
IT-Projekt

Künstliche Neuronale Netze

August 2021

Inhaltsverzeichnis

IT-Projekt.....	1
Inhaltsverzeichnis.....	2
Ziele.....	3
Künstliche Intelligenz.....	4
Neuronale Netze.....	4
Topologie.....	5
Neuron.....	6
Netzwerktypen.....	7
Aktivierungs-Funktionen.....	8
Lernen / trainieren.....	9
Bibliotheken.....	10
Vorbereitete Software.....	11
Aufgaben.....	12
Einfache Beispiele.....	12
AND-Gatter.....	12
XOR-Gatter.....	12
2-Bit-Binärzahl.....	14
Mustererkennung / Bilderkennung.....	15
Weitere mögliche Aufgaben und Ergänzungen.....	16
Dokumentation.....	17
Doxygen.....	17
SVN.....	17
Gnuplot.....	17
Quellen.....	18

Ziele

Hier soll ein Programm (oder mehrere) mit einem künstlichen neuronalen Netzwerk erstellt werden. Das Programm bzw. das darin realisierte neuronale Netz soll auf eine bestimmte Aufgabe trainiert werden. Die Erkennung handgeschriebener Ziffern Null bis Neun wäre eine bevorzugte, jedoch sehr aufwändige Aufgabe. Es sind jedoch auch andere Verwendungszwecke denkbar. Für ein erstes „Kennenlernen“ sind aber einfachere Aufgaben eher geeignet.

Die Lösung der Grundaufgabe soll passend erweitert werden. Insbesondere hier können und sollen eigene Ideen verwirklicht werden. So kann der Einfluss der verwendeten Algorithmen auf Genauigkeit und die Geschwindigkeit der Konvergenz, auf das Fehlerverhalten und auf Grenzen der Erkennbarkeit numerische untersucht werden.

Das IT-Projekt sollte auch durch Teile ergänzt werden, die nicht unmittelbar in die Programmierung der Grundaufgabe einfließen. Eine Software-Dokumentation mittels Doxygen oder ein Versionskontrollsystem auf der Basis von SVN sind solche Ergänzungen, die auch in die Benotung eingehen.

Dieses IT-Projekt kann auf einem PC oder auf einem Notebook bearbeitet werden. Tablets oder gar Smartphones sind nur unzureichend geeignet, da sie zu wenig Komfort für die Entwicklung bieten.

Künstliche Intelligenz

Künstliche Intelligenz (KI, engl. artificial intelligence AI) ist ein Bereich der Datenverarbeitung. Gerade in letzter Zeit sind einige Aspekte der künstlichen Intelligenz besonders bekannt geworden – Erkennung der Gesichter von Personen, autonom fahrende Autos, etc. Neuere Technologien, etwa spezielle Prozessoren, haben die Leistungsfähigkeit der KI drastisch erhöht.

Zu unterscheiden sind insbesondere Expertensysteme, Mustererkennung und neuronale Netze. Expertensysteme sind beispielsweise geeignet, anhand von aufgeführten Symptomen eine oder mehrere Krankheiten als wahrscheinlich zu diagnostizieren. Mustererkennung wird bei Texterkennung, Gesichtserkennung und vielem mehr angewendet. Bilderkennung und deren Interpretation sind eine Domäne der künstlichen neuronalen Netze.

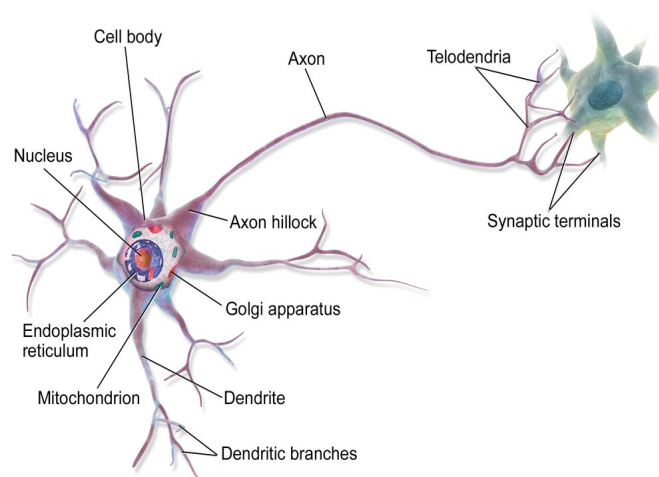
Neuronale Netze

Wikipedia schreibt im Artikel “neuronales Netz“

Als neuronales Netz wird in den Neurowissenschaften eine beliebige Anzahl miteinander verbundener Neuronen bezeichnet, die als Teil eines Nervensystems einen Zusammenhang bilden, der einer bestimmten Funktion dienen soll. Abstrahiert werden in Computational Neuroscience darunter auch vereinfachte Modelle einer biologischen Vernetzung verstanden. [Wik2021a]

In der Informatik, Informationstechnik und Robotik werden deren Strukturen als künstliches neuronales Netz modelliert und technisch nachgebildet, simuliert und abgewandelt. [Wik2021a]

Neuron und deren Verknüpfung über Axone zu Synapsen sind das biologische Vorbild der künstlichen neuronalen Netze.



