Die in Tabelle 1 und Tabelle 2 enthaltenden Zeiten, zeigen, dass die Verwendung von expliziten Ableitungen keinen Vorteil gegenüber der Verwendung von automatic differentiation erbringt. Bis auf die Berechnung von J für den Fall N=10000, dauerte die Berechnung mittels AD immer kürzer. Interessant ist, dass die Zeiten für die explizite Berechnung der zweiten Ableitung fast linear mit der Anzahl der auszuwertenden Punkte steigen. Bei der AD fällt auf, dass bei der Berechnung beider Ableitungen die Zeiten einen größeren Sprung von N nach m machen als bei m zu k, sodass man auf keine Linearität bzgl. der Anzahl der auszuwertenden Punkte schließen kann. Dieses Verhalten lässt sich auch bei der ersten Ableitung mit der expliziten Methode erahnen.

Als nächstes stellen wir die neuronalen Netzwerke vor, welche wir für unsere numerischen Experimente verwenden.

In diesem Kapitel stellen wir einen Ansatz vor, bei welchen wir ein

Wir führen numerische Experimente durch, in welchen

Dabei wollen wir die Abweichung der durch die jeweiligen Netzwerke produzierten Outputs von denen durch die in Sektion 3 vorgestellte FVM Methode erzeugten Outputs auf demselben Punktegitter vergleichen, um den überlegeneren Typ von neuronalen Netzwerk zu bestimmen.

In sektion 3 präsentieren wir Methoden zur expliziten Berechnung der Ableitungen erster und zweiter Ordnung bzgl. des Inputs der in Sektion 3 vorgestellt Typen von neuronalen Netzwerken. Wir führen numerische Tests durch, durch welche ersichtlich werden soll, ob diese anstatt automatic differentiation zu verwenden sinnvoll ist, wenn der in sketion 2 vorgestellte Approach in Python implementiert ist.

Durchschnittlich benötigte Zeit in Millisekunden zur Berechnung von Ableitungen erster und zweiter Ordnung eines fully-connected FNN.

Wir führten mit unseren Implementierungen numerische Tests durch, in welchen wir die für die berechnungen der Werte mit der expliziten Methode benötigte Zeit mit der benötigten Zeit

Wir führten numerische Tests für unsere Implementierungen, welche den Pseudocodes gegeben durch Gleichung 3 und gleichung

Wir vergleichen die benötigte Zeit bei allen numerischen Tests für die Anwendung der jeweiligen Methode auf mehrere Punkte gleichzeitig, da sowohl unsere Implementierungen wie auch die durch Gradienttape zur verfügung gestellte AD

ein Numerisches Experimente durch, in welchem wir die benötigte Zeit für die Berechnung

in welchem überprüft wird, ob sich die explizite Berechnung der Ableitungen zum Einsatz in unseren PINN Approach lohnt, indem wir die benötigte Zeit zur Berechnung der expliziten Ableitungen mit der benötigten Zeit der äquivalenten Werte berechnet durch D mit Hilfe von Gradienttape von Tnesorflow. Dadurch

Es sei f ein FNN mit ein-dimensionalen Output, d.h. n = 1, dann kann die erste Ableitung von durch den Gradienten und die zweite Ableitung von f durch die Hesse Matrix dargestellt werden. In diesem Fall lassen sich natürlich der Gradient iterativ durch Gleichung 3 und die Hesse Matrix iterativ durch Gleichung 4 berechnen. Wir haben diese rekursiven iterativen Berechnungen für diesen Fall in Python implementiert. Dabei folgt die Methodik der Implementierung zur Berechnung von F dem nachfolgenden Pseudocode. Für die Implementierung zur Berechnung von G können einfach alle Zeilen 1, 2 und 3 weggelassen werden. Durch geschickte Anwendung von mehrdimensionalen Arrays und Vektorisierung, welche in Python einfach umzusetzen ist, haben wir für ein FNN mit n>1 die rekursive Berechnung für die Jacobi Matrix definiert durch Gleichung 3 und eine rekursive Berechnung für die Einträge gegeben durch Gleichung 4 für den Tensoren H implementiert. Dabei wurde die Implementierung die gerade erwähnte Implementierung für alle Outputs des Netzwerkes vektorisiert angewendet.

