Bearbeitung der Aufgabe 6.1

Bearbeitet von János Sebestyén und Frank Mehne

Relevante Dateien:
Aufgabe6_1_a.m
Aufgabe6_1_b.m
Aufgabe6_1_a.fig
Aufgabe6_1_b.fig

Beobachtung und Interpretation der Ergebnisse:
Die Beobachtungen aus Aufgabenteil a) und b) wurden zusammengefasst und miteinander verglichen. Daher ist im Folgenden, anders als auf dem Aufgabenblatt, keine Unterscheidung der einzelnen Aufgabenteile ersichtlich.

In der Aufgabe wurden auf zwei verschiedenen Problemstellungen unterschiedlicher Komplexität neuronale Netze trainiert. Zunächst ein Netz mit sechs Neuronen in einer verdeckten Schicht, um als Zielfunktion den Cosinus auf dem Wertebereich [-2π, 2π] zu lernen. Anschließend wurde ein Netz mit drei verdeckten Schichten (5 Neuronen in Schicht 1 und 2, 2 Neuronen in Schicht 3) auf die folgende Zielfunktion

$$f(x,y) = \sin(x) \cdot (0.01 \cdot y^4 + 0.3 \cdot y^3 - 2 \cdot y^2 + y)$$

mit den Wertebereichen [0,4π] für x und [0,5] für y trainiert.

Um die einzelnen Trainingsmethoden miteinander zu vergleichen wurden folgende Trainingsalgorithmen auf der jeweils selben initialen Netzkonfiguration verwendet:

traingd: Einfacher Batch Lernalgorithmus mit Gradientenabstiegsverfahren
traingdm: Gradientenabstiegsverfahren mit Momentumsterm
traingdx: Gradientenabstiegsverfahren mit Momentumsterm und adaptiver Lernrate
trainrp: Resilient Backpropagation (RProp)
traincgf: Konjugierter Gradientenabstieg mit Fletcher-Reeves-Updates
traincgp: Konjugierter Gradientenabstieg mit Polak-Ribiére-Updates
traincgb: Konjugierter Gradientenabstieg nach Powell-Beale
trainscg: skalierter konjugierter Gradientenabstieg
trainbfg: BFGS-Verfahren (Quasi-Newton-Verfahren)
trainoss: OneStepSecant backpropagation
trainlm: Levenberg-Marquardt-Algorithmus

Beobachtungen:
Die drei Gradientenabstiegsverfahren (traingd, traingdm, traingdx) zeigen die schlechteste Performance. Selbst nach 1000 Lernepochen werden von traingd und traingdm beim lernen des Cosinus keine ausreichend kleinen Testfehler erreicht, die zu einem Abbruch des Lernens führen. Traingdx hingegen kann durch seine adaptive Lernrate den Testfehler wesentlich schneller minimieren. Im Fall der komplexeren Zielfunktion brechen die Algorithmen verfrüht ab, ohne den Testfehler besonders gut minimiert zu haben. Dies liegt an einer Abbruchbedingung, die besagt, dass der Lernprozess gestoppt wird, sofern der Validierungsfehler mehrere Epochen nicht kleiner geworden ist. Die einfachen Gradientenabstiegsverfahren sind für komplexere Lernaufgaben daher nicht geeignet.
Um ein vielfaches schneller und besser ist der Algorithmus trainrp, welcher Änderungen des Gradienten der vorangegangenen Epoche mit in die Berechnung der Anpassung jedes Gewichts mit einbezieht.
Vergleichbar hierzu sind die Algorithmen, die auf eine konjugierte Gradienten setzen (traincgf, traincgp, traincgb, trainscg). Hierbei ist bei der Funktionsapproximation des relativ simplen Cosinus kein Unterschied zwischen den einzelnen Arten der Algorithmen mit konjugierten Gradienten erkennbar. Bei der komplexeren Problemstellung allerdings wird ersichtlich, dass lediglich die Methode mit skalierten konjugierten Gradienten (trainscg) nicht zu einem verfrühten Abbruch führt. Sie hat den Vorteil, nicht bei jeder Iteration die aufwändige Berechnung eines line search für den konjugierten Gradienten durchführen zu müssen. Ähnlich schnell wie die Algorithmen mit konjugierten Gradienten sind die Verfahren trainbfg und trainoss. Diese quasi-Newton Verfahren sind zwar bei komplexeren Problemstellungen oft schneller und besser als die Verfahren mit konjugiertem Gradienten, brauchen aber durch die Benutzung der Hesse-Matrix mehr Rechenaufwand und Speicher, auch wenn hier die Inverse der Hesse-Matrix nur angenähert wird um schneller zu sein (zB. bei trainbfg).
Die beste Performance bezogen auf die Minimierung des Testfehlers zeigte der Algorithmus nach Levenberg-Marquardt. Sowohl bei weniger komplexen Zielfunktion, als auch bei der komplexeren Funktion konnte der Testfehler mit dem Algorithmus gut minimiert werden. Ein Nachteil des Algorithmus ist allerdings sein hoher Bedarf an Rechenleistung und Speicherplatz. Dies machte sich vor allem bei der komplexeren Problemstellung bemerkbar. Hier rechnete der Algorithmus in etwa genau so lang, wie alle anderen Algorithmen zusammen. Bei komplexen Berechnungen ist von dieser Methode also abzuraten, sofern nicht Rechenzeit/-leistung und Speicherkapazität im Überfluss vorliegt.