Genetic Algorithm Neural Network Architecture Synthesizer*

Merten Joost und Wolfram Schiffmann
Universität Koblenz–Landau, Institut für Physik
Rheinau 1, D-56075 Koblenz, Tel.: 0261–9119–358, FAX 354
e-mail: evol | schiff @uni-koblenz.de
http://www.uni-koblenz.de/∼schiff

Abstract: In this paper a genetic algorithm will be introduced which automatically adapts the architecture of a multilayer feedforward network to a given problem. Instead of coding solely the network topology we use a new approach which models the spatial arrangement (architecture) of the processing units. In this way the search space will be reduced and we get more biological plausibe codings which support the application of genetic operators, e.g. crossover. The breeded architectures can be displayed and compared more easily. We used two applictions to evaluate the GANNAS algorithm: contiguity and handwritten digit recognition. For the contiguity problem, GANNAS generated the *human solution* with perfect generalization behaviour. The best network for the digit recognition problem performs similar to the best manually optimized network design.

Keywords: topology vs. architecture optimization, automatic network design, genetic algorithms

1 Einleitung

Mehrschichtige feedforward Netze (MFN) eignen sich sowohl für die Funktionsapproximation als auch zur Lösung von Klassifikationsaufgaben. Für optimale Ergebnisse ist es jedoch nötig, zu einem gegebenen Problem eine optimale Netzwerktopologie vorzugeben. Diese Topologie muß einerseits in der Lage sein, die Trainingsmuster ausreichend genau abzubilden, andererseits soll aber gleichzeitig auch auf der Testmenge ein möglichst geringer Fehler erreicht werden. Das erstgenannte Ziel ist am besten mit einer großen Zahl von internen Knoten zu erreichen, während das letztgenannte Ziel mit möglichst wenig Knoten (und damit mit wenig Verbindungsgewichten) erreicht wird. Die Topologie–Optimierung (TO) muß hier einen geeigneten Kompromiß finden. Im allgemeinen wird man bei manueller TO nur die Zahl der internen Knoten und deren Verteilung über eine bis maximal drei interne Schichten optimieren können. Die einzelnen Schichten werden dann vollständig vernetzt (fully interconnected). Dabei muß noch entschieden werden, ob nur benachbarte Schichten verbunden werden oder ob auch schichtüberspannende Verbindungen (shortcuts) erlaubt sind.

Da die manuelle Suche nach einer geeigneten Topologie sehr mühsam ist, sollte man Werkzeuge zur automatischen TO entwickeln. Die TO ist ein kombinatorisches Optimierungsproblem mit verrauschter Gütefunktion (wegen der zufälligen Initialisierung der Gewichte). Aufgrund der hohen Komplexität des Problems können *keine enumerativen* Verfahren eingesetzt werden. Prinzipiell sind *destruktive* und *konstruktive* Verfahren zur TO zu unterscheiden. Destruktive Verfahren gehen von einer ausreichend großen Starttopologie aus und entfernen redundante Netzkomponenten. Einen Überblick über diese sogenannten *Pruning* Algorithmen wird in [4]

*Das GANNAS – Projekt wurde von der Deutschen Forschungsgemeinschaft gefördert (Schi 304/1-2).

gegeben. Bei den konstruktiven Verfahren beginnt man mit den Schnittstellenknoten und fügt solange interne Netzkomponenten hinzu, bis der Trainingsfehler hinreichend klein geworden ist. Ein typisches Beispiel ist der *Cascade–Correlation* Algorithmus [2]. Genetische Algorithmen (GAs) beginnen mit einer Population zufällig strukturierter Netztopologien, die im Laufe der Evolution durch Operatoren modifiziert werden. Dabei werden sowohl Netzkomponenten hinzugefügt als auch entfernt. Somit stellen GAs eine Mischform aus konstruktiven und destruktiven Verfahren zur TO dar.

GAs verwenden *genotypische* Netzcodierungen, die vor dem Training zu Phenotypen decodiert werden müssen. Die genotypische Netzcodierung ist biologisch plausibler als eine phenotypische Repräsentation, da auch bei Lebewesen nicht jedes Neuron und jede Synapse genetisch festgelegt ist. Die Netzcodierung bei GAs muß eine *lineare* Form (string) haben, damit der crossover-Operator angewandt werden kann. Der *crossover* – Operator kann Nachkommen produzieren, die von ihren Vorfahren radikal verschieden sind. Dieser Effekt ist nicht immer wünschenswert und führt daher auch zu Schwierigkeiten bei der Anwendung dieses Operators (competing conventions problem [5]).

Die Selektion bei GAs erfolgt zufallsgesteuert und ist proportional zur Güte der Topologien (roulette wheel selection). Bei verteilten Implementierungen von GAs wird häufig die tournament selection benutzt, bei der jeweils die besten Individuen aus räumlich entfernten Teilpopulationen selektiert und anschließend miteinander kombiniert werden. Es gibt eine sehr große Vielfalt von GA–orientierter Ansätze, die in der sogenannten *EDNA Bibiliography* zusammengestellt sind [1].

2 Architektur–Codierung

Die Codierung einer Netzwerkarchitektur soll eine kompakte Darstellung strukturierter Netzwerke erlauben und dabei die räumliche Anordnung der Neuronen berücksichtigen. Man geht davon aus, daß auch natürliche neuronale Netze einen geschichteten Aufbau haben. Alle Neuronen einer solchen Schicht haben dabei prinzipiell eine ähnliche Funktionalität, arbeiten jedoch auf räumlich getrennten Bereichen des Eingaberaumes. Ein Beispiel hierfür ist der visuelle Cortex höherer Primaten, der bereits relativ gut erforscht ist [3]. Wir haben für unsere Darstellung der Netzwerkarchitektur eine parametrisierte und layerorientierte Codierung verwendet. Unter einem Layer soll hierbei ein zweidimensionales Feld von Neuronen verstanden werden. Pro Layer müssen Höhe und Breite spezifiziert werden. Schließlich werden je Layer die vorhandenen synaptischen Verbindungen angegeben. Für jede Verbindung müssen die Nummer des Layers zu dem die Verbindungen führen, Höhe und Breite des *rezeptiven* Feldes und der Typ der Verbindung angegeben werden. Dabei werden vier verschiedene Typen unterschieden:

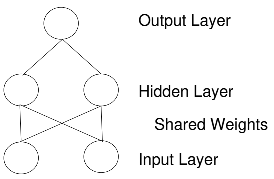
1. Der Typ **FULLY** zeigt an, daß eine vollständige Vernetzung zwischen den beiden Schichten vorgenommen werden soll. Die Abmessungen des rezeptiven Feldes werden bei diesem Typ ignoriert.

2. Beim Typ **NORMAL** wird jedes Neuron über ein rezeptives Feld der angegebenen Größe mit dem anderen Layer verschaltet.

3. Bei Verwendung des Typs **SHARED** wird wie bei NORMAL eine rezeptive Verschaltung hergestellt. Es werden jedoch miteinander gekoppelte Gewichte (*shared weights*) verwendet, d.h alle Neuronen verwenden dieselben synaptischen Werte zu den rezeptiven Feldern

der vorgelagerten Schicht. Die Gewichte können sich nur noch innerhalb des rezeptiven Feldes unterscheiden. Ferner wird ein gemeinsames shared weight für alle Biaswerte verwendet.