Der nachfolgende Pseudocode beschreibt, wie man den Gradienten und die Hesse Matrix eines eindimensionalen FNN mit L-schichten, so wie es durch Gleichung 3 und Gleichung 4 beschrieben wird, in einer Implementierung berechnet. Möchte man nur den Gradienten berechnen, verzichtet man auf alle Zeilen, welche für die Berechnung von eta notwendig sind, also auf Zeile 1, 2 und 3. Wir haben in Python die durch den nachfolgenden Pseudocode gegebene Methodik zur Berechnung von x, delta und eta implementiert und durch Ausnutzung multidimensionaler Arrays und die durch Python einfach umzusetzende Vektorisierung auf ein FNN mit mehrdimensionalem Output verallgemeinert.

Natürlich

Diese oben vorgestellte Methode zur Berechnung der expliziten Ableitungen eines FNN haben wir in Python implementiert. Dabei folgt die Methodik dem nachfolgenden Pseudocode. Da sich in Python viele Implementierungen einfach vektorisieren lassen, konnte diese Implementierung durch kleine Änderungen auch auf ein allgemeines FNN angewendet werden

Wir haben ein numerisches Experiment durchgeführt, bei welchem wir die benötigte Zeit für die Berechnungen der expliziten Ableitungen mit der benötigten Zeit der Berechnung mit automatic differentiation der Geichen ausdrücke. Wir haben dieses Experiemnt einmal für ein FNN f , welches dem neuronalen Netzwer gegeben durch Gleichung 3 entspricht, durchgeführt und ein weiteres mal welches dem Netzwerk gegeben durch gleichung 4 entspricht.

In dieser Arbeit wurde noch ein zweiter Typ von neuronalen Netzwerk vorgestellt, das sogenannte Resnet gegeben durch Gleichung 3.

Auch für dieses leiten wir explizite Ableitungen erster und zweiter Ordnung her. Die erste Ableitung ist definiert durch

Der interessante Term hier ist gegeben durch X, wobei wir hierfür keine Herleitung angeben, da das durch wiederholte Anwendung der Kettenregel und durch Gleichung 3 und Gleichung 4 dem Leser selbst überlassen ist. Es sei nur so viel gesagt, dass die durch Gleichung 4 gegeben rekursive Berechnung für x zunächst ausgeschrieben wird und dann die daraus entstehenden einzelnen Terme abgeleitet werden.

Um nun die zweite Ableitung gegeben durch Gleichung 4 des gesamten Outputs des FNN gegeben durch gleich 3 zu bekommen, müssen wir Gleichung 5 für jeden Eintrag des Outputs anwenden.

Sodass wir L Terme in R haben, in welchen immer jeweils ein D bzgl. x innerhalb des Gradienten abgeleitet wird, und die Summe dieser L Terme die Hesse matrix von f ergibt.

Wir bemerken, dass D = W gilt.

Daraus folgt, dass wir die Produktregel L mal anwenden müssen. Wir gehen wie folgt vor: Wir betrachten den Gradienten für die l-te Schicht, welcher gegen ist durch D, siehe Gleichung 3.

Wir differenzieren in G nun D bzgl. x, wobei wir dies elementweise machen, da x in jedem Diagonalelement von D auftaucht. Natürlich müssen wir auch hier die Kettenregel wieder anwenden:

Um die Ableitung nun

Um die Notation zu vereinfachen, definieren wir die folgende Matrix:

In einer Implementierung wird der Gradient gegeben durch Gleichung 3 rekursiv berechnet.

Bei einer Implementierung des Gradienten gegeben durch Gleichung 3, kann dieser rekursiv berechnet werden, wobei man ähnlich wie bei der Backpropagation vorgeht.

Um den Gradienten in einem Computerprogramm zu berechnen, geht man rekursiv vor.

Wir bemerken, dass die abbildung natürlich selbst auch ein neuronales Netzwerk ist.

