CW – Dekodierung:
Versuche mit neuronalen Netzen

Inhaltsverzeichnis

1. Vorwort

Auf der Suche nach Verfahren zur Dekodierung von CW-Signalen hat sich als eine Möglichkeit die Verwendung von neuronalen Netzen angeboten. Von den Versuchen, die Möglichkeiten der CW Dekodierung mit Hilfe eines neuronalen Netzes mit dem „Wissen“ eines Laien auszuloten soll hier berichtet werden.

2. Wie es dazu kam

Schon während der Vorbereitung auf die Amateurfunkprüfung in den 60er Jahren kam der Wunsch auf, CW-Signale nicht nur „mühsam“ persönlich zu dekodieren und handschriftlich zu Papier zu bringen zu müssen sondern das Ganze auch automatisieren zu können. Zu dieser Zeit waren „Rechner“ mit denen so etwas möglich gewesen wäre für einen normalen Schüler absolut unerschwinglich. Deshalb blieb es vorerst bei dem Wunsch.

2.1 Die ersten Schritte

In der 80er Jahren war es dann doch so weit – es gab die ersten brauchbaren Mikroprozessoren. Ich hatte mir damals eine Testplatine mit einem TMS9900 besorgt – ein Prozessor der den anderen zu dieser Zeit üblichen schon um viele Jahre voraus war:

- es war ein 16Bit – Prozessor
- er hatte keinen Akku sondern man konnte zusätzlich zu drei „Hardware“-Registern selbst Registersätze zu je 16 Registern definieren wobei man jedem Interrupt seinen ureigenen Registersatz zuordnen konnte.
- Diese Softwareregister waren alle „orthogonal“ d.h. alle Befehle konnten auf alle Register in gleicher Weise angewendet werden.
- Die Maschinensprache war so eingängig, dass es sogar leichter war den Prozessor in hexadezimaler Form zu programmieren als in dem mitgelieferten Assembler.

Mit diesem Prozessor war es schon nach einem Tag geschafft eine CW-Kodierung zu schreiben, also auf einer Tastatur eingegebene Zeichen als CW auszugeben. Im nächsten Schritt sollte die Dekodierung von CW realisiert werden.

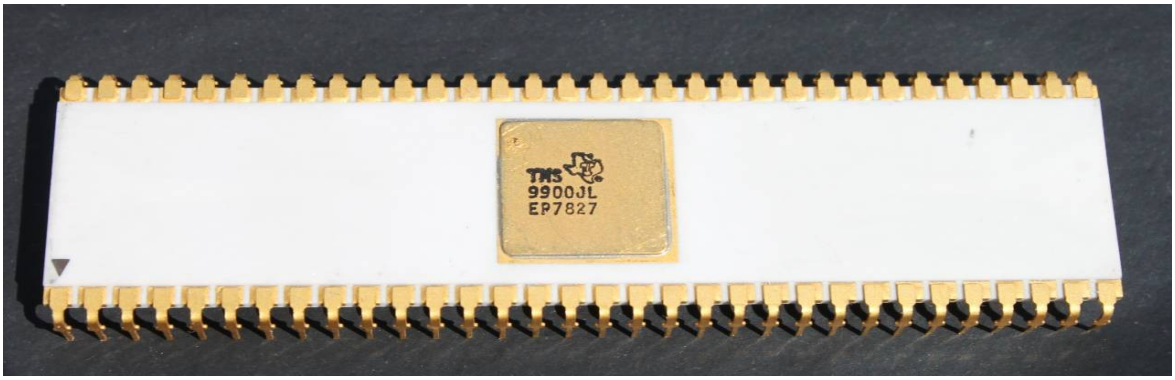

Bild 1:
Der Prozessor für die Realisierung des ersten CW-Dekoders

2.2 Die Dekodierung von CW – erste Erfahrungen

Im Gegensatz zur Kodierung hat sich die Dekodierung als deutlich komplexer erwiesen. Nach etwa 2 Tagen war es dann doch so weit – der erste CW-Dekoder hat funktioniert. Die Tests erbrachten dann folgende Ergebnisse: Der Dekoder war dem eigenen Ohr deutlich überlegen wenn folgende Bedingungen erfüllt waren:

- Das gegebene CW-Tempo musste recht genau dem des Dekoders entsprechen.
- Die Schrift musste Maschinenschrift oder sehr maschinennahe Schrift sein.
- Im Signal durfte so gut wie kein QRM vorhanden sein.

Diese Voraussetzungen konnte man mit einer automatischen Taste für die Tests zwar problemlos schaffen. In der realen Welt – also auf Kurzwelle – sah die Welt ganz anders aus. Damals waren die meisten Schriften noch Handschriften, das Tempo war kaum vorhersehbar und das QRM war nicht schwächer als heutzutage. Kurz – für den realen Betrieb war der Dekoder ungeeignet und das eigene Ohr dem Prozessor weit überlegen.

In den folgenden Wochen habe ich dann versucht, das Programm gegen Handschriften, unterschiedliche Geschwindigkeiten und QRM toleranter zu machen. Gemessen am Aufwand war das Ergebnis aber eher dürftig. Das „Projekt“ CW-Dekodierung wurde deshalb auf Eis gelegt.

3. Ein möglicher neuer Ansatz

Während dieser Zeit kam mir ein erster Artikel über neuronale Netze in die Hände. Nach der Beschreibung hätte genau so ein Netz den passenden Ansatz für einen solchen Dekoder geliefert – für einen Studenten allerdings jenseits aller Realisierungsmöglichkeiten. Der für ein solches Netz erforderliche Hardwareaufwand war einfach um viele Größenordnungen zu hoch. Damit musste auch diese Lösung auf Eis gelegt werden.

Die Situation änderte sich als ich vor einigen Jahren während der Installation einer Linux-Distribution bei den vielen tausend Programmpakten auf ein Paket mit dem Namen SNNS (Stuttgart Neuronal Network Simulator) gestoßen bin. Eine Beschreibung von gut 300 DIN-A4-Seiten war auch dabei. Das Programmpaket war schnell installiert. Beim starten ging eine große Zahl von Fensterchen auf. Die Hoffnung dass das Programm intuitiv zu bedienen sei hat sich leider sehr schnell als trügerisch erwiesen. Damit geriet dieser „Simulator“ vorerst in Vergessenheit.

Einige Jahre später kam mir dieser Simulator auf der Suche nach einem „Hirnkaugummi“ für ein Wochenende wieder in den Sinn. Zwei Tage Kampf gegen die Beschreibung führte schließlich zu ersten kleinen Erfolgen bei der Bedienung des Simulators und dem erneuten aufleben des uralten Dekoderprojekts. Nach den ersten Erfolgen sollte im ersten Schritt ausgeleuchtet werden was sich mit einem neuronalen Netz erreichen lässt und welcher Aufwand für einen guten CW-Dekoder nötig sein könnte.

Von diesen Überlegungen und Versuchen soll hier berichtet werden.

4. Vergleich der Welten

Um die Wirkungsweise und die Einsatzgebiete von neuronalen Netzen besser verstehen zu können ist es sinnvoll deren Funktionsweise, Stärken und Schwächen mit denen der meist besser bekannten Prozessorstrukturen zu vergleichen. Deshalb sollen hier beide Welten gegenübergestellt werden.

4.1 Die klassische Rechnerstruktur

Die klassische Rechnerstruktur könnte man sich als erweiterte „Spieluhr“ vorstellen. Eine Spieluhr besteht aus einem Spielwerk mit „integrierter Ausgabe“ und einem Anweisungsspeicher der die Reihenfolge der Töne vorgibt – oft als Walze mit Nadeln oder als Lochstreifen ausgeführt. Übersetzt in die Rechnerwelt könnte man sich einen primitiven Rechner damit vorstellen als

- einem Rechenwerk (wie dem eines Taschenrechners)
- einem Speicher für die Eingabeparameter und die Rechenergebnisse
- einer Anweisungsliste die Anweisung für Anweisung abgearbeitet wird.

