Untersuchungen zur Untergrundreduktion mit einem neuronalen Trigger beim H1-Experiment bei HERA

Diplomarbeit an der Fakultät für Physik
der
Ludwig-Maximilians-Universität München

vorgelegt von
Ludger Janauschek
München
28. Juli 1998

angefertigt am
Max-Planck-Institiut für Physik München
Werner-Heisenberg-Institut

Erstkorrektor: Prof. Dr. Chr. Kiesling

Zweitkorrektor: Prof. Dr. D. Schaile

Inhaltsverzeichnis

1 Einleitung

Schon in antiker Zeit war das Interesse der Menschen am Aufbau der Materie sehr groß. Jedoch waren ihre Bemühungen zunächst eher spekulativer Art. Thales von Milet (ca. 640 - 546 v. Chr.) löste sich als erster aus der Welt des Mythos und suchte nach rationalen Erklärungen. Mit der Lehre von den vier Elementen war Empedokles ein Vorläufer der Atomisten. Deren erste Vertreter Leukipp (um 450 v. Chr.) und sein Schüler Demokrit (ca. 465 - 375 c. Chr.) kamen durch reine Überlegung zu der Vorstellung, daß Materie nicht beliebig oft teilbar sei und damit aus unteilbaren Bestandteilen aufgebaut sein müsse: [1]

Λεύκιππος [...] πρῶτος τε ἀτόμους ἀρχας ὑπεστήσατο.

In der neuzeitlichen Forschung wurde für das Atom der Begriff vorschnell verwendet.

Schon die Betrachtung gewöhnlicher Gegenstände mit dem bloßen Auge stellt die Messung der gestreuten Lichtquanten dar. Um das Auflösungsvermögen, welches mit der Energie der Streuteilchen steigt, zu vergrößern, wurde von Lichtmikroskopen zu Elektronenmikroskopen übergegangen. Da für das Auflösungsvermögen die Schwerpunktsenergie zwischen Streuteilchen und Untersuchungsobjekt maßgeblich ist, ist es von Vorteil, auch das zu untersuchende Objekt zu beschleuigen und mit dem Streuteilchen zur Kollision zu bringen.

Am Speicherring HERA bei DESY werden Elektronen/Positronen mit einer Energie von 27 GeV mit Protonen der Energie 820 GeV zum Zusammenstoß gebracht. Dadurch wird ein Auflösungsvermögen bis zu 10^{-18} m erreicht, welches weit unter der Ausdehnung von Nukleonen liegt.

Neben den interessanten Elektron-Proton-Kollisionen treten mit einer um mehrere Größenordnungen höheren Rate Untergrundereignisse auf. Da die Bandbreite bei der Datenübertragung begrenzt ist, ist es nötig, interessante Physikereignisse bevorzugt auszuwählen und störenden Untergrund zurückzuweisen. Dies ist die Aufgabe des sog. Triggers.

In dieser Arbeit wird die spezielle Klasse von Untergrundereignissen betrachtet, deren Ereignisse, die sog. *big events*, viele Signale in den zentralen Spurkammern verursachen und dadurch auf höhergelegenen Triggerstufen bei der Spurrekonstruktion durch lange Rechenzeiten die Totzeit des Experiments erhöhen.

Ziel ist es, einen Trigger auf niederer Stufe (*level* 2) zu schaffen, der diese Untergrundereignisse aufgrund gering vorverarbeiteter Detektordaten erkennt und zurückweist. Für diese Aufgabe sind eindimensionale Schnitte aufgrund mangelnder Effi-

[1] Leukipp nahm als erster den unteilbaren Grundstoff an.

zienz unzureichend. Es müssen also hochdimensionale Korrelationen der Eingabedaten betrachtet werden. In diesem Musterraum sollen die Daten dieser speziellen Untergrundklasse eingegrenzt werden. Aufgrund dieser Eigenschaft wird das Triggerprojekt *background encapsulator* genannt.

Für Dichteschätzungen und Enkapsulierungen in hochdimensionalen Musterräumen haben sich neuronale Netzt bewährt. Insbesondere werden für diese Aufgabe unüberwachte Trainingsmethoden verwendet, mit denen Zentren und Ausdehnung von Datenanhäufungen (*cluster*) erkannt werden sollen.

Im Rahmen dieser Arbeit, wurden unüberwachte Trainingsalgorithmen entwickelt und implementiert. Deren Anwendung auf die spezielle Untergrundklasse der *big events* wird untersucht.