Dabei wird zunächst das neuronale Netzwerk an dem Punkt x ausgewertet und während die Information vorwärts durch das Netz propagiert, wird in jeder Schicht die Aktivierung gespeichert. Dann wird der sogenannte Fehler rückwärts durch das Netz propagiert, indem man die Kettenregel von der letzten Schicht zur ersten Schicht in umgekehrter Richtung anwendet. Die folgende iterative Berechnung kann zu diesem Zweck verwendet werden

Wir bemerken, dass D gilt.

Um die Hesse matrix für f zu erhalten, müssen wir den gradienten nochmals bzgl. x ableiten. Da x für die Berechnung von jedem D gebraucht wird, müssen wir jedes D nach x ableiten.

Wir sehen in Gleichung 3, dass wir nl mal die Produktregel anwenden müssen.

Das ist möglich, weil der Output

In diesem Kapitel werden verschiedene Preliminaries eingeführt und mathematische Grundlagen geschaffen, welche für die Betrachtung von neuronalen Netzwerken benötigt werden. Zuerst stellen metrischen Graphen und ihre Eigenschaften vor. Dann definieren wir auf so einem metrischen Graphen eine Menge von Differentialgleichungen, welche Dift-Diffussiongleichungen genannt werden und den Verkehrsfluss auf einem kompakten Straßennetz modellieren. Danach stellen wir neuronale Netzwerke mit ihren Eigenschaften und ihrer Relevanz als Funktionenapproximatoren vor und gehen am ende dieses Kapitels auf PINNs ein, welche eine Methode zur Approximation der Lösung verschiedener Differentialgleichungen bieten und im Moment ein heiß diskutiertes Thema in der angewandten Mathematik sind.

Als nächstes berechnen wir die Ableitung 2. Ordnung des neuronalen Netzwerkes bzgl. einer Inputvariable. Dies ist ein , welcher gegeben ist durch

Da man in partiellen Differentialgleichungen, wie der Name schon sagt, nur einzelne

Desweiteren verwenden wir für ein numerisches Experiment ein neuronales Netzwerk, welches durch folgende Formel definiert ist

In welchem die Dichte

für die numerische Lösung von potenziellen MFG- und MFC-Modellen

zur Lösung hochdimensionaler Probleme durch die Kombination von Lagrangeschen und Eulerschen Gesichtspunkten und die Nutzung der jüngsten Fortschritte beim maschinellen Lernen.

Die Idee stammt aus Paper 3, in welchem hochdimensionale MFG- und MFC-Modelle annähernd gelöst werden, indem Lagrangesche und Eulersche Sichtweisen kombinieren werden und eine maßgeschneiderte Parametrisierung des neuronalen Netzes für das Potenzials der MFG/MFC-Lösung verwendet wird, um jegliche räumliche Diskretisierung zu vermeiden.

Wir lösen hochdimensionale Probleme annähernd, indem wir Lagrangesche und Eulersche Sichtweisen kombinieren und die jüngsten Fortschritte des maschinellen Lernens nutzen.

Schließlich hilft uns eine maßgeschneiderte Parametrisierung des neuronalen Netzes für das Potenzials der MFG/MFC-Lösung, jegliche räumliche Diskretisierung zu vermeiden.

Wir stellen nun unsere Parametrisierung des neuronalen Netzes für das Potenzials

Für das dritte numerische Experiment verwenden wir einen ganz anderen Ansatz neuronale Netzwerke einzusetzen. Wir

Die Benutzung eines einzelnen neuronalen Netzwerkes ist durch die Hoffnung motiviert, dass die Neuronen in den versteckten Schichten sich die Struktur des Graphen und die daraus folgenden Bedingungen für alle kanten antrainieren, d.h. dass alle Wechselwirkungen im Graph von den Neuronen berücksichtigt werden.

Außerdem hoffen wir, dass sich der Rechenaufwand verringert, wenn das Netz die Ausgabe für alle Kanten gleichzeitig erzeugt.

Dass das Netzwerk die Ausgabe für alle Kanten gleichzeitig erzeugt ist ebenfalls ein Vorteil, da in einer Implementierung nur die Ausführung eines Netzwerkes anstatt mehrere notwendig ist.