Bleibt man bei genau dieser Struktur halten sich allerdings auch die möglichen Funktionen in engen Grenzen. Erweitert man die möglichen Anweisungen jedoch um eine Klasse, die die Reihenfolge der Abarbeitung der Anweisungen selbst beeinflusst (indem man also bedingte Sprünge innerhalb der Anweisungsliste ermöglicht), wird aus der simplen „Spieluhr“ das was wir heute als Rechner kennen. Die Eigenschaften dieser Struktur sehen so aus:

- Die Struktur ist deterministisch d.h. dass bei gleichen Voraussetzungen (also gleichem Speicherinhalt und gleichem Programm) nach dessen Ablauf immer das gleiche Endergebnis im Speicher steht. Vorausgesetzt natürlich dass das Programm ein wirkliches Ende hat und nicht in einer Endlosschleife läuft.
- Treten Programmzustände auf, die nicht im Voraus bedacht wurden, reagiert die Struktur auch in einer nicht unbedingt vorausgesehenen und erwünschten Weise. In harmlosen Fällen kann das der berühmte blaue Bildschirm sein, in spektakulären Fällen Dinge wie der Absturz der ersten Ariane 5 – Rakete.

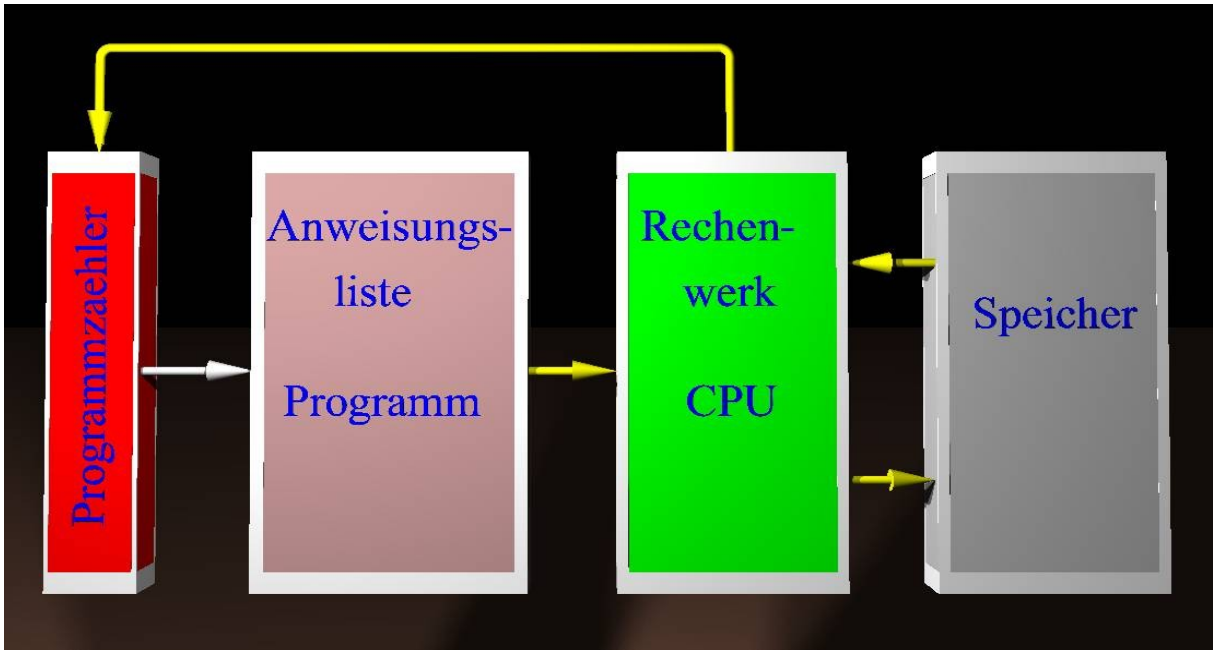

Bild 2:

Die klassische Rechnerstruktur

Mit diesen Eigenschaften lässt sich die beschriebene Rechnerstruktur für die Lösung von Problemen nutzen deren Problemlösung genau bekannt ist und sich exakt beschreiben lässt. Bei komplexeren Problemen muss viel Zeit aufgewendet werden um alle Randbedingungen und mögliche Ereignisse im Voraus zu erfassen um das Systemverhalten in allen möglichen Konstellationen im Griff zu haben.

So groß die Zahl von mit dieser Sorte von Rechnern bearbeitbaren Aufgaben auch ist, gibt es dennoch eine gewaltige Zahl von Problemstellungen sie sich damit kaum oder überhaupt nicht bearbeiten lassen. Das sind alle die Problemstellungen bei denen es keine mathematische Beschreibung gibt wie sich aus den Eingangsparametern das Ergebnis bestimmen lässt. In vielen Fällen sind nicht einmal die Eingangsparameter selbst bekannt. Beispiele für solche Problemstellungen sind Vorhersagesysteme wie

- die Vorhersage von Aktienkursen
- die Vorhersage des Stromverbrauchs in den verschiedenen Regionen eines Versorgungsgebietes über kürzere oder längere Zeiträume
- oder Systeme zum lesen von Handschriften wie z.B. die Adressangaben auf Briefen.

Auf der Suche nach Rechnerstrukturen, die für die Lösung solcher Aufgaben geeignet sind kam schließlich Hilfe von Neurologen die das Verhalten von Nervenzellen aus Gehirnen untersucht haben.

4.2 Vom Neuron zum neuronalen Netz

Bei der Untersuchung von von Nervenzellen aus Gehirnen hat man folgende Eigenschaften herausgefunden:

- Jede Zelle hat eine oder mehrere unterschiedlich große „Einbuchtungen“ an denen andere Nervenzellen „angedockt“ sind.
- Jede Zelle hat einen oder mehrere Fortsätze mit denen sie an anderen Zellen „andockt“.
- Über die Andockstellen – den Einbuchtungen – empfängt die Zelle Reize von den anderen Zellen. Je größer die Fläche der jeweiligen Einbuchtung ist, desto empfindlicher ist die Zelle an diesem Eingang.
- Übersteigt die Summe der (durch die unterschiedlichen Flächen gewichteten) Reize einen bestimmten Schwellwert, dann „feuert“ die Zelle – d.h. sie gibt einen Reiz an die über die an den Fortsätzen angeschlossenen Zellen weiter.

Eine Struktur mit vielen Millionen oder Milliarden dieser eng vermaschten Zellen deren Verhalten auf diesen wenigen Grundprinzipien basiert sollte schließlich das Verhalten ausmachen das wir als das eines Gehirns kennen.

Bild 3:

Schematische Darstellung einer Nervenzelle mit 5 sichtbaren Eingängen und einem Ausgang

5. Die technische Realisierung eines Neurons

Bei den offenbar recht einfachen Verhaltensweisen realer Neuronen lag die Idee nahe, so etwas technisch nachzubauen und zu untersuchen ob sich auch mit solchen technischen Neuronen und daraus aufgebauten Netzen gehirnähnliche Funktionen zu realisieren lassen.

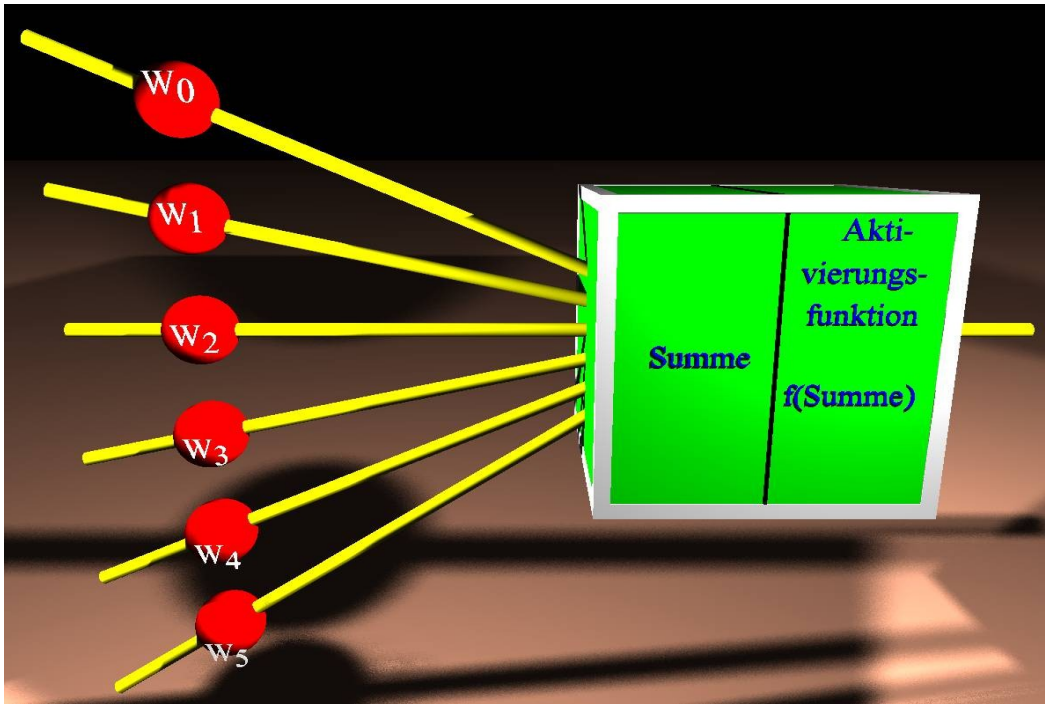

Bild 4:

Beispiel eines technischen Neurons mit 6 gewichteten Eingängen W0 ...W5 und einem Ausgang

Wie aus dem Bild 4 zu erkennen ist, hat das technisch realisierte Neuron einen oder mehrere Eingänge. Jeder dieser Eingänge kann mit einem eigenen „Gewicht“ versehen werden (durch die Kugel dargestellt). Der dort dargestellte Faktor wij gibt das Gewicht des Eingangs i der Zelle j an. „1“ entspricht dabei einer „normalen“ Empfindlichkeit, Werte über 1 demnach einer erhöhten, Werte zwischen 0 und 1 niedrigeren Empfindlichkeit. Eingänge deren Gewicht 0 ist sind wirkungslos. Der Wertebereich lässt sich bei diesen technischen Neuronen zusätzlich auf negative Werte erweitern. Mit solchen Eingängen lassen sich dann positive Reize von anderen Eingängen aus blockieren.

Das Verhalten „wenn die Summe der gewichteten Reize einen bestimmten Wert überschreitet dann feuert die Zelle“ lässt sich in Form einer „Aktivierungsfunktion“ beschreiben. Im Worten ausgedrückt würde das heißen:

- Wenn die Summe aller gewichteten Reize kleiner oder gleich einem bestimmten Schwellwert ist dann 0 ausgeben
- Wenn die Summe aller gewichteten Reize größer als der Schwellwert ist dann 1 ausgeben.

Bei den Experimenten mit neuronalen Netzen hat sich aber gezeigt, dass es aber besser eine etwas andere Aktivierungsfunktion zu verwenden.

$$a = \frac{1}{1 + e^{-(g(\sum w_i o_i - s))}};$$

wobei

wi das Gewicht des Eingangs i ist,

oi der Wert am Eingang i,

g eine Gewichtung die die resultierende Kurvenform bestimmt

s der Schwellwert ist bei dem die Zelle feuern soll und

a das Ergebnis am Zellenausgang

Wie aus dem Bild 5 zu erkennen ist wird mit dieser Funktion der harte Übergang von 0 nach 1 beim überschreiten des Schwellwertes „aufgeweicht“. Je kleiner der Faktor g ist, desto weicher wird der Übergang. Der weiche Übergang wird für ein effektives trainieren eines solchen Netzwerks benötigt.

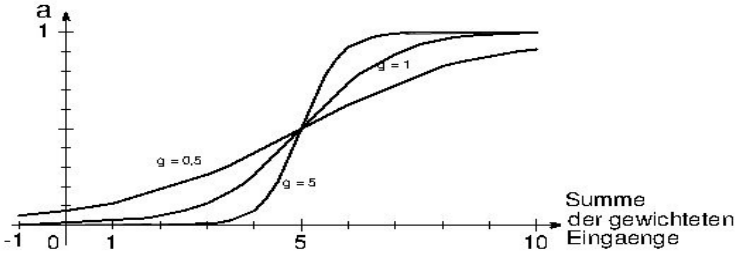

Bild 5:

Darstellung der Aktivierungsfunktion für s = 5 und verschiedene Werte von g

6. Überlegungen zur Konstruktion eines Netzwerks

Dieser Punkt hat sich für die eigenen Überlegungen vorerst als einer der schwierigsten erwiesen. In der Literatur gibt es genügend mathematisch unterfütterte Untersuchungen wie Netzwerke für bestimmte Anwendungen aussehen müssen. Diese Untersuchungen zu verstehen erfordert aber viel mathematisches Wissen und viel Zeit. Nachdem mir genau dieses Wissen fehlt und ich die Zeit nicht aufwenden wollte um in das Thema einzusteigen blieb mir nur einige wenige Grundweisheiten zusammenzukratzen.

Man könnte sich ein universelles Netz vorstellen in dem jedes Neuron mit jedem verbunden ist (einschließlich sich selbst). So eine Struktur bleibt bei ganz winzigen Netzen noch überschaubar. Nachdem die Anzahl der Verbindungen aber quadratisch mit der Zahl der Neuronen ansteigt kommt man sehr schnell in Dimensionen die nicht mehr beherrschbar sind:

10 Neuronen: 100 Verbindungen

100 Neuronen: 10000 Verbindungen

1000 Neuronen: 1000000 Verbindungen.

Damit scheidet diese „Lösung“ wohl aus.

Eine Möglichkeit die universelle Struktur etwas einzuschränken ist die, Rückkopplungen zu vermeiden. Das hat allerdings zwei Konsequenzen

- Das Netzwerk kann sich von einem Takt zum nächsten nichts merken.
- Umgekehrt hat das aber auch den Vorteil, dass damit keine Schwingungen auftreten können.

Schließlich gab es noch die Idee, die Gesamtaufgabe in Teilaufgaben zu zerlegen um es so mit mehreren einfacheren Netzwerken zu tun zu haben.

Beim durchsuchen der Literatur ist aufgefallen, dass es eine Struktur gibt die sehr häufig auftaucht und die damit recht erfolgversprechend erscheint: Eine Ebenenstruktur wobei alle Neuronen einer Ebene untereinander keine Verbindung haben, dafür aber Verbindungen mit den Neuronen der darüberliegenden Ebene (Empfangsrichtung) und der darunter liegenden Ebene (Senderichtung).

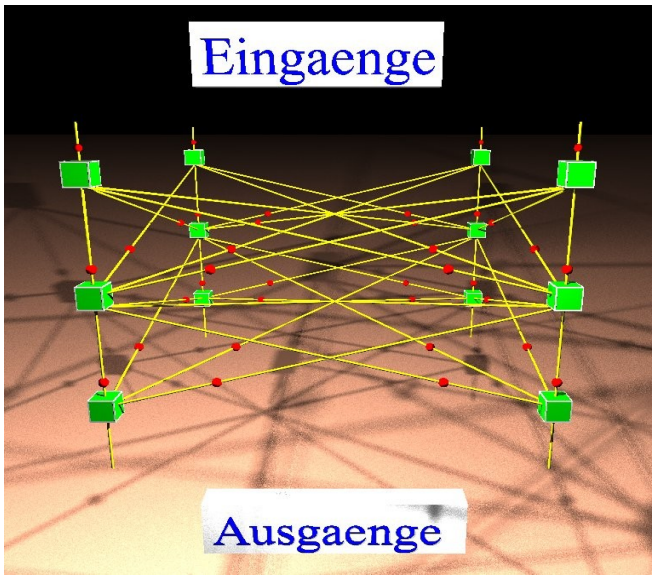

Bild 6:

