus dem Intervall $[-1, 1]^2$ ziehen und z mit Hilfe von `rosenbrock_th(x,y,a=1, b=100)` berechnen.

3. Visualisieren Sie die Fehlerkurve für die ersten 100 Schritte der Optimierung bei einer Batchgröße von 1, 32 und 128 in einem Plot. Welchen Einfluss hat die Batchgröße auf das Konvergenzverhalten der Optimierung?

Hinweis: Wenn eine Klasse von torch.nn.Module erbt, können Sie auf alle Parameter dieser Klasse mit `object.parameters()` zugreifen und sie an den Optimierer übergeben.

c) Optimierung mit Nebenbedingung ***(5 Bonuspunkte)*** In der dritten Aufgabe werden Sie Ihre eigene PyTorch-Optimierungsklasse implementieren. Ihr Optimierer soll die Gradientenrichtung verwenden, um den Parameter zu aktualisieren und dabei sicherstellen, dass $\|x\|_2 = 1$ nach jedem Schritt gilt. Wir werden dann diesen Optimierer verwenden, um eine Lösung für das eingeschränkte Optimierungsproblem zu finden

$$x^* = \arg \min_{x \in \mathbb{R}^2; \|x\|=1} f(x) \text{ wobei } f \text{ in Gleichung 1 definiert ist.} \tag{2}$$

Um einen eigenen Optimierer zu implementieren, werden wir die Klasse `optim.Optimizer` erweitern, die die meisten PyTorch-spezifischen Details für uns erledigt:

1. Implementieren Sie eine Klasse `ConstraintOptimizer`, die von der Klasse `torch.optim.Optimizer` erbt. Der Konstruktor Ihrer Klasse soll eine Liste von Parametern und eine Lernrate als Argument erwarten. Erstellen Sie ein Dictionary, das einen Schlüssel `lr` mit den Lernraten als Wert enthält. Rufen Sie den Konstruktor der übergeordneten Klasse auf und übergeben Sie die Parameterliste und das Dictionary als Argumente.

2. Implementieren Sie eine Memberfunktion `step(self, closure=None)`, die den Decorator `@torch.no_grad()` verwendet. Diese Funktion soll jeden Parameter in der Parameterliste aktualisieren. Weitere Einzelheiten finden Sie in der Referenzimplementierung der Klasse SGD. Anstelle der standardmäßigen additiven Aktualisierungsregel für den Gradientenabstieg sollen Sie eine benutzerdefinierte Aktualisierungsregel implementieren:

   - Verwenden Sie die Gradientenrichtung, um eine parametrische Kurve auf der Einheitskugel $\mathcal{B}_{\mathbb{R}^2}$ zu identifizieren. Hinweis: Verwenden Sie die Parametrisierung einer Kugel.
   - Machen Sie einen Schritt entlang der Kurve in Gradientenrichtung und setzen Sie das neue Gewicht als die resultierende Position auf der Einheitskugel. Die Schrittweite sollte proportional zur Lernrate sein. Der Wert eines Tensors `p` kann durch den Aufruf von `p.set_(newP)` ersetzt werden.

3. Verwenden Sie Ihre Optimiererklasse, um eine Lösung für Gleichung 2 zu finden und sowohl die Funktion $f(x)$ auf $x \in \{x \in [-1, 1]^2 \mid \|x\| = 1\}$ als auch den Parameter in jeder Iteration zu visualisieren. Beginnen Sie die Optimierung an dem Punkt $x = \frac{1}{\sqrt{2}}[-1, 1]$, und setzen Sie die Lernrate auf $\eta = 0.05$.