Multipolar Neuron [Wik2021b]

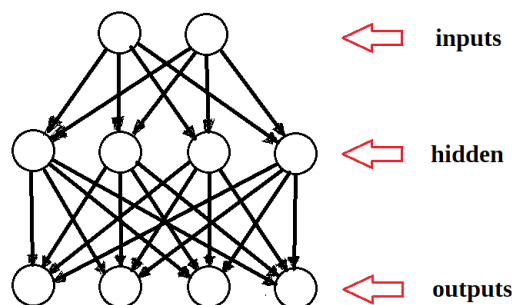
Real ausgeführte künstliche neuronale Netze entsprechen dem biologischen Vorbild jedoch nur in sehr begrenzter Form. Oftmals enthalten künstliche neuronale Netze Eigenschaften, die in der Natur so nicht nachgewiesen werden konnten oder nicht vorhanden sind.

Nachfolgend sind zur Abkürzung unter neuronalen Netzen immer die künstlichen neuronalen Netze gemeint.

Topologie

Ein neuronales Netz besteht aus einigen, oft sehr vielen Neuronen/Zellen und Axonen/Verbindungen zwischen diesen. Die Verbindungen von einem Neuron zu einem oder meist mehreren anderen Neuronen ist gerichtet. Dies entspricht auch dem Verhalten natürlicher Axone.

Für gewöhnlich unterscheidet man Neuronen im Eingangsbereich (input neurons), Neuronen im Ausgangsbereich (output neurons) und dazwischen liegende, versteckte Neuronen (hidden neurons) oftmals in mehreren Schichten angeordnet. In den zeichnerischen Darstellungen liegt üblicherweise die Eingangs- zu Ausgangs-Richtung von links nach rechts oder von oben nach unten. Die Axone folgen dieser Richtung meist, können bei manchen Netzwerktypen jedoch auch andere Wege nehmen (siehe unten).



Im Beispiel hier liegt ein Netz mit 2 Eingangs-Neuronen, 4 versteckten Neuronen in nur einer Ebene und 4 Ausgangs-Neuronen vor. Von jedem Neuron gehen Axone zu allen Axonen der nachfolgenden, nächsten Schicht. Dieses 2-4-4-Netz wird auch in folgenden Beispielen in ähnlicher Form benutzt.

Dabei ist es nicht erforderlich, dass ein Neuron wie im Beispiel oben mit allen Neuronen der vorangegangenen Schicht und nur mit diesen verbunden ist. Viele andere Topologien sind möglich und werden auch genutzt – BIAS Neuron oder rekurrente Netze.

Die Aufgabenstellung an das neuronale Netz wird an die Eingangs-Neuronen geleitet. Jedes einzelne Eingangs-Neuron erhält eine der gemessenen oder sonstwie vorliegenden Eigenschaften in digitalisierter Form. Durch die Bearbeitung des (trainierten) Algorithmus in dem Netz wird an den Ausgangs-Neuronen das Ergebnis dargestellt. Jedes Ausgangs-Neuron gibt eine einzelne Eigenschaft wieder. Typischerweise in Form einer Wahrscheinlichkeit mit Zahlenwerten zwischen 0 und 1.

Als ein besonders einfaches Beispiel denke man an die Interpretation zweier Bits als Eingangswerte

zu einem UND- oder ODER-Gatter. Das erforderliche neuronale Netz benötigt 2 Eingänge für die beiden Bits, einen Ausgang für das Ergebnis der UND- oder ODER-Verknüpfung und (eventuell) eine dazwischen liegende Schicht aus versteckten Neuronen. Für eine so triviale Aufgabe wäre eigentlich kein neuronales Netz erforderlich.

Ein weiteres einfaches Beispiel ist die Interpretation einer Binärzahl aus genau 2 Bits. Mit 2 Bit sind die (Dezimal-)Zahlen 0 bis 3 darstellbar. Auch hier wäre für eine so triviale Aufgabe kein neuronales Netz erforderlich. Die beiden Bits der Binärzahl gehen an die beiden Eingangs-Neuronen und nach der Bearbeitung des (trainierten) Algorithmus im Netz zeigen die Ausgangs-Neuronen das Ergebnis an. Die vier Ausgangs-Neuronen zeigen jeweils eines der möglichen Ergebnisse 0 bis 3 an. Für eine binäre 1 (als Bits 0 und 1) an den beiden Eingangs-Neuronen zeigte beispielsweise das zweite Ausgangs-Neuron zuständig für die Erkennung der 1 einen hohen Wahrscheinlichkeitswert und alle anderen zuständig für die Erkennung von 0, 2 und 3 einen niedrigen Wahrscheinlichkeitswert.

Bei dem “normalen” Betrieb eines neuronalen Netzes werden die Eingangsdaten nur einmalig in Vorwärtsrichtung durch das Netz geleitet – FeedForward. Beim Trainieren eines Netzes kommt eine zusätzliche und mathematisch deutlich aufwendigere Berechnung hinzu – BackPropagation.

Neuron

Ein Neuron enthält zunächst nur einen recht einfachen Algorithmus.

Die Aktivität als interner Zwischenwert des Neurons wird gebildet als Summe seiner gewichteten Eingänge (Axone).

$$q_j = \sum w_{ij} * o_i$$

Über eine Aktivierungsfunktion oder Transferfunktion F wird daraus der Ausgangswert des Neuron.

$$o_j = F(q_j)$$

Eine Verbindung von einem Neuron zu einem anderen enthält ein Gewicht w für diese Verbindung. Das Gewicht wird nur während des Trainings (aka Lernen) verändert, bleibt aber nach dem Training konstant. Für das Training und nur dafür enthält die Verbindung zusätzlich einen Wert für die letzte beobachtete Änderung.

Ein Neuron selbst enthält nur den Ausgangswert und nur für das Training zusätzlich einen Gradienten für das Maß der Änderung.

Die Transferfunktionen sind häufig für alle Neuronen gleich, müssen es aber nicht sein. Die verwendeten Transferfunktionen liefern für einen reellen Eingangswert typischerweise ein Resultat im Bereich -1 bis +1 (z.B. tanh) oder aber im Bereich 0 bis +1 (z.B. sigmoid).

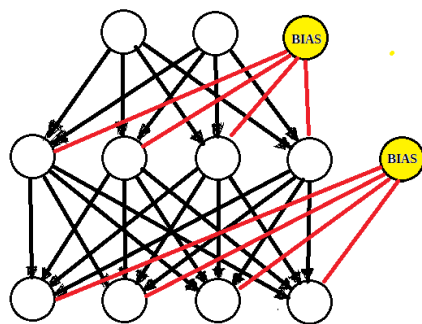
Die Verbindungen werden meist an einer praktisch zugänglichen Stelle hinterlegt. Die Schichten sind oft dafür geeignet. Aber die Verbindungen können auch implizit oder durch eine Tabelle an beliebiger Stelle definiert sein.

Ein erheblicher Teil der Mathematik eines neuronalen Netzes kann mittels Matrizen beschrieben und berechnet werden. Hierin liegt auch der Grund für die Verwendung von Graphik-Prozessoren bzw. Graphikkarten für die Berechnung. Darüber hinaus werden auch spezielle Prozessoren für neuronale Netze eingesetzt.

Netzwerktypen

Die einfachsten künstlichen neuronalen Netze (ANN) sind feed forward nets, also Netze wie das oben erwähnte, welches nur Verbindungen aus einer Schicht in die nächst folgende enthält. Dabei können Verbindungen von jedem Neuron einer Schicht zu sämtlichen Neuronen der nächsten Schicht gehen (siehe oben) oder aber es gehen nur einzelne Verbindungen von ausgewählten Neuronen einer Schicht zu den Neuronen der nachfolgenden Schicht.

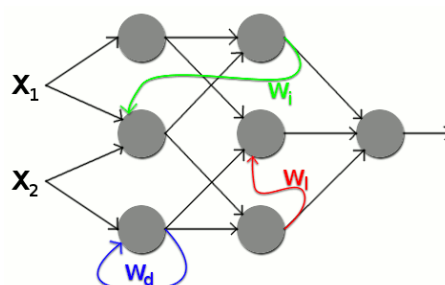
Häufig findet man in ausgeführten Netzen sogenannte BIAS-Neuronen. Von diesen gehen Verbindungen zu anderen Neuronen. Jedoch gibt es keine Verbindungen die zu den BIAS-Neuronen führen. BIAS-Neuronen liefern immer den konstanten Wert 1 als ihren Ausgangswert. Beim Training werden aber auch die Gewichte der Verbindungen von den BIAS-Neuronen zu den anderen Neuronen eingestellt.



Netz mit BIAS-Neuronen (gelb)

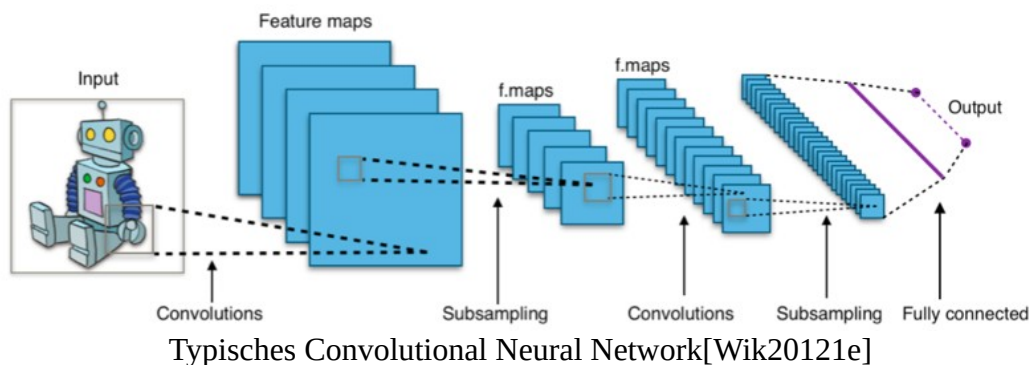
Ein BIAS-Neuron in der Ausgangs-Schicht hat keinen Einfluß und kann daher weggelassen werden. Manchmal ist es aus Gründen der Symmetrie oder zur Vereinfachung des Programms dennoch vorhanden.

Enthalten neuronale Netze eine Rückkopplung über einzelne Neuronen spricht man von rekurrenten Netzen (recurrent nets, RNN).