Beispiel eines neuronalen Netzes in Ebenenstruktur mit 4 + 4 + 4 Neuronen

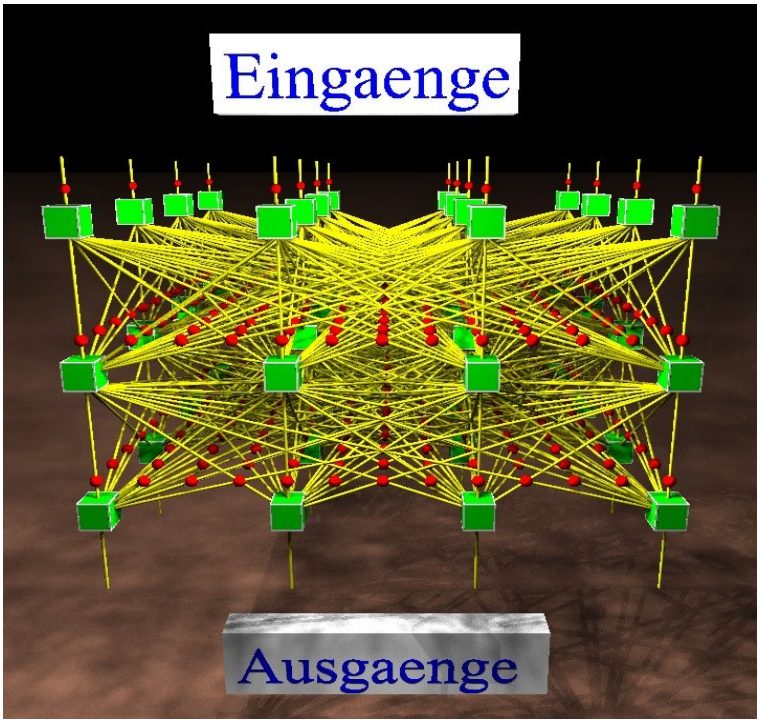

Bild 7:

Beispiel eines neuronales Netzes in Ebenenstruktur mit 16 + 16 + 16 Neuronen

7. Programmieren eines neuronalen Netzes

Nachdem nun eine Struktur gefunden ist taucht die nächste Frage auf: „Wie bringt man ein solches Netzwerk dazu das zu tun was man eigentlich möchte?“ Klar ist, dass der Schlüssel zur individuellen Netzwerkfunktion die Einstellung der Gewichte wij an den einzelnen Eingängen ist. Bei kleinen Netzwerken, genau definierten Aufgabenstellungen und Lösungswegen könnte man diese Einstellung noch „von Hand“ machen. Normalerweise scheidet dieser Weg aber aus weil genau diese Bedingungen (kleines Netzwerk und bekannter Lösungsweg) nicht erfüllt sind. Abhilfe schafft da das was es bei den normalen Gehirnen auch gibt – das trainieren.

7.1 Trainieren eines Netzes

Um ein Netz dazu zu bringen dass es letztendlich das tut was davon erwartet, muss man eine Möglichkeit finden, die Gewichte an den vielen Eingängen korrekt einzustellen. Wie man an dem Bild 7 mit einem noch recht einfachen Netzwerk aus 3x16 Neuronen erkennen kann, lässt sich sehr schnell nicht mehr überblicken welche Wirkung eine einzelnes Gewicht an einer beliebigen Stelle im Netzwerk hat. Es muss also ein Verfahren geben mit dem sich die Gewichte einstellen lassen ohne dass man die Wirkung eines jeden einzelnen kennt. Eines der verwendeten Verfahren ist das der „Back propagation“

Dazu gibt man dem Netzwerk eine Aufgabe vor dessen Ergebnis man schon kennt (z.B. ein Stück Handschrift als Aufgabe und den zugehörigen Klartext als Ergebnis) und schaut wie weit das Ergebnis des Netzwerks von dem erwarteten Ergebnis weg ist. Die Frage wie stark sich Soll- und Istergebnis voneinander unterscheiden lässt sich z.B. mit dem Verfahren der Summe der

Fehlerquadrate ermitteln bei dem bei jedem der Ausgänge das Ist vom Soll abgezogen wird. Diese Differenzen werden einzeln quadriert und alles zusammengezählt. Ist die Summe 0, stimmen Ist und Soll exakt überein, je größer die Zahl ist, desto größer sind die Unterschiede.

Nun verstellt man die Gewichte der Eingänge der letzten Ebene um ganz kleine Werte jeweils in der Richtung, in der die Differenz zwischen Ist und soll kleiner wird. Sind die Gewichte der letzten Ebene eingestellt, wird mit der darüberliegenden in gleicher Weise verfahren. Ist diese Ebene fertig, kommt die nächst höhere dran. Das wiederholt sich bis zur obersten Ebene. Ist die Einstellung der obersten Ebene abgeschlossen, beginnt wieder alles von vorne (genauer bei der letzten Ebene) und das Ganze wiederholt sich. Ist die gewählte Struktur für die vorgesehene Aufgabe geeignet, wird sich der Fehler (also die Differenz zwischen Soll und Ist) langsam zu ganz kleinen Werten (im Idealfall zu null) hin bewegen. An dieser Stelle wird auch klar, warum die Aktivierungsfunktionen der einzelnen Neuronen keine Sprungfunktionen sondern sind sondern die Sprünge durch die E-Funktionen „aufgeweicht“ werden. Bei reinen Sprungfunktionen würde das beschriebene Verfahren nicht funktionieren weil kleine Veränderungen von Gewichten in vielen Fällen keine Auswirkungen auf das Ergebnis des betreffenden Neurons haben.

Ist das „Training“ für eine Aufgabe abgeschlossen, wählt man eine weitere Aufgabe dessen Ergebnis man kennt und verfährt mit diesem neuen Satz von Daten genauso. Ist das Netzwerk für die Aufgabe geeignet, werden sich auch für die weiteren Datensätze Gewichte einstellen, dass sich auch dort Fehler ergeben, die langsam Richtung Null tendieren.

Anstatt der „Kunst“ in den traditionellen Prozessorstrukturen ein Programm zu erstellen besteht hier die Kunst darin, geeignete Trainingsdatensätze auszuwählen.

7.2 Das Problem mit der Vergesslichkeit

Unglücklicherweise kann es auch bei technischen neuronalen Netzen vorkommen, dass das Netzwerk Dinge wieder „vergisst“. Das kann durch eine ungünstige Struktur, ein für die Aufgabe zu kleines Netzwerk oder durch ungünstig ausgewählte Trainingsaufgaben ausgelöst werden. Wenn man in der glücklichen Lage ist, zu erkennen welche der Gewichte für die „Vergesslichkeit“ verantwortlich sind und wie sich diese im Gesamtnetzwerk auswirken, kann man die Veränderungen einzelner Gewichte ab einem vorgebbaren Zeitpunkt sperren. (Zumindest ist das eine Option des schon beschriebenen Simulators).

7.3 Training realer Vorhersagenetzwerke

Ein großer Teil von realen neuronalen Netzwerken dient als Vorhersagesystem. In diesen Systemen ist ein fortlaufendes Training besonders einfach, weil sich jede Vorhersage nach Ablauf der Vorhersagezeit mit dem realen Ereignis vergleichen lässt. Mit jeder Vorhersage entsteht damit ein Datensatz der sich in Zukunft für das fortlaufende Training des Netzwerks einsetzen lässt.

8. Das Projekt „CW-Dekodierung“

Nachdem es sich bei CW um eine Art eindimensionale Handschrift handelt, sollte ein CW-Dekoder die Testapplikation für ein solches neuronales Netz werden. Vor dem Start musste aber zuerst ein Problem aus dem Weg geräumt werden: Beim CW treffen die Informationen zeitlich nacheinander

ein (Punkte, Striche), das Netzwerk in der beschriebenen Ebenenstruktur kann sich aber nichts „merken“ - es könnte damit keine zeitlich hintereinanderliegenden Ereignisse miteinander verknüpfen. Das Problem lässt sich mit einem Schieberegister lösen, in das das CW-Signal hineingetaktet wird. An den parallelen Ausgängen des Registers lassen sich dann die zeitlich nacheinander eingetroffenen Ereignisse (Punkte, Striche, Pausen) parallel abnehmen. Das Schieberegister übernimmt damit die dem Netz fehlende Speicherfunktion.

Bei einer genaueren Untersuchung wie sich die Gewichte im Netz wohl einstellen werden zeigte sich, dass sich die Struktur eines Korrelators für die Erklärung der resultierenden Werte verwenden lässt.

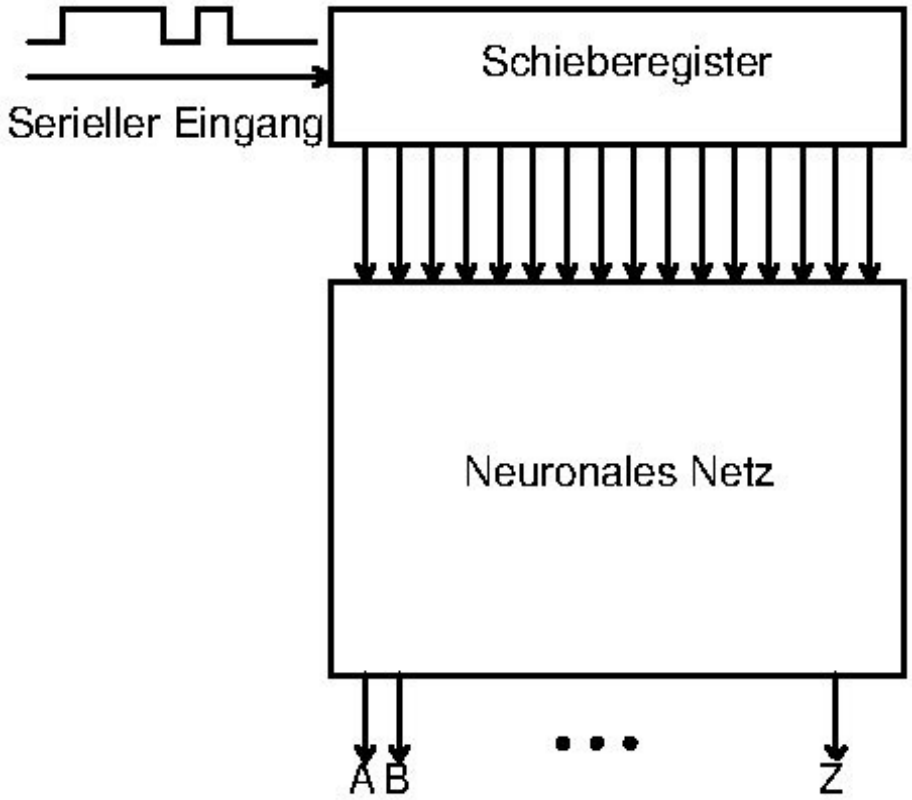

Bild 8:

Das neuronale Netz im Dekoder. Das Schieberegister übernimmt die Speicherfunktion.

8.1 Mustererkennung mit Hilfe eines Korrelators

Um in technischen Systemen Muster zu vergleichen, wird gerne ein Korrelator verwendet. Dabei werden die Abtastwerte eines Testmusters mit denen des zu untersuchenden Musters multipliziert und Einzelergebnisse zusammengezählt. Je höher das Ergebnis, desto besser ist die Übereinstimmung der beiden untersuchten Muster. Für jedes Testmuster gibt es ein individuelles Maximum das die vollständige Übereinstimmung anzeigt.

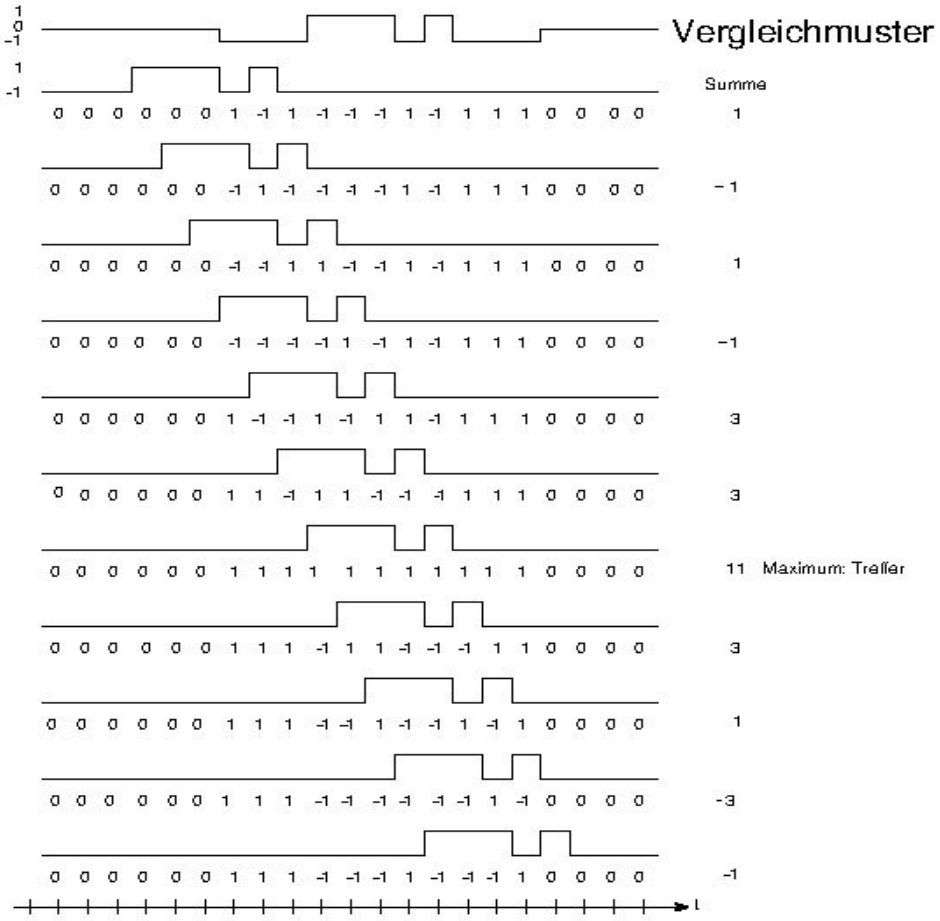

Bild 9:

Ergebnisse eines Korrelators beim „durchschieben“ eines Testmusters.

Sieht man die Signale an den parallelen Ausgängen des Schieberegisters des oben beschriebenen Dekoders als zu untersuchendes Muster an, dann müsste man jedes der Signale der Schieberegisterausgänge mit einem dem Testmuster entsprechenden Wert multiplizieren, die Ergebnisse zusammenzählen und vergleichen ob das Ergebnis dem für das individuelle Testmuster gültigen Maximalwert entspricht. Interpretiert man die Gewichte der Eingänge eines Neurons als Testmuster für ein bestimmtes Zeichen, kann man die Korrelatorfunktion für das betroffene Zeichen auch als Neuron für genau dieses Zeichen verstehen. In diesem Fall sind also neben der Netzwerkstruktur sogar die Gewichte an den Eingängen schon bekannt.

8.2 Das erste Testnetz

Das erste Testnetz bestand deshalb vorerst nur aus einem Neuron das mit Hilfe der Funktionen des Simulators auf den Buchstaben „A“ trainiert werden sollte. Das Training bestand daraus, alle Muster des Alphabets, der Ziffern und Zeichen vorzugeben und nur beim Zeichen „A“ eine „1“ als korrektes Ergebnis zuzulassen. Wie erhofft stellten sich beim Training tatsächlich die Werte ein, die man auch beim Korrelator erwartet hätte. Nachdem ein Neuron für genau ein Zeichen zuständig ist, besteht das gesamte Netz also aus einer Zeile von 49 Neuronen (26 Buchstaben, 3 Umlaute, 10 Ziffern und 10 Satzzeichen inklusive Wortabstand).