4. Der Typ **FULLY_SHARED** arbeitet wie NORMAL benutzt jedoch ein einziges shared weight für alle Synapsen und einen weiteres shared weight für alle Biaswerte.




```
NLAYER: 3
LAYER2: 1 1
    NCONNECTIONS: 1
    1: 1 1 FULLY
LAYER1: 2 1
    NCONNECTIONS: 1
    0: 2 1 SHARED
LAYER0: 2 1
    NCONNECTIONS: 0
```

Abbildung 1: Architektur und Beschreibung eines einfaches Netz

3 Genetische Operatoren

Als genetische Codierung der Netzarchitekturen wurde das oben beschriebene Format gewählt, da auf diesem besonders einfach die genetischen Operatoren implementiert werden konnten. Die verwendeten genetischen Operatoren sind in drei Bereiche eingeteilt:

1. Mutationsoperatoren
2. Crossoveroperatoren
3. Selektionsschema

Im folgenden sollen diese Operatoren kurz beschrieben werden:

Mutation: Der Mutationsoperator bewirkt ein zufälliges Verändern der codierten Netzarchitekturen. Da eine Architektur auf unterschiedliche Weise verändert werden kann, werden entsprechende Mutationsparameter benötigt. Diese lassen sich in zwei Teilbereiche unterteilen. Es existieren zum einen Parameter, die auf der Ebene der Layer operieren und zum anderen Parameter, die die Verbindungstrukturen verändern.

- **Mutationsparameter auf der Layerebene**
  - **layer_insert:** Beschreibt die Wahrscheinlichkeit, an einer bestimmten Postion in der Layerliste einen neuen Layer einzufügen. Dieser neue Layer wird mit der Wahrscheinlichkeit **connect_layer** mit den vorhandenen Schichten der Architektur verbunden.
  - **max_dim_layer:** Beschreibt die maximale Ausdehnung eines neuen Layers sowohl in x als auch in y Richtung. Die Größe des neuen Layers bestimmt sich zufällig aus $0 < x \leq \mathbf{max\_dim\_layer}$ und $0 < y \leq \mathbf{max\_dim\_layer}$.

- **layer_delete:** Ist ein Wahrscheinlichkeitsmaß, einen bestimmten Layer in der Layerliste zu löschen. Dabei werden alle Verbindungsstrukturen von und zu diesem Layer ebenfalls entfernt.
- **layer_change:** Beschreibt die Wahrscheinlichkeit, die Größe eines Layers zu verändern. Die Veränderung erfolgt dabei zufällig inkrementell oder dekrementell. Die obere Grenze **max_dim_layer** wird dabei beachtet.

- **Mutationsparameter auf der Verbindungsebene**
  - **connect_layer:** Ist ein Wahrscheinlichkeitsmaß, einen neu eingefügten Layer zu verbinden. Form und Art der Verbindung wird vom Parameter **max_dim_field** bestimmt.
  - **max_dim_field:** Beschreibt die maximale Ausdehnung einer neuen Verbindung sowohl in x als auch in y Richtung. Die Größe dieser Verbindung bestimmt sich zufällig aus $0 \leq x \leq \texttt{max\_dim\_layer}$ und $0 \leq y \leq \texttt{max\_dim\_layer}$. Der Typ der neuen Verbindung wird ebenfalls zufällig gewählt.
  - **field_insert:** Beschreibt die Wahrscheinlichkeit, zwei bislang nicht verbundene Layer miteinander zu verbinden. Form und Art der Verbindung wird dabei ebenfalls vom Parameter **max_dim_field** bestimmt.
  - **field_delete:** Ist ein Wahrscheinlichkeitsmaß, eine existierende Verbindungsstruktur zu löschen.
  - **field_change:** Beschreibt die Wahrscheinlichkeit, die Größe eines rezeptiven Feldes zu verändern. Die Veränderung erfolgt dabei zufällig inkrementell oder dekrementell. Die obere Grenze **max_dim_field** wird dabei beachtet.
  - **couple_change:** Beschreibt die Wahrscheinlichkeit, den Typ einer Verbindung zu verändern. Der neue Typ wird dabei zufällig gewählt.

Crossover: Die folgenden Crossoverparameter steuern das Rekombinieren (Kreuzen) zweier Architekturen:

- **anzahl:** Bestimmt die Anzahl der Kreuzungspunkte. Die Lage der Kreuzungspunkte innerhalb der Architekturbeschreibungen wird zufällig ermittelt.
- **wkeit:** Beschreibt die Wahrscheinlichkeit, an einem Kreuzungspunkt ein Crossover durchzuführen. Wird einer dieser Parameter auf Null gesetzt, so erfolgt kein Crossover.

Selektion: Als Selektionsschema wurde das bekannte *Roulette*–Auswahlverfahren implementiert. Dabei werden auf einem Rouletterad Bereiche für alle Mitglieder der aktuellen Population reserviert. Der Bereich eines Mitgliedes ist dabei umso größer, je höher seine Güte *(Fitness)* ist. Somit besitzen Mitglieder hoher Güte eine höhere Chance selektiert zu werden. Das Gütemaß wird dabei in zwei Stufen ermittelt. Zunächst wird anhand des Fehlers auf der Teststichprobe[†] selektiert. Wenn alle in diesem Schritt selektieren Individuen eine bestimmte Mindestgüte besitzen, wird zusätzlich anhand der Anzahl der freien Parameter selektiert. Ziel ist dabei,

[†]Die Benutzung einer Teststichprobe als Gütekriterium vermeidet die Auswahl übertrainierter Netze. Diese besitzen im allgemeinen kein gutes Generalisierungverhalten.

bei gleichbleibender Güte auf der Teststichprobe, Architekturen mit minimaler Komplexität auszuwählen.

Aus Gründen der Effizienz wurde auf getrennte Eltern/Nachkommen – Populationen verzichtet. Es existiert lediglich eine Population, auf der die genetische Suche operiert (steady state selection). Ein neues Individuum wird dann in die Population aufgenommen, wenn es eine höhere Güte aufweist als das bis dahin schlechteste Individuum. Dieses wird dann durch die neue Architektur ersetzt.

4 Simulationsergebnisse

Zur Erprobung des oben beschriebenen genetischen Algorithmus wurden zwei unterschiedliche Aufgabenstellungen benutzt. Zum einen ein synthetisches binäres Klassifikationsproblem, das sogenannte *Contiguity–Problem*. Es ist allgemein bekannt, daß der Erfolg beim Lösen dieser Aufgabenstellung mit MFNs entscheidend von der Netztopologie abhängt [6]. Außerdem existiert für dieses Problem eine plausible „menschliche" Lösung. Zum anderen wurde unser Verfahren auf die Klassifikation handgeschriebener Ziffern angewandt. Hier stand uns eine sehr große Ziffernstichprobe mit einem Umfang von 20000 Ziffern zur Verfügung.

4.1 Das Contiguity–Problem

Das Contiguity–Problem eignet sich besonders gut, um das Generalisierungsvermögen von MFNs zu analysieren. Das Netz soll bei einem einem m–dimensionalen binären Eingangsvektor **p**erkennen, ob mindestens k_0 Blöcke mit zusammenhängenden Einsen enthalten sind:

$$\mathbf{p} = (p_0, p_1, \ldots, p_{m-1}) \quad \forall \quad p_i \in \{0, 1\} \quad \text{für} \quad 0 \leq i \leq m$$

$$o = \begin{cases} 1 & : \text{ falls } k_0 \text{ oder mehr Eins-Blöcke} \\ 0 & : \text{ sonst} \end{cases}$$

Man kann für die Ränder $p_{-1} = p_m = 0$ annehmen oder man kann das Muster periodisch fortsetzen, d.h. der erste Eingang p_0 ist zum letzten Eingang p_{m-1} benachbart. Als Problemstellung wurde eine Stichprobe *ohne* zyklische Randbedingungen gewählt. Der binäre Eingangsvektor besitzt die Breite $m = 10$, die untere Grenze für die Anzahl der Blöcke beträgt $k_0 = 4$. Die gesamte Stichprobe besteht aus $2^{10} = 1024$ Muster, die in eine Lern– und eine Teststichprobe aufgeteilt wurden. Die Lernstichprobe besteht dabei aus 507, die Teststichprobe aus 517 Mustern.

Der genetische Algorithmus erzeugte während seines Laufes 3773 verschiedene Architekturen. Bereits ab der 634ten Architektur bestand die Population nur aus Architekturen, die die geforderte Güte (Fehler ≤ 1.0) auf der Teststichprobe erfüllten (siehe Abbildung 2). Bis zu diesem Zeitpunkt nahm die Anzahl der freien Parameter, und damit auch die Größe der Architekturen ständig zu. Da nun neben dem Fehler als Gütemaß auch die Zahl der Parameter optimiert wurde, konnten im weiteren Verlauf Architekturen erzeugt werden, die bei gleichbleibendem Fehler weniger freie Parameter aufweisen. Ab ca. der 1000ten Architektur stagniert die Entwicklung. Der evolutionär entwickelte Architekturtyp besitzt im wesentlichen folgende Merkmale:

1. Die Teststichprobe wird in vollem Maße korrekt klassifiziert.

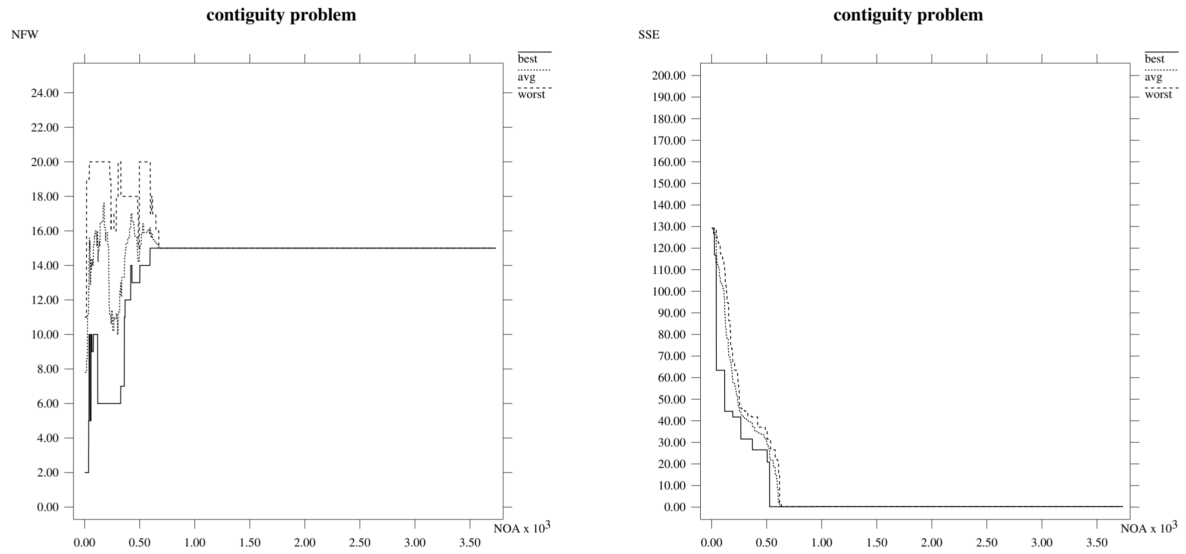


Abbildung 2: Der Fehlerverlauf während der genetischen Entwicklung im Architekturen–Pool (NOA = number of architectures), sowie die Entwicklung der freien Parameter (NFW = number of free weights), dargestellt sind die Parameter der jeweils besten und schlechtesten Architektur, sowie der Durchschnitt der Population.

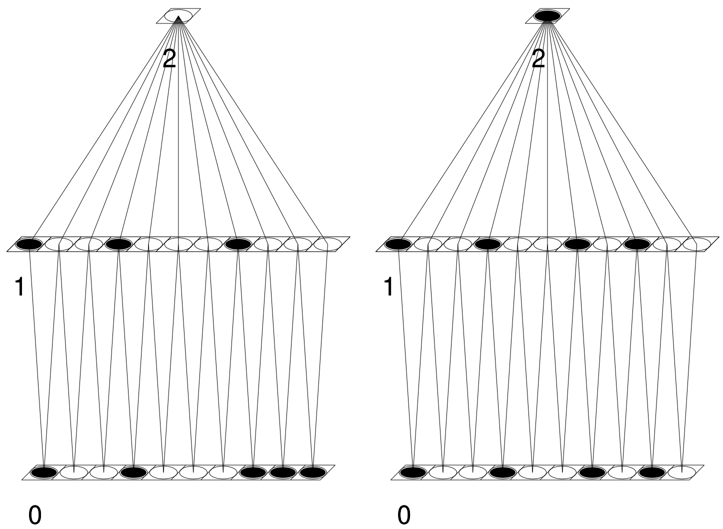


Abbildung 3: Die Topologie der besten Architektur. Weiterhin wird die Aktivierung der Neuronen anhand zweier Musterbeispiele aus der Teststichprobe gezeigt. Die Neuronen der mittleren Schicht erkennen $0 - 1$ Übergänge. Das Ausgangsneuron ist dann aktiviert, sofern mehr als 4 Neuronen der mittleren Schicht aktiv sind.

2. Die Architektur besteht aus 3 Layern.
3. Die mittlere Schicht ist dabei mit dem Eingangslayer mit rezeptive Feldern der Breite 2 verbunden, deren Gewichte shared sind.
4. Der Ausgangslayer ist mit der mittleren Schicht vollständig verbunden. Die Gewichte sind vom Typ NORMAL.
5. Ausgangslayer und Eingangslayer sind nicht direkt miteinander verbunden.
6. Die Anzahl der freien Parameter ist 15, die Anzahl der verwendeten Neuron und Verbindungsgweichte beträgt 43.

Die beste Architektur, die der genetische Algorithmus entwickelte ist in Abbildung 3 dargestellt. Man erkennt die rezeptiven Felder der Breite 2, die auf Kantendetektoren schließen lassen. Weiterhin zeigt die Abbildung die Aktivierung der Neuronen anhand zweier ausgewählter Testmuster. Es wird deutlich, daß genau die Neuronen aktiv werden, die in ihrem rezeptiven Feld einen $0-1$ Übergang besitzen.

4.2 Ziffernerkennung

Zur Erprobung wurde ein Datensatz von handgeschriebenen Ziffern verwendet. Dieser wurde von der Firma Daimler Benz zur Verfügung gestellt und besteht aus insgesamt 20000 Exemplaren der Ziffern 0 bis 9. Die Ziffern liegen segmentiert in einer Größe von 16x16 Pixeln mit einer Auflösung von 256 Graustufen vor. Als Startnetz für den genetischen Algorithmus wurde eine Architektur augewählt, die nur aus den benötigten Ein– und Ausgabeknoten bestand, die vollständig miteinander verbunden waren. Als Trainings- und Testdaten wurden die Stichproben mit jeweils 1000 Musterbeispielen ausgewählt. Bei der genetische Entwicklung der optimierten Architekturen zur Ziffernerkennung wurden 11019 unterschiedliche Architekturen erzeugt. Das beste Netz wurde in der 7532 Generation gefunden. Es weist auf der Teststichprobe einen quadratischen Fehler von 89.56 auf. Abbildung 4 zeigt den Aufbau dieses Netzes. Man erkennt zwei verdeckte Schichten, von denen eine mittels rezeptiver Felder vom Typ SHARED und der Größe $5 * 5$ mit der Eingangsschicht verbunden ist. Die zweite verdeckte Schicht ist mit der Ersten voll vernetzt. Zudem weist sie über rezeptive Felder Verbindungen zur Eingangsschicht auf, die vom Typ FULLY_SHARED sind. Die Ausgangsschicht besitzt ebenfalls rezeptive Felder zur Eingangsschicht, die von diesem Typ sind. Weiterhin ist die Ausgangsschicht mit beiden verdeckten Schichten vollständig vernetzt, da das angegebene rezeptive Feld zur zweiten verdeckten Schicht diese vollständig überdeckt. Auf der Testmenge wurde mit diesem Netz eine Klassifikationsgüte von 98.2% erreicht

5 Diskussion

Für die beiden vorgstellten Klassifikationsprobleme wurden Netzwerke entwickelt, die auf die gegebene Problemstellung optimal angepaßt sind. Beim Contiguity Problem entwickelte der genetische Algorithmus zudem eine Topologie, die genau der *menschlichen* Löschung entspricht. Das Netz zur Ziffernerkennung hingegen zeichnet sich durch eine sehr geringe Komplexität bei gleichzeitig hohem Generalisierungsverhalten aus. In weiteren Experimenten zeigte sich, daß

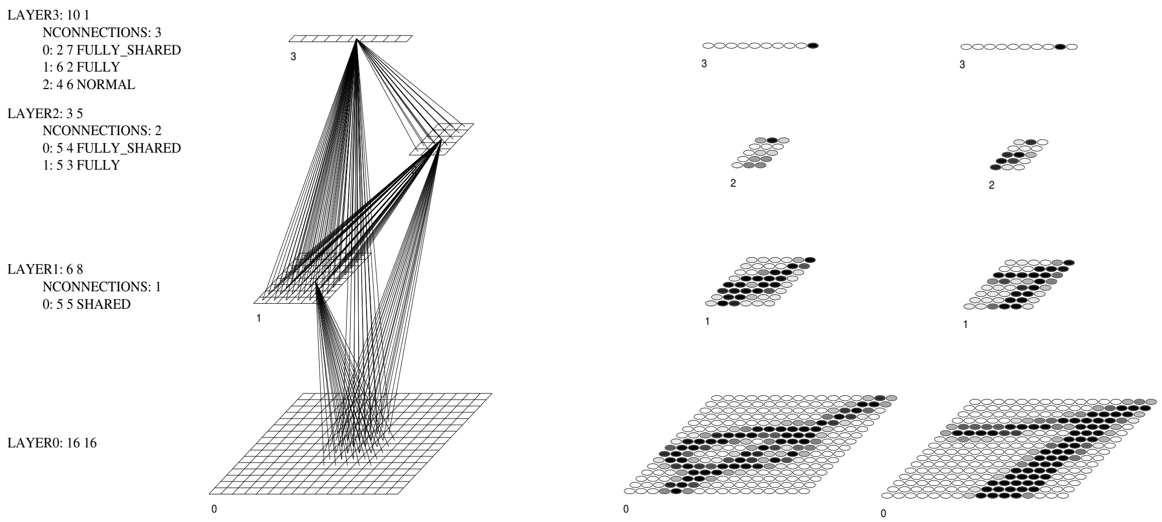


Abbildung 4: Die entwickelte Architektur zur Ziffernerkennung, sowie die Aktivierung der Neuronen anhand zweier Musterbeispiele.

eine vergleichbare Güte nur durch manuell optimierte Architekturen erreicht werden, die eine deutlich höhere Konnektivität aufweisen.

Literatur

[1] K. Balakrishnan and V. Honavar. Evolutionary design of neural architectures – a preliminary taxonomy and guide to literature. Technical Report CS TR # 95-01, Iowa State University, Dep. of Computer Science, http://www.cs.iastate.edu/~gannadm/homepage.html, ftp.cs.iastate.edu: /pub/gann/bib/gann.bib.Z, January 1995.

[2] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Neural Information Processing Systems 2*, pages 524–532, 1990.

[3] D.H. Hubel. *Auge und Gehirn: Neurobiologie des Sehens*. Spektrum der Wissenschaft, 1989:.

[4] R. Reed. Pruning algorithms – a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, September 1993.

[5] W. Schiffmann. Topologie–Optimierung von Neuronalen Netzen mittels evolutionärer Algorithmen. *Nachrichten Neuronale Netze (N^3)*, 3/95:2–5, 1995.

[6] S.A. Solla. Learning and generalization in layered neural networks: The contiguity problem. In L. Personnaz and G. Dreyfus, editors, *Neural Networks from Models to Applications*, pages 168–177. I.D.S.E.T., Paris, 1989.