2 HERA und H1-Detektor

... παρὰ δὲ χρυσόθρονος Ἥρη.
[19, Erster Gesang, Vers 611]

Am 6.3 km langen Speicherring HERA befinden sich 4 Experimente. Mit H1 und ZEUS werden Elektron-Proton-Kollisionen untersucht. Bei HERMES wird die Wechselwirkung zwischen polarisierten Elektronen und einem feststehenden Wasserstofftarget für die Bestimmung spinabhängiger Strukturfunktionen beobachtet, während HERA-B B-Zerfälle über Reaktionen von Protonen aus dem Protonstrahlhalo mit Drahttargets analysiert.

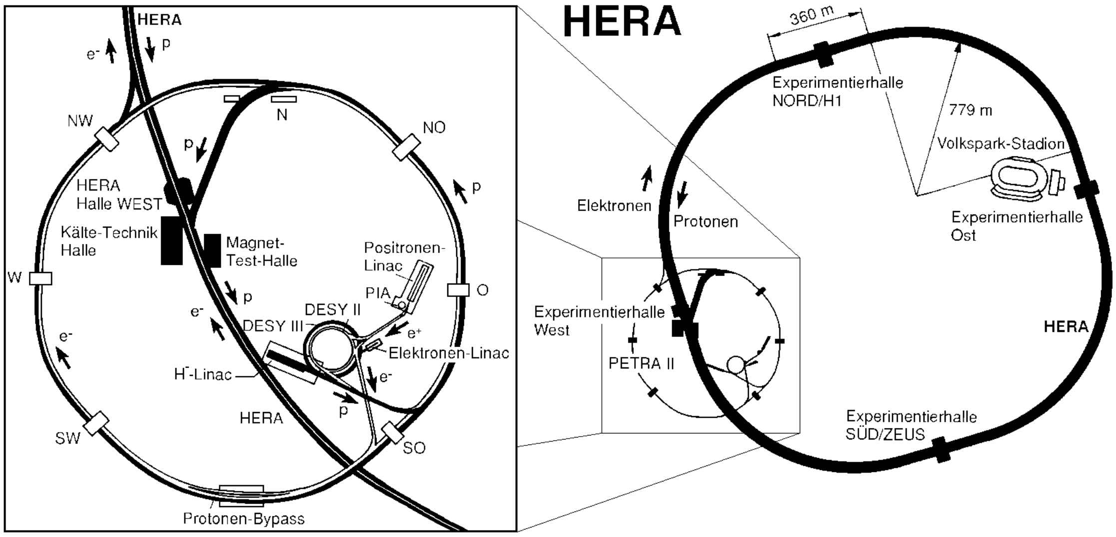

Abbildung 1: Speicherring HERA mit seinen Vorbeschleunigungsanlagen und den vier Experimentierhallen

Elektronen (Positronen) werden auf eine Energie von 27.5 GeV beschleunigt und mit auf 820 GeV beschleunigten Protonen bei einer Schwerpunktsenergie von 300 GeV zur Kollision gebracht. Um wesentlich längere Strahllebensdauern zu erreichen, wurden von 1994 bis 1997 Positronen anstelle von Elektronen verwendet. 1998 sollen, nach Verbesserung des Vakuumsystems, wieder Elektronen beschleunigt werden.

Während Lebensdauern über 24 h für den Protonstrahl erreicht wurden, betrug die Lebensdauer des Elektronenstrahls in der Anfangsphase von HERA ca. 6 h. Durch die Verwendung von Positronen wird diese um den Faktor 2 vergrößert. [2]

Um die hohe Luminosität (Designwert $\mathcal{L} = \infty //$ pb$^{-\infty}$ a$^{-\infty}$ [16]) zu erreichen, welche für Hochpräzessionsmessungen und die Untersuchung seltener Ereignisse in Elektron-Proton-Wechselwirkungen benötigt wird, stellt HERA *multi-bunch beams* (maximal 210 Protonbündel und maximal 210 Elektronbündel) zur Verfügung. Bei einem Umfang von 6.3 km führt dies zu einer Wechselwirkungsfrequenz von 10.4 MHz. Das Zeitintervall zwischen zwei Elektron-Proton-Bündeldurchkreuzungen (*bunch crossing*, BC) beträgt 96 ns und bildet die Einheit für die HERA-*clock*. Im folgenden wird BC auch als Bezeichnung für das Zeitintervall 96 ns verwendet. Aus dem Zusammenhang sollte klar werden, ob das Zeitintervall oder der Zeitpunkt gemeint ist.