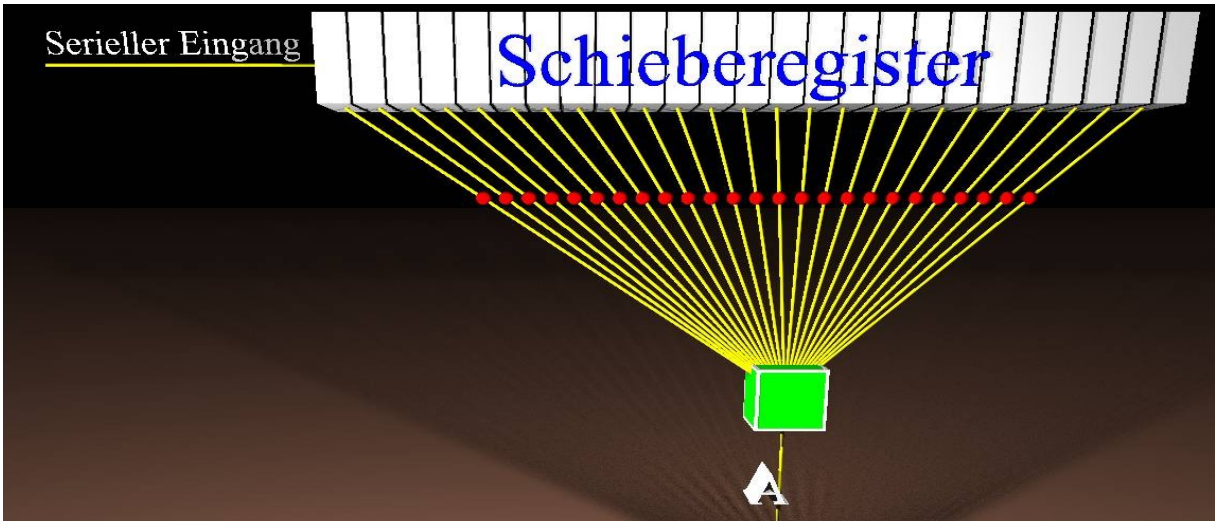

Bild 10:

Dekodierung eines Buchstabens mit Hilfe eines Neurons

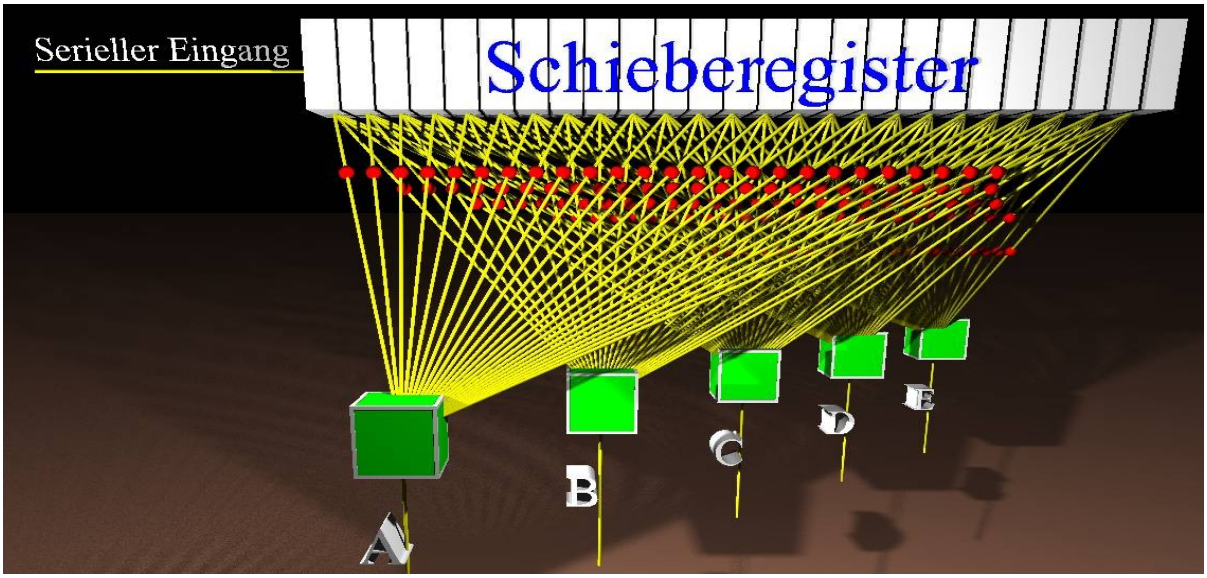

Bild 11:

Erweiterung des Netzwerks für die Dekodierung der ersten 5 Buchstaben

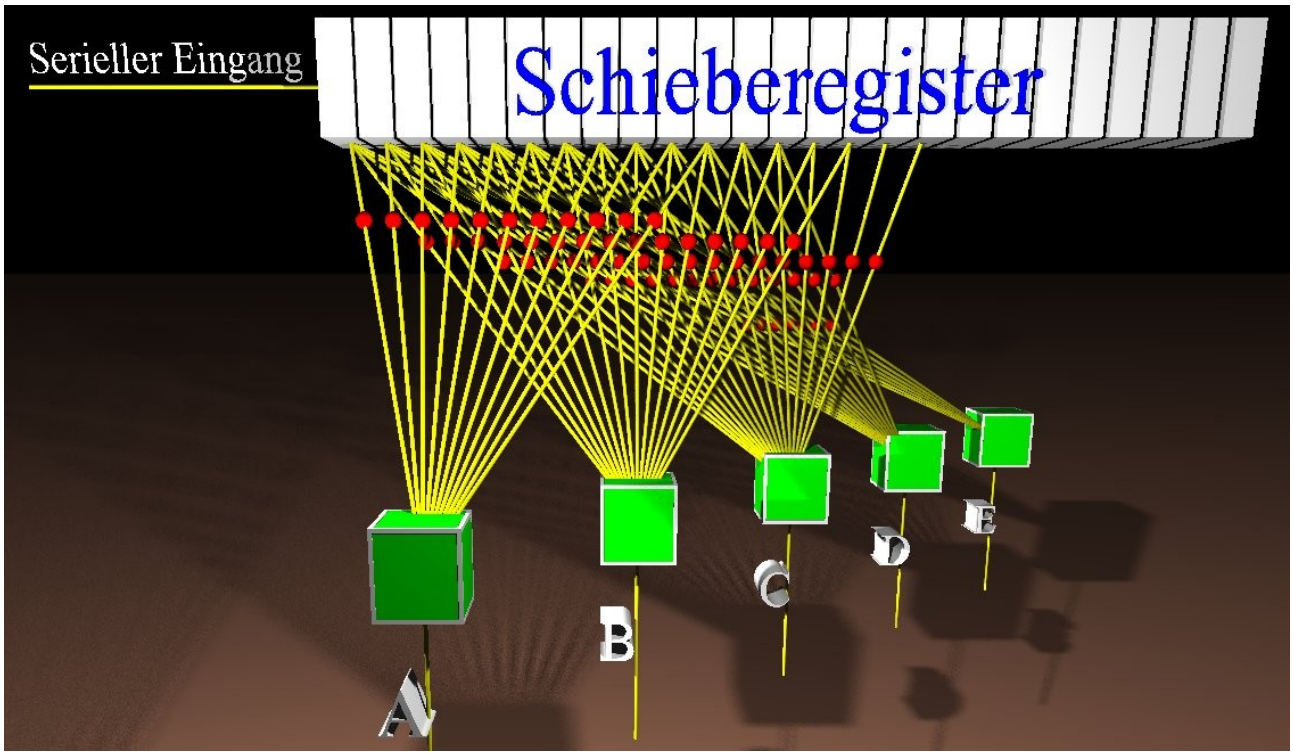

Bild 12:

Vereinfachung des Netzwerks durch entfernen von Verbindungen mit dem Gewicht „0“

8.3 Die Grenzen des ersten Testnetzes

Sieht man sich den Mustervergleich genau an, so fällt auf, dass man mit dieser Struktur genau so weit ist, wie bei dem allerersten CW-Dekoder:

- die Schrift muss eine exakte Maschinenschrift sein,
- die Geschwindigkeit muss genau der Taktrate des Schieberegisters entsprechen
- Es darf kein QRM geben (jeder Bitfehler führt zu einer fehlerhaften Dekodierung)

Die Toleranz des Netzwerks gegenüber QRM oder Handschriften lässt sich verhältnismäßig einfach erhöhen: Anstatt einem Abtastwert pro Punkt kann man 10 oder 20 Abtastwerte erzeugen (den Takt des Schieberegisters also um den Faktor 10 oder 20 erhöhen). Geht man von Faktor 10 aus, würde erst dann das falsche Zeichen dekodiert wenn 5 der 10 Abtastwerte durch QRM verfälscht wären. Außerdem lässt sich das Netzwerk dann auch auf bestimmte Handschriften trainieren. Entsprechend dem anderen Punkt/Strichverhältnis würde sich andere Gewichte an den Eingängen einstellen. Bei der Erhöhung der Abtastrate vervielfacht sich dann natürlich auch die Anzahl der Eingänge zu der zugehörigen Gewichte entsprechend.