Rekurrentes neuronales Netz [Wik2021c]

Bei Convolutional Neural Networks werden zunächst nur Teile eines Datensatzes für eine nachfolgende Schicht (feature maps) bearbeitet und ein weiteres neuronales Netz angeschlossen.



Diese Topologie eines neuronalen Netzes ist vor allem für Bilddateien und andere 2- oder 3-dimensionale Daten geeignet.

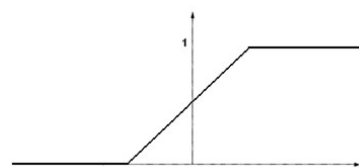
Aktivierungsfunktionen

Als Aktivierungsfunktion oder Transferfunktion sind viele, mehr oder weniger einfache Funktionen geeignet. Der Hauptzweck liegt in einer Zuordnung von einem beliebigen Fließkommawert zu einem Wert im Bereich 0 bis 1 oder -1 bis 1.

Häufig verwendete Aktivierungsfunktionen
[Wik20121f], teilweise [SAE2021]

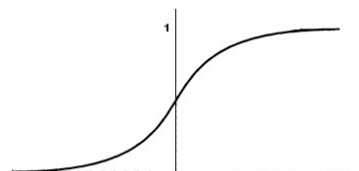
linear ramp or piecewise linear

$$F(q_j) = \begin{cases} 0 & q_j < T_1 \\ \frac{(q_j - T_1)}{(T_2 - T_1)} & T_1 < q_j < T_2 \\ 1 & q_j > T_2 \end{cases}$$



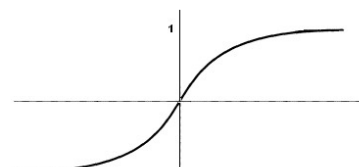
sigmoide

$$F(q_j) = 1 / (1 + e^{-a \cdot q_j})$$



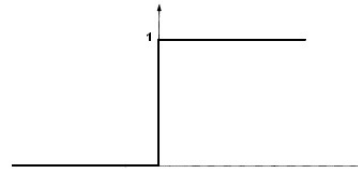
hyper tangent

$$F(q_j) = \tanh(q_j)$$

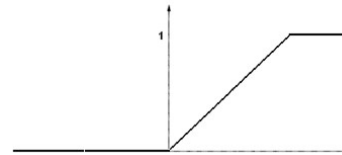


binary step

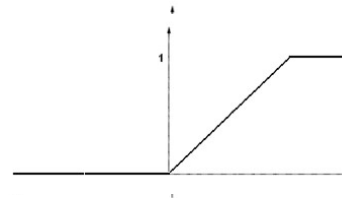
$$F(q_j) = \begin{cases} 0 & : q_j \geq T \\ 1 & : q_j < T \end{cases}$$

**ReLU (rectified linear unit, most used)**

$$F(q_j) = \max(0, q_j)$$

**Leaky ReLU (leaky rectified linear unit)**

$$F(q_j) = \begin{cases} q_j & : q_j > 0 \\ \min(-1, b * q_j) & : q_j \leq 0 \end{cases}$$

**Lernen / trainieren**

Zunächst müssen (die meisten) neuronale Netze für ihre jeweilige Aufgabe trainiert bzw. angelernt werden (supervised learning). Hierzu werden dem Netz eine größere Anzahl von Beispielen und die erwarteten Resultate vorgeführt. Nach der normalen Auswertung eines Beispiels durch das Netzwerk (feed forward) erfolgt ein zusätzlicher, oftmals sehr aufwendiger weiterer Berechnungsschritt (back propagation), der interne Daten (Gewichte und Gradienten) passend zum erwarteten Resultat modifiziert. Dieses Training wird bei neuronalen Netzen “Lernen“ genannt.

Ein solches Training ist nicht in allen neuronalen Netzen vorhanden (unsupervised learning). Auch ist es in den natürlichen neuronalen Netzen nicht vorhanden.

Bei der back propagation wird (häufig) ein Gradientenabstieg als Lernregel verwendet. Hiermit wird der quadratische Fehler zwischen gewünschtem und erhaltenem Resultat an den Ausgangsneuronen minimiert.

Bibliotheken

Um die Entwicklung von neuronalen Netzen zu beschleunigen wurden eine Reihe von freien und/oder proprietären Bibliotheken veröffentlicht. Für den Einsatz in kommerziellen Produkten sind diese geradezu ideal. Als Grundlage für ein erstes Studium und tieferem Verständnis der Funktion der Netze sind diese jedoch weniger geeignet.

Beispiele sind (Reihenfolge ohne Bewertung)

TensorFlow

<https://www.tensorflow.org/>

free and open source (Google)

Keras

<https://github.com/keras-team/keras>

Python Schnittstelle zu Tensorflow

Caffe

<http://caffe.berkeleyvision.org/>

BSD licence

opennn

<https://www.opennn.net/>

open source

MemBrain

<https://www.membrain-nn.de/>

open source, free for private and noncommercial use

Microsoft Cognitive Toolkit

<https://www.g2.com/products/microsoft-cognitive-toolkit-formerly-cntk/reviews>

open source

und viele weitere

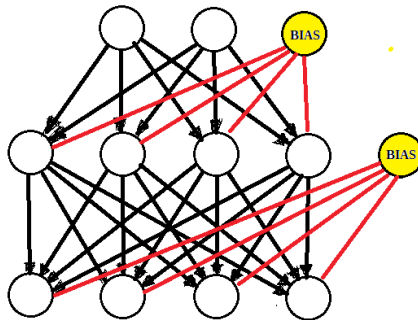
Vorbereitete Software

Im IT-Projekt hier wird eine stark überarbeitete Version der Software von David Miller, [Mil2011] eingesetzt als Grundlage für eine Einführung und für Beispiele. Hier können die Neuronen gewissermaßen „von Hand und einzeln“ angefasst werden. Auch für den weiteren Ausbau und die Projektbearbeitung sollte diese Software als Basis dienen, weil beispielsweise Hilfestellungen einfacher sind. Jedoch sind andere Lösungswege ebenfalls möglich und geeignet.

Für den Einstieg in diese Software ist zunächst einmal der grundlegende Aufbau des neuronalen Netzes – hier `NeuralNetwork` genannt - zu kennen. Ausgangspunkt ist die Anzahl der Ebenen des Netzes – hier `Layer` genannt – im `NeuralNetwork` und die Anzahl der Neuronen – hier `Neuron` genannt – in einem `Layer`. Zusätzlich kann jedes `Layer` ein `BIAS`-Neuron als letztes Neuron enthalten.

Vorhanden ist die Klasse `NeuralNetwork` für das komplette neuronale Netz, die Klasse `Layer` für eine einzelne Ebene, die Klasse `Neuron` für die Neuronen in einer Ebene und die Struktur `Connection` für die Verbindungen eines Neuron zu denen des nachfolgenden Layers. Dargestellt wird die Netzstruktur durch einen Topologie-Vektor bestehend aus der Anzahl der Neuronen der einer Ebene ohne die `BIAS`-Neuronen.

Das oben dargestellt 2-4-4-Netzwerk jedoch mit `BIAS` Neuronen hat einen Topologie-Vektor mit den Werten $\{ 2, 4, 4 \}$. Es erhält somit 3 `Layer` mit 3, 5 und 4 Neurons. Also jeweils 1 Neuron zusätzlich als `BIAS`-Neuron in jeder Ebene ausgenommen der letzten Ausgangsebene. Den kompletten Aufbau und die Feineinstellung eines neuronalen Netzes erledigen der Konstruktor der Klasse `NeuralNetwork` und einige weitere Methoden der Klasse.



Für die „normale“ Bearbeitung/Analyse durch das neuronale Netz wird dem Netz über die Funktion `FeedForward` ein passender Satz Eingangsdaten zugeführt und mit der Funktion `GetResults` wird das Ergebnis abgefragt.

Zunächst aber muss das Netz trainiert werden. Hierzu wird dem Netzwerk zunächst und sehr oft ein vorbereiteter Satz Eingangsdaten angeboten und durch die Funktion `FeedForward` verarbeitet und jeweils direkt danach mit dem zum Eingangsdatensatz passenden Datensatz für das Resultat der erwartete Ausgangswert für die Funktion `BackPropagation` zugeführt. Das Training findet in der Funktion `BackPropagation` statt.

Aufgaben

Einfache Beispiele

Sehr einfache Beispiele, bei denen das Resultat a priori bekannt ist, werden besonders gerne für eine erste Kontaktaufnahme und für vorausgehende Untersuchungen verwendet.

AND-Gatter

Als einfaches erstes Beispiel (zum Anwärmen :-)) liegt ein vorbereitetes Programm 2-1-Net als Quellcode vor. Hierin ist ein neuronales Netzwerk mit zwei Eingangsneuronen und einem Ausgangsneuron für ein AND-Gatter programmiert (ready to use). Es besitzt eine Schicht mit vier versteckten Neuronen und den BIAS-Neuronen. Es kann aber auf einfache Weise umprogrammiert werden – etwa für ein AND- oder XOR-Gatter, für mehr oder weniger versteckte Neuronen oder Schichten, etc. Und es kann als Basis für eine Reihe von Versuchen verwendet werden.

XOR-Gatter

Ein sehr typisches Beispiel ist die Simulation eines XOR-Gatters mit 2 Eingängen. Bei solchen trivialen logischen Gattern ist das zu erwartende Ergebnis schon vorher bekannt. Eigentlich ist hierfür kein neuronales Netz erforderlich. Aber die benötigten Trainingsdaten können leicht durch ein kleines Programm erzeugt werden. Hierfür kann das Programm 2-1-Net als Grundlage dienen.

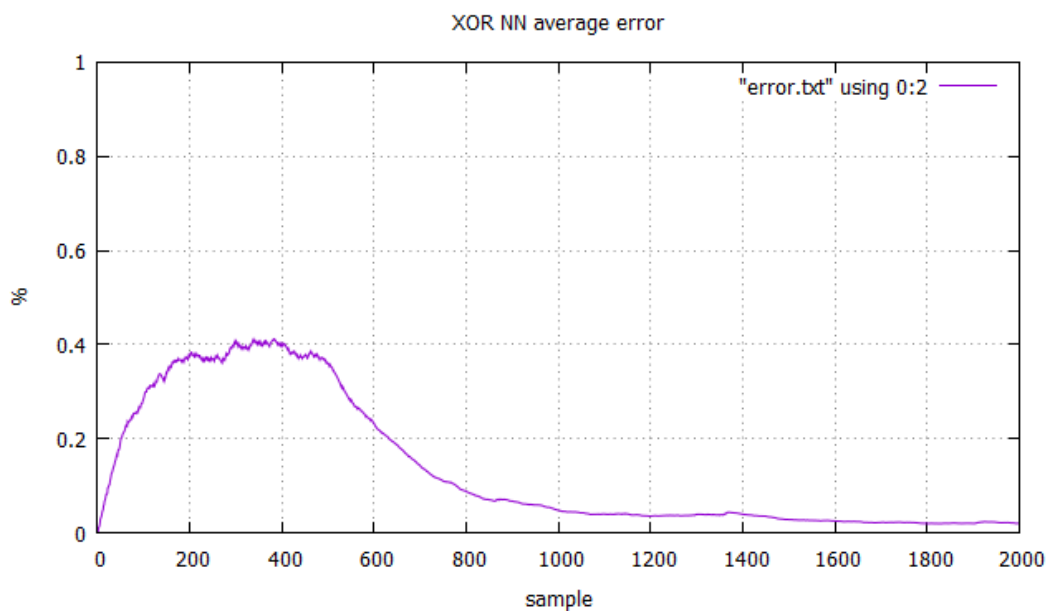
Es sollte ein 2-4-1 neuronales Netzwerk mit BIAS-Neuronen verwendet. Es besitzt 2 Eingangs-Neuronen für die Eingänge des XOR-Gatters, 4 Neuronen für die innere, unsichtbare Struktur und ein Ausgangs-Neuron für das Resultat. Hinzu kommen für jede Ebene ein BIAS-Neuron.

Interessant wäre ein Versuch mit einem 2-1 neuronales Netzwerk mit einem BIAS-Neuron, also ohne eine versteckte Schicht. Anders als bei einem AND- oder OR-Gatter (ebenfalls versuchen) kann ein solches Netz nicht für ein XOR-Gatter verwendet werden.

Das Programm XOR_NN_trained enthält ein bereits trainiertes neuronales Netz zur Simulation eines Exklusiv-Oder-Gatters. Auf Eingabe von zwei Bits, etwa „0 1“ (mit Leerzeichen zwischen den Ziffern) erhält man die ermittelte Antwort des Gatters in Form von Wahrscheinlichkeiten.

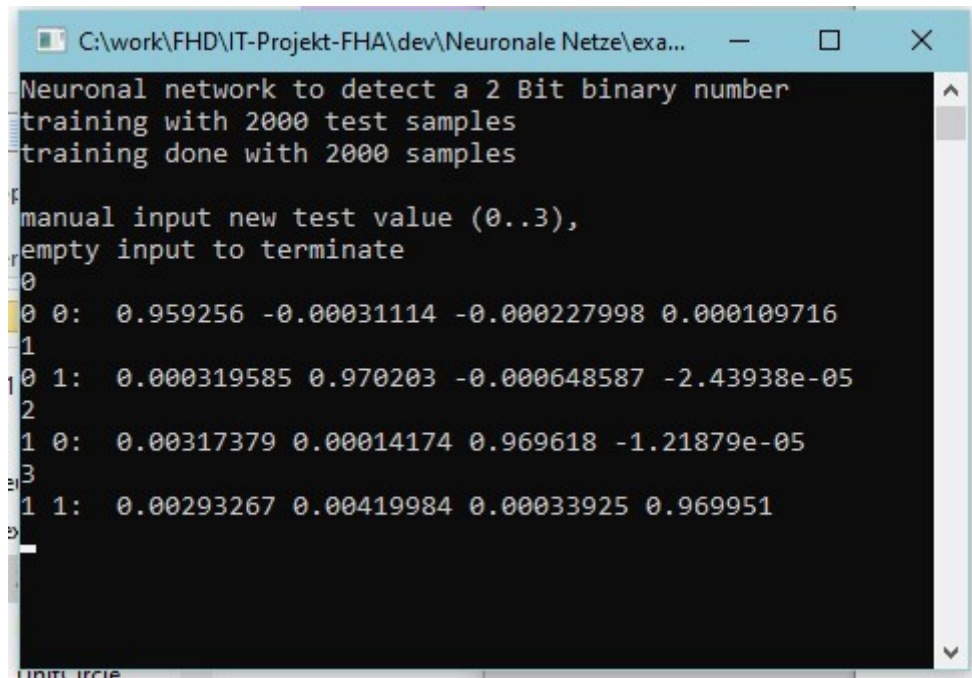
```
C:\WINDOWS\system32\cmd.exe
Neural net with back propagation for a XOR gate
manual input new test data for XOR gate, e.g. "0 1",
empty input to terminate
0 0
result: 0.00407489
1 1
result: 0.00725784
1 0
result: 0.972662
0 1
result: 0.97228
```

Der Erfolg des Trainings kann über die Anzeige des intern berechneten Fehlers oder berechneten durchschnittlichen Fehlers angezeigt werden.



2-Bit-Binärzahl

Mit einem nur wenig größeren 2-4-4 neuronalen Netz kann wie oben erwähnt eine zweistellige Binärzahl erkannt werden. Ein passendes Programm 2BitNumber kann leicht auf der Basis des vorhandenen Programms 2-1-Net erstellt werden..



```
C:\work\FHD\IT-Projekt-FHA\dev\Neuronale Netze\exa...
Neuronal network to detect a 2 Bit binary number
training with 2000 test samples
training done with 2000 samples

manual input new test value (0..3),
empty input to terminate
0
0 0:  0.959256 -0.00031114 -0.000227998 0.000109716
1
0 1:  0.000319585 0.970203 -0.000648587 -2.43938e-05
2
1 0:  0.00317379 0.00014174 0.969618 -1.21879e-05
3
1 1:  0.00293267 0.00419984 0.00033925 0.969951
```

Die vier Werte der angezeigten Ausgangs-Neuronen stellen die Wahrscheinlichkeit für die Erkennung der Zahlen 0 bis 3 dar. In der letzten Zeile im Bild ist der vierte Wert ungefähr 1 (sehr wahrscheinlich), die anderen drei Werte sind ungefähr 0 (sehr unwahrscheinlich).

Für die Ausgabe könnte man auch eine Bewertungsfunktion hinzufügen und das Resultat fertig interpretiert anzeigen.

Mustererkennung / Bilderkennung

Eine mögliche Aufgabe in diesem IT-Projekt ist die Erkennung von handgeschriebenen Ziffern 0 bis 9 aus kleinen Bildern. Mit dem vorhandenen, einfach aufgebauten neuronalen Netz ist diese Aufgabe schwer und kaum vollständig zu lösen. Zunächst andere Aufgaben zu suchen und zu lösen wäre sinnvoll.

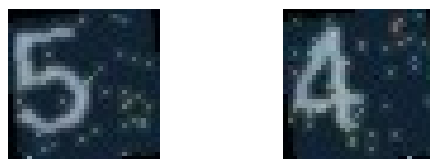
Eine bekannte Sammlung von 60000 plus weitere 10000 handgeschriebene Ziffern [Cun2021, Wik20121d] liegen in kompaktierten Dateien vor. Hier sind alle Pixel aller Bilder direkt in einer Datei zusammengefasst. Die einzelnen Bilder haben jeweils 28x28 monochrome Pixel unterschiedlicher Helligkeit. Eine weitere Datei enthält in passender Reihenfolge die abgebildeten Ziffern in Binärform.



Man beachte, dass in Teilen der Welt die Ziffer 1 als einfacher senkrechter Strich geschrieben wird. Hiermit wird die Unterscheidung der Ziffern 1 und 7 vereinfacht.

Die Bilder der Ziffern sind aber oftmals auch für den menschlichen Betrachter kaum erkennbar. Einige Ziffern sind eher unleserliches Gekritzel.

Eine Bildersammlung von weiteren 5000 handgeschriebene Ziffern von [Mil2011] liegen dem Projekt ebenfalls vor.



Die dort enthaltenen Bilder besitzen jeweils 32x32-Pixels. Nach einer Vorverarbeitung der einzelnen Pixel wären sie damit für ein neuronales Netz mit 32x32 Eingangs-Neuronen geeignet.

Naheliegenderweise sollte das verwendete neuronale Netz 10 Ausgangs-Neuronen für die erkannten Ziffern 0 bis 9 besitzen.

Für die Verwendung von Bilddateien kann die freie Bibliothek FreeImage [Fre2018] genutzt werden. Sie verarbeitet eine Vielzahl von Bilddatei-Formaten und enthält viele Funktionen zur Manipulation. Ein sehr einfaches Beispiel befindet sich im Verzeichnis `extract_pix`. Dort sind auch die Bibliotheken bereits erstellt und können unmittelbar gelinkt werden.

Weitere mögliche Aufgaben und Ergänzungen

Schon ein extrem einfaches künstliches neuronales Netz bestehen aus 2 Eingangs-Neuronen und einem Ausgangs-Neuron ohne zusätzliches BIAS-Neuron und ohne eine versteckte (hidden) Schicht kann erfolgreich auf das Ergebnis einer OR-Verknüpfung trainiert werden. Versuchen Sie dies auch für AND- und XOR-Verknüpfungen.

Es sind viele andere lohnende Aufgaben für ein künstliches neuronales Netz denkbar. Diese können auch gänzlichen anderen Arbeitsbereichen zugeordnet sein – z.B. Überwachung eines Bankkontos oder Warenlagers.

Eine Analyse der Mathematik in den diversen Funktionen eines neuronalen Netzes kann zu einem besseren Verständnis führen und die Grundlage für Änderungen und Versuche bilden. Hier tut sich ein weites Feld für Ergänzungen auf, beispielsweise durch Austausch der Transferfunktionen und Analyse ihres Einflusses.

Eine Bewertung der Leistung des neuronalen Netzes, etwa Anhand von Fehler-Metriken.

Wichtige sind auch Variationen des neuronalen Netzes und ein Vergleich untereinander. Es können z.B. mehr Neuronen, mehr Ebenen, mit/ohne BIAS-Neuron, Rückführungen ausprobiert werden.

Ein neuronales Netz mit 3 Eingangs- und 3 Ausgangs-Neuronen könnte als Eingang die 3 Bits einer 3-stelligen Binärzahl erhalten und auf die Berechnung der Bits der jeweils nächsten Zahl trainiert werden.

Ein neuronales Netz mit 1 Eingangs- und 1 Ausgangs-Neuron könnte zu Approximation einer stetigen Funktion $y = F(x)$ benutzt werden. Hier sollte man sich auf einen kleinen Bereich für x-Werte beschränken – etwa $[-1..1]$ und erforderlichenfalls auch die y-Werte auf den Bereich $[-1..1]$ oder gar $[0..1]$ skalieren.

Smartphone App oder PC-Programm zur manuellen Eingabe der Ziffern.
(sehr aufwendig)

Dokumentation

Ein wesentlicher Teil des IT-Projektes ist eine Dokumentation. Diese sollte aus einer funktionalen Beschreibung der Regelung und der erstellten Software hierfür bestehen.

Die Form der Dokumentation ist freigestellt. Typisch und häufig verwendet wird hierzu Word. Auch Fotografien, ggf. Video sind inzwischen als erläuternder Teil in Dokumentationen gängig.

Es ist aber auch möglich (und vielleicht sogar handlicher und leistungsfähiger) die Dokumentation direkt im Quellcode des Programms einzupflegen und mit Doxygen zu erstellen.

Doxygen

Doxygen ist eine weitverbreitete Software zur Dokumentation von Software innerhalb ihres Quellcodes. Gesteuert von speziellen Markierungen in den Quelldateien kann sie Dokumentationen in verschiedenen Formaten erstellen. Hier im IT-Projekt ist eine Dokumentation in FoProgrammsrm von HTML-Seiten und kompiliertem HTML etwas vorbereitet. Die HTML-Seiten werden mit einem HTML-Compiler in CHM-Dateien umgesetzt und komprimiert.

Siehe auch den Quellcode des Beispiels Xyz.cpp, Xyz.CHM und Xyz.doxy.

SVN

Software soll meist über den Tag hinaus bestehen bleiben und muss reproduzierbar sein. Da Software meist weiterentwickelt wird, sind zur Fehlersuche häufig auch die vorangegangenen Versionen wichtig. Als Backup und zur Versionsverwaltung werden verschiedene Software eingesetzt.

Eine solche, weit verbreitete und moderne Versionsverwaltung ist SVN (Apache Subversion). Die Software des IT-Projektes soll auf einem SVN-Server hinterlegt werden.

Gnuplot

Gnuplot ist eine viel verwendete, freie Software zur Anzeige von datenbasierten Graphiken. Mit ihr kann u.a. das zeitliche Verhalten von Eigenschaften in Form von y-t-Diagrammen dargestellt und als Bild für weitere Zwecke bereitgestellt werden.

Im einfachsten Fall werden die Ausgaben bzw. Messdaten der Software in passender Form in einer Textdatei protokolliert. Gnuplot kann daraus dann diverse graphische Darstellungen erzeugen.

Erforderliche Eingaben im Gnuplot Kommando-Fenster, wenn die vorhandenen Messdaten in der Datei irgendwo\Result.dat vorliegen.

```
set title("Result");           // netter Titel für die Grafik
set xlabel("sample");          // Beschriftung der horizontalen Achse
set ylabel("%");               // Beschriftung der vertikalen Achse
set grid;                      // Gitter anzeigen
plot "Result.dat" using 1:2 with lines // Auswahl der anzuzeigenden Daten in der Form von Linien
```

Gnuplot besitzt eine erhebliche Menge an Einstellungen für viele verschiedene grafische Darstellungen. Entsprechend umfangreich ist die Dokumentation von Gnuplot.

Um die Einstellbefehle nicht ständig neu tippen zu müssen, können die Kommandos auch in einer Script-Datei hinterlegt und durch einfaches klicken im Datei-Explorer gestartet werden.

Auch ist es möglich Diagramme mit Gnuplot unmittelbar von einem C/C++-Programm aus zu erzeugen und anzuzeigen.

Quellen

- [Fre2018] FreeImage 2018, <https://freeimage.sourceforge.io/>
- [Cun2021] Yann le Cun, MNIST database of handwritten digits,
<http://yann.lecun.com/exdb/mnist/>
- [Mil2011] David Miller, <https://inkdrop.net/dave/docs/neural-net-tutorial.cpp>
- [SAE2021] https://www.saedsayad.com/artificial_neural_network.htm
- [Wik2021a] https://de.wikipedia.org/wiki/Neuronales_Netz, 2021-03-20
- [Wik2021b] Bruce Blaus,
https://commons.wikimedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png
- [Wik2021c] Mercyse, CC BY-SA 4.0,
<https://commons.wikimedia.org/wiki/File:Neuronal-Networks-Feedback.png>
- [Wik2021d] MNIST database, https://en.wikipedia.org/wiki/MNIST_database
- [Wik2021e] Aphex34, 2015, https://commons.wikimedia.org/wiki/File:Typical_cnn.png
- [Wik2021f] activation function https://en.wikipedia.org/wiki/Activation_function