Bei HERA werden neue kinematische Bereiche für DIS(deep inelastic scattering) erreichbar. Solche Experimente haben eine grundlegende Rolle im Verständnis der fundamentalen Kräfte gespielt und haben den ersten Hinweis auf die punktartigen Bausteine der Nukleonen geliefert.

HERA ermöglicht den *hadronic recoil* zu beobachten sowie Ereignisse des *neutral current* (NC) und *charged current* (CC) bei hohem Impulsübertrag zu untersuchen.

Da die Endzustände aus mehreren Leptonen und/oder Quark-Gluon-Jets bestehen, werden an die Detektoren folgende Anforderungen gestellt:

- gute Leptonidentifizierung (insbesondere Elektronen)
- hohe Granularität und Auflösung in den Spurkammern und Kalorimetern für die Erkennung von Jets
- gute Hermitizität, um Teilchen, wie z. B. Neutrinos, aufgrund von fehlender transversaler Energie feststellen zu können

Dazu werden ein bestmögliches hadronisches Kalorimeter, ein hochauflösendes und einen großen Raumwinkel abdeckendes System von Spurkammern und ein ausgefeiltes Triggersystem benötigt.

2.1 H1-Detektor

Um eine saubere Identifizierung der Elektronen und die Messung ihrer Energie zu ermöglichen, befindet sich die große Solenoidspule außerhalb des elektromagnetischen und hadronischen Kalorimeters. Dadurch wird die Menge der toten Materie vor den Kalorimetern reduziert.

Das LAr-Kalorimeter bietet erprobte Stabilität, einfachen Kalibration und feine Granularität. Letztere ermöglicht in einem hohen Grad die Separation von Elektronen und Pionen.

Die unterschiedliche Energie des Elektron- und Proton-Strahls führt zu einem mit $\gamma = 2.86$ *geboosteten* Schwerpunksystem. Deshalb ist der H1-Detektor asymetrisch aufgebaut. In Vorwärtsrichtung, d. h. Protonstrahlrichtung bzw. Richtung der z-Achse, ist er maßiver und höher segmentiert.

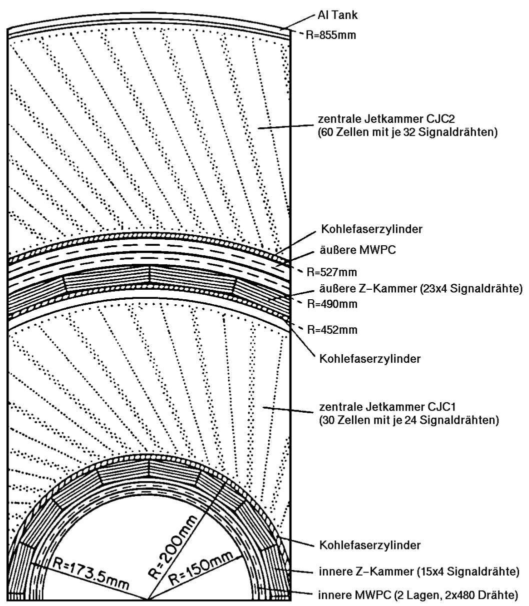

Abbildung 2: Schnitt durch die zentralen Spurkammern senkrecht zur z-Achse

Der H1-Detekor ist zwiebelschalenartig um den nominellen Wechselwirkungspunkt ($x = y = z = 0$ cm) aufgebaut (von innen nach außen):

- zentrale, rückwärtige und Vorwärtsspurkammern
- elektromagnetisches und hadronisches LAr-Kalorimeter, SpaCal und Plug-Kalorimeter
- supraleitende Solenoid-Spule
- Magnetflußrückführungsjoch aus Eisen mit Kammern zur Myonerkennung
- Vorwärtsmyonspurkammern mit Eisentoroidmagnet
- Protontagger, Luminositätssystem

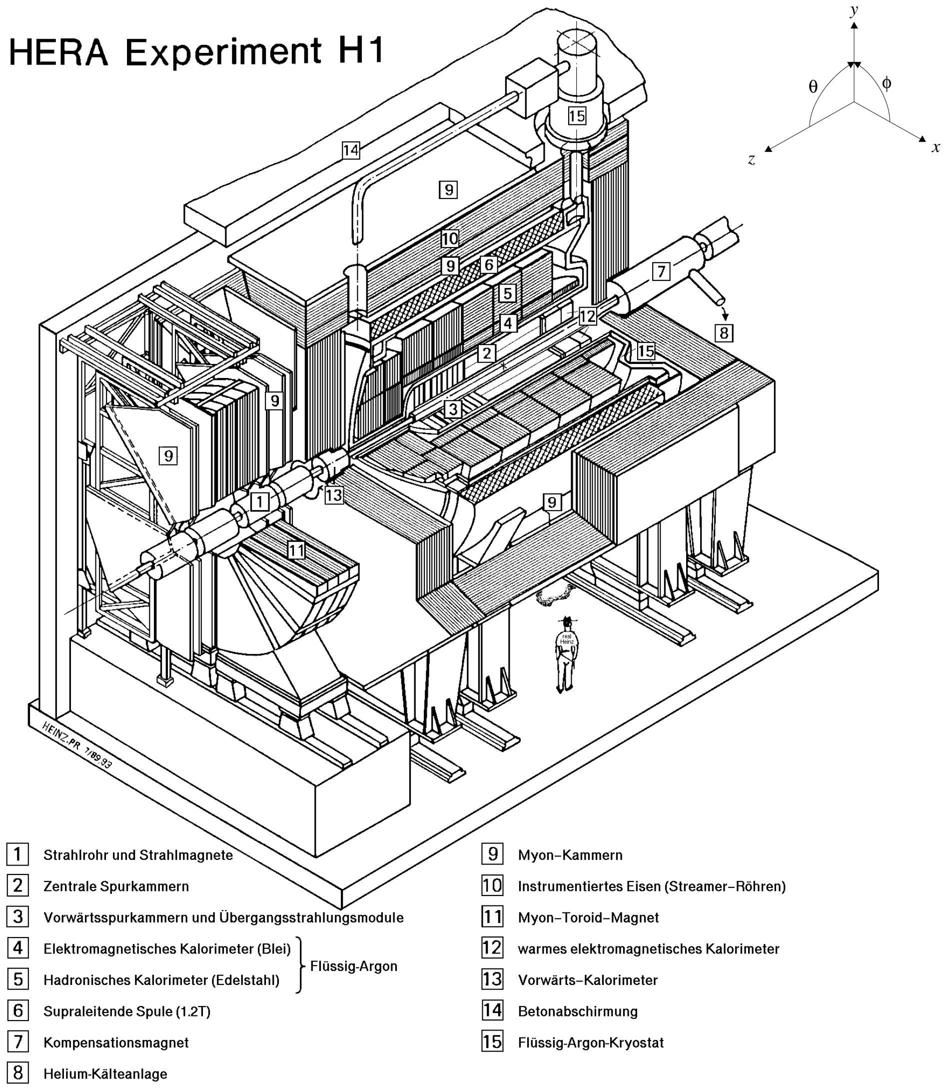

Abbildung 3: H1-Detektor (1994): Das warme elektromagnetische Kalorimeter $\boxed{12}$ wird seit dem Winter-*shutdown* 1994/95 durch das Spaghettikalorimeter SpaCal (elektromagnetisch, hadronisch) ersetzt. Nicht eingezeichnet sind die 1995 eingebauten Siliziumspurdetektoren CST und BST sowie die rückwärtige Driftkammer BDC. Die z-Achse der H1-Koordinatensystems zeigt in die Richtung des Protonenstrahls, die y-Achse senkrecht nach oben und die x-Achse zum Mittelpunkt des HERA-Ringes. Der Abstand von der Strahlachse wird mit $r = \sqrt{x^2 + y^2}$ bezeichnet.

2.1.1 supraleitende Solenoidspule

Die supraleitende Solenoidspule besteht aus 4 getrennten, koaxialen, in Serie geschalteten Spulen. Diese befinden sich in einem gemeinsamen Kryostaten mit 575.0 cm Länge und innerem bzw. äußeren Durchmesser von 518.0 cm bzw. 608.0 cm. Zur Kühlung wird flüssiges Helium aus der zentralen Versorgungsanlage von DESY verwendet.

Bei nominalem Strom 5514 A und Spannung 12 V wird durch die Solenoidspule und das Eisenrückführungsjoch im Bereich der Spurdetektoren ($r < 80.0$ cm, -112.5 cm $< z <$ 250.0 cm) ein fast homogenes Magnetfeld mit durchschnittlich 1.15 T erzeugt. Die größte Abweichung beträgt 4.5% am rückwärtigen Rand der zentralen Spurkammern. Die für die Spurrekonstruktion wichtige Komponente B_z parallel zur Strahlröhre ist über das Volumen der Spurdetektoren mit einer Genauigkeit von 0.3% bekannt. [2]

2.1.2 Spurkammern

Das 1.5 T starke strahlparallele Magnetfeld führt dazu, daß geladene Teilchen sich auf Helixbahnen bewegen, deren Achse parallel zur z-Achse verläuft. Aus diesen Spuren können die Impulse der Teilchen, die Ereignistopologie und die Position des Vertex bestimmt werden.