Unabhängig davon wieviele Eingänge pro Neuron auch verwendet werden, die Toleranz gegenüber „falschen“ Geschwindigkeiten lässt sich mit einer einzeiligen Netzstruktur nicht verbessern.

8.4 Dekodierung bei verschiedenen Geschwindigkeiten

Der einzige Weg auch verschiedene Geschwindigkeiten korrekt dekodieren zu können ist,

zusätzliche Zeilen von Neuronen zu schaffen, für jede Geschwindigkeit eine eigene Zeile. Geht man davon aus, dass die Dekodierung bei einem Geschwindigkeitsunterschied von etwa 5% nicht mehr korrekt funktioniert bräuchte man für einen Geschwindigkeitsbereich von 30 Zeichen pro Minute bis 200 Zeichen pro Minute 40 Zeilen. Die Neuronenebene hätte also 49x40=1960 Neuronen. Diese Erweiterung hat zur Folge, dass es jetzt für jedes der 49 Zeichen 40 Ausgänge gibt. Bei einer bestimmten Geschwindigkeit werden für ein Zeichen normalerweise mehrere unterschiedliche Ausgänge gesetzt sein von denen immer nur einer korrekt ist. (Beispiel das „E“ bei Tempo 30 wird bei Tempo 90 als „T“ erkannt) . Nachdem man für eine sinnvolle Dekodierung jeweils nur ein Zeichen am Ausgang sehen möchte (nämlich das, das der aktuellen Geschwindigkeit entspricht) muss man eine weitere Ebene von 1960 Neuronen hinzufügen in denen das richtige Zeichen ausgewählt wird. Damit tut sich aber sofort das nächste Problem auf: „Wie lässt sich die Geschwindigkeit der aktuellen CW-Signale erkennen“?

Nachdem sich die Erkennung der Geschwindigkeit vollkommen von der Dekodierung der Zeichen unterscheidet macht es Sinn dafür ein Teilnetz zu erzeugen und das unabhängig von dem anderen zu trainieren und dessen Gewichte danach einzufrieren.

8.5 Ermittlung der Geschwindigkeiten

Wie bei der Dekodierung der Einzelzeichen selbst stellt sich auch für die Ermittlung der Geschwindigkeit die Frage nach einem dafür optimalen Netzwerk. Auch hier gibt es eine Struktur die sich auch als neuronales Netz aufbauen lässt: Die Struktur des Algorithmus der schnellen Fourier Transformation (FFT). Diese Struktur besteht aus einer Reihe von „Butterfly“-Operationen. Eine solche Butterfly-Operation hat zwei Eingänge (einer mit dem Gewicht 1, der andere mit einem Gewicht PI/N) und zwei Ausgänge (an einem steht die Summe, am anderen die Differenz der Eingänge an). Damit lässt sich jede dieser Butterfly-Operationen mit zwei Neuronen abbilden.

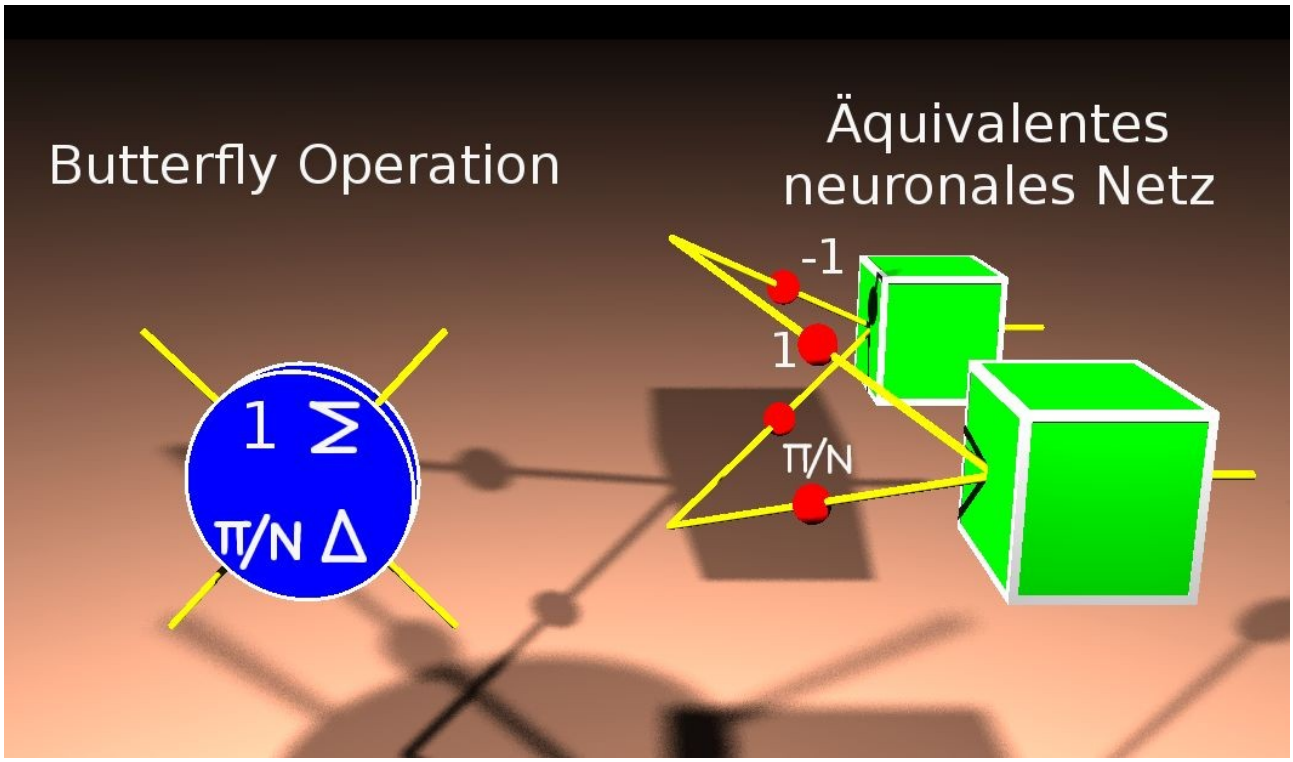

Bild 13

Butterfly Operation und das dazu äquivalente neuronale Netz

Für eine FFT-Struktur braucht man für N Abtastpunkte ld(N) Zeilen von je N/2 Butterfly Operationen. Nachdem für jede Butterfly-Operation zwei Neuronen nötig sind, sind N Neuronen in ld(N) Ebenen nötig. In Zahlen ausgedrückt: Für 512 Abtastpunkte sind für die Ermittlung der Geschwindigkeit 8 Ebenen zu je 512 Neuronen also 4096 Neuronen erforderlich. So hoch die Zahl der Neuronen in diesem Fall auch ist, wird in dem trainierten Netzwerk jedes Neuron nur zwei Verbindungen zu Neuronen der tiefer liegenden Ebene haben (d.h. Von den 1835008 internen Verbindungen werden nur 7168 ungleich Null sein).

9. Abschätzung der Netzwerkgröße

Vor der Planung einer wirklichen Realisierung eines neuronalen CW-Dekoders macht es Sinn über den Dafür nötigen Umfang und die damit erreichbaren Werte nachzudenken.

9.1 Anzahl der Abtastpunkte

Zerlegt man die zu dekodierenden Zeichen in Einzelschritte kommt man zu einem Maximum von 25 Schritten. (Beispiel: Die Ziffer NULL besteht aus 3 Pausenschritten vor dem Zeichen, 3 Pausenschritten nach dem Zeichen, 5 x 3 Schritten für die Striche und 4 Zwischenräumen). Zerlegt man einen Schritt in 20 Teilschritte kommt man in Summe auf 500 Schritte. Die Aufteilung einer Punkt- oder Pausenlänge in 20 Teilschritte mag sehr groß erscheinen, dieses Verhältnis gilt leider nur für das langsamste CW-Tempo. Bei 60 Zeichen/Minute ist das Verhältnis nur noch 1/10, bei 200 nur noch 1/3. Die Ausblendung von QRM ist bei dieser Geschwindigkeit dann kaum mehr möglich, Handschriften mit dem Tempo sind aber eher unwahrscheinlich.

Ergebnis: Das Schieberegister braucht mindestens eine Länge von 500 Stellen.

9.2 Anzahl der Neuronen im Dekodernetzwerk

Die Zahl wurde schon einmal berechnet. Staffelt man die möglichen Dekodiergeschwindigkeiten zwischen 30 Zeichen/Minute und 200 Zeichen/Minute kommt man auf 40 Geschwindigkeitsbereiche. Das gibt für die oberste Ebene 40 x 49 = 1960 Neuronen.

Die zweite Ebene sollte dazu dienen aus den in der ersten Ebene dekodierten Zeichen dasjenige zu selektieren das dem augenblicklichen CW-Tempo entspricht. Damit ist für jedes Neuron der ersten Ebene eines in der zweiten Ebene nötig, in Summe also ebenfalls 1960 Neuronen.

Schließlich müssen die Ausgänge der 1960 Neuronen der zweiten Ebene in einer dritten auf die möglichen 49 Zeichen zusammengefasst werden wozu 49 Neuronen nötig sind.

9.3 Anzahl der Verbindungen im Dekodernetzwerk.

Geht man von der Maximalzahl möglicher Verbindungen aus, so kommt man auf der Eingangsseite zwischen Schieberegister und den Neuronen der ersten Ebene auf 500 x 1960 = 980000 Verbindungen (und damit einzustellende Gewichte!).

Eine volle Verbindungszahl zwischen erster und zweiter Ebene würde 1960 x 1960 = 3841600 Verbindungen und damit einzustellende Gewichte erfordern. Dazu kommen noch die Verbindungen zu dem Netzwerk das das CW-Tempo ermittelt: 512 x 1960 = 1003520 Verbindungen was zu einer Gesamtsumme von 4845120 Verbindungen führt.

Zwischen der Ebene 2 und der Einzelzeile der dritten Ebene sind es schließlich 1960 Verbindungen.

Nach dem Training wird es allerdings deutlich einfacher aussehen:

Bei den Verbindungen vom Schieberegister zu den Neuronen der ersten Ebene sind natürlich nur die relevant die zu einem möglichen Zeichen gehören. So besteht das Zeichen „A“ nur aus 11 Schritten, die Schritte 12 bis 25 werden damit das Gewicht Null bekommen und können deshalb entfallen. Bei 20 Teilschritten pro Punkt- oder Pausenlänge werden beim Zeichen A und der niedrigsten Geschwindigkeit nur 220 der 500 Verbindungen aktiv sein, bei der doppelten Geschwindigkeit – also 60 Zeichen pro Minute nur noch 110 der 500. Zählt man die für die niedrigste Geschwindigkeit erforderliche Anzahl von Verbindungen aus so kommt man auf 16460 von den möglichen 24500, bei 60 Zeichen/Minute sind es dann nur noch 8230 von 24500 und 200 Zeichen/Minute 2429 aktive Verbindungen.

Am deutlichsten ist die Vereinfachung zwischen Ebene 1 und 2. Dort gibt es zwischen den Dekoderneuronen jeweils nur eine Verbindung (in Summe also 1960) und 13 aktive Verbindungen zwischen jedem Dekoderneuron und den Ausgängen des Geschwindigkeitsnetzwerks (512 / 40 * 1960) was dann in Summe 27440 Verbindungen (und Gewichte) ausmacht.

Da alle Zeichen in allen Geschwindigkeiten möglich sind, werden auch alle möglichen 49 x 40 = 1960 Verbindungen aktiv sein.

9.4 Anzahl Neuronen und Verbindungen im FFT-Netzwerk

Anzahl von Neuronen und die zugehörige Anzahl von Verbindungen wurde schon im Punkt 8.5 beschreiben. Demnach sind 4096 Neuronen und 7168 interne Verbindungen, 500 Eingänge und 13 x 1960 = 25480 Ausgänge die in das Dekodernetzwerk laufen erforderlich.

10. Zusammenfassung

Offensichtlich ist es tatsächlich möglich, mit Hilfe eines neuronalen Netzes einen guten CW-Dekoder zu bauen. Da sich die dafür erforderlichen Teilnetze aus anderen Strukturen (Korrelator, FFT) herleiten lassen ist davon auszugehen, das sich ein so aufgebautes Netz erfolgreich trainieren lässt. Sieht man sich die unter den oben genannten Randbedingungen erforderlichen Neuronen und Verbindungen an

Summe Neuronen: 8062

Summe Verbindungen vor dem Training: 6780600

Summe Verbindungen nach dem Training: ca. 366350

so zeigt sich, dass der Bau eines CW-Dekoders mit neuronalen Netzen zumindest mit den untersuchten Strukturen zu einem gigantischen Aufwand führt. Trotzdem bleibt eine Realisierung z.B. in einem geeigneten FPGA denkbar. Für mich persönlich bleibt die Variante eine CW-Dekoder mit Hilfe eines neuronalen Netzes zu realisieren damit weiterhin „auf Eis“.

Im Gegensatz zu den sonst üblichen neuronalen Netzen ist im Fall des hier beschriebenen CW-Dekoders die Aufgabe jedes einzelnen Neurons genau bekannt (Dekodierung eines Einzelzeichens, Auswahl der relevanten Geschwindigkeitszeile, Ermittlung der Geschwindigkeit mit einer FFT). Damit lässt sich der Dekoder mit weit geringerem Aufwand und ähnlichen Ergebnissen natürlich auch „klassisch“ realisieren.

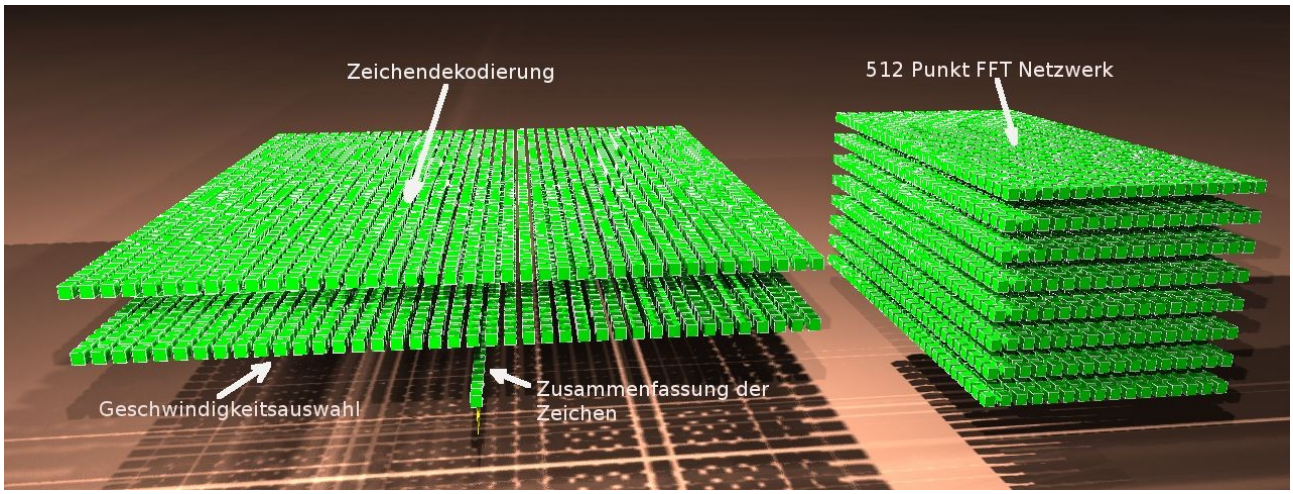

Bild 14:

Alle für den CW-Dekoder erforderlichen 8062 Neuronen. Aus Gründen der Übersichtlichkeit wurden die 6780600 gewichteten Verbindungen hier nicht dargestellt.