Für die Rekonstruktion der Spuren werden die Daten der Spurdetektoren verwendet.

Im Polarwinkelbereich ϑ von ca. 20° bis 160°, dem Zentralbereich, werden für die Spurrekonstruktion die Daten der Driftkammern [30] CJC1, CJC2, CIZ und COZ verwendet.

Bei den *central jet chambers* CJC1 und CJC2 verlaufen die Signaldrähte parallel zur z-Achse. Diese sind so in Reihen angeordnet, daß 30 bzw. 60 Zellen in φ mit je 24 bzw. 32 radialen Lagen in der CJC1 bzw. CJC2 enstehen. Die Signaldrahtebenen sind gegenüber der radialen Richtung um einen Winkel von 35° geneigt. Durch die Neigung der Zellen wird der aufgrund des Magnetfeldes entstehende Winkel zwischen elektrischem Feld und den driftenden Elektronen (Lorentzwinkel) kompensiert, so daß die Drift der Elektronen fast senkrecht zur Spur erfolgt. Dadurch wird die Ortsauflösung in der $r\varphi$-Ebene verbessert [44].

Die Neigung der Signaldrahtebenen bewirkt auch, daß Spuren mit Transversalimpuls $p_t >$ 400 MeV/ c wenigstens einmal die Signaldrahtebenen von CJC1 und CJC2 kreuzen. Die Driftzeit zu Drähten dieser Ebenen ist kleiner als 50 ns; also kleiner als die Zeit zwischen zwei *bunch crossings* (96 ns). Dadurch kann der DC$r\varphi$-Trigger (siehe Abschnitt 3.1.4) Spuren ein *bunch crossing* zuordnen.

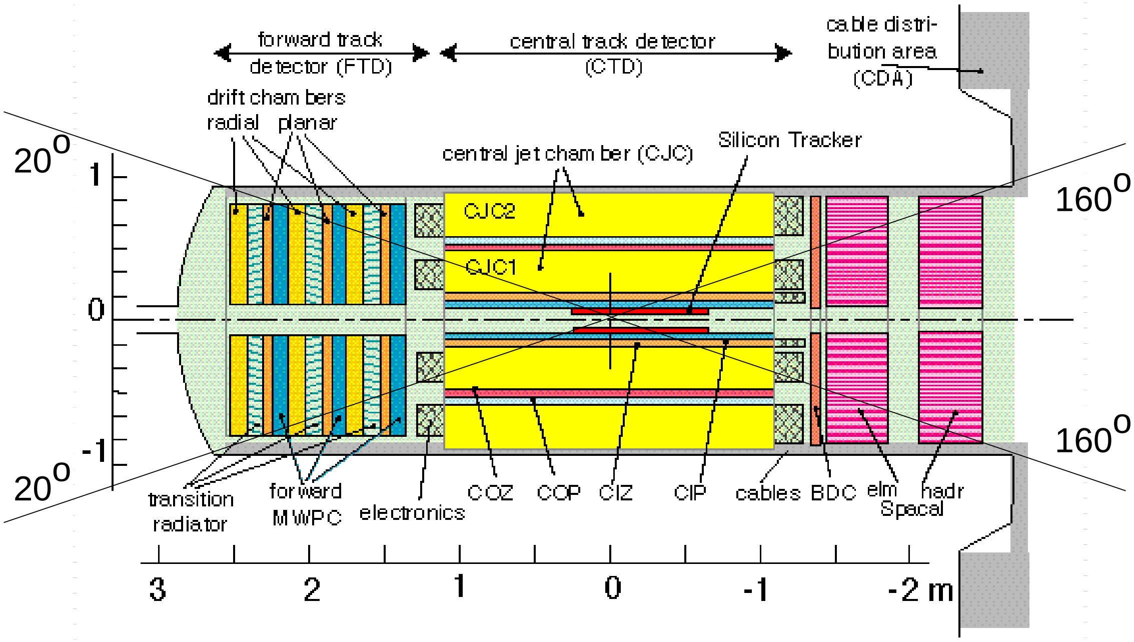

Abbildung 4: Seitenansicht des Spurkammersystems des H1-Detektors. Der rückwärtige Siliziumspurdetektor BST ist nicht eingezeichnet.

Hochenergetische Spuren haben zusätzlich auch mindestens einen Zellenübergang. Dadurch ist es möglich rechst/links-Mehrdeutigkeiten der Driftkammern aufzulösen, da die Spiegelspur in der Nachbarzelle nicht fortgesetzt wird. Um diese Mehrdeutigkeit auch für kurze Spurstücke auflösen zu können, sind die Signaldrähte abwechselnd um 150 μm aus der Signaldrahtebene verschoben (*staggering*). Dadurch sind die Punkte der Spiegelspur für benachbarte Signaldrähte um 300 μm verschoben.

Die Ortsauflösung in der $r\varphi$-Ebene beträgt $\sigma_{r\varphi} = 170$ μm.

Die Verschiebung der Signaldrähte gegenüber der Signaldrahtebene führt auch mit der elektrostatischen Abstoßung der Signaldrähte untereinander zu einer kontrollierten Positionierung.

1992 wurde beispielsweise die CJC mit dem nichtbrennbaren Kammergas $Ar/CO_2/CH_4$ (84 : 10 : 1) betrieben. Für dieses Gas ist die Driftgeschwindigkeit der Elektronen 49 μm/ ns und der Lorentzwinkel 42° [44].

Die z-Kammern CIZ, COZ (*central inner/outer z chamber*) dienen der Rekonstruktion der z-Koordinate der Spuren. Bei diesen Kammern verlaufen die Signaldrähte ringförmig um die Strahlachse mit einer Aufteilung in 15 bzw. 24 Segmente in der CIZ bzw. COZ. Die Ortsauflösung beträgt $\sigma_z^{\mathrm{CIZ}} = 260$ μm bzw. $\sigma_z^{\mathrm{COZ}} = 200$ μm.

An der Innenseite der CIZ bzw. der Außenseite der COZ befinden sich die innere (CIP) bzw. die äußere (COP) Proportionalkammer (*multi wire proportional chamber* (MWPC), [30]) (*central inner/outer proportional chamber*). Jede dieser Kammern besteht aus einer Doppellage von Drähten. In z-Richtung ist die CIP bzw. COP in 60 bzw. 18 Segmente unterteilt. In der $r\varphi$-Ebene sind beide Kammern in 16 Sektoren unterteilt.

Die Kammern CIP, COP werden hauptsächlich für Triggerzwecke verwendet (siehe z-Vertex-Trigger im Abschnitt 3.1.4).

Den rückwärtigen Bereich von $150° < \vartheta < 177.5°$ abdeckend, befindet sich die *backward drift chamber* (BDC) bei $z = -142$ cm. Sie besteht aus 4 Doppellagen, die jeweils um 11.5° zueinander verdreht sind. Die Drähte jeder Lage sind oktogonal um die Strahlrichtung gespannt.

Den Vorwärtsbereich von $5° < \vartheta < 25°$ decken die vorderen Spurkammern ab. Diese bestehen aus drei hintereinander angeordneten Supermodulen, welche jeweils aus einer radialen und einer planaren Driftkammer, einer Proportionalkammer und einem Übergangsstrahlungsmodul [25, 9] aufgebaut sind.

Im Vorwärtsbereich, außerhalb des H1-Magnetfeldes befindet sich das Vorwärtsmyonspektrometer. Es besteht aus je drei Driftkammerebenen vor und hinter dem Eisentoroidmagneten und deckt den Winkelbereich $4° < \vartheta < 17°$ ab. Da in dieser Arbeit hautpsächlich die Signale der zentralen Spurkammern verwendet werden, wird auf die Vorwärts- und Rückwärtsspurkammern hier nicht weiter eingegangen.

Für die Siliziumspurdetektoren *central silicon tracker* (CST) und *backward silicon tracker* (BST) sei auf die Literatur verwiesen (siehe z.B. Literaturverzeichnis von [2]).

2.1.3 Kalorimeter

Das LAr-Kalorimeter [2, 25] umgibt das Spurkammersystem und deckt den Winkel $4^\circ < \vartheta < 155^\circ$ ab. Es ist aufgebaut aus schichtweise angeordneten Absorberplatten und flüssigem Argon, welches als aktives Jonisationsmedium dient. Die Absorberplatten des elektromagnetischen Teils bestehen aus Blei, während im hadronischen Teil Edelstahl verwendet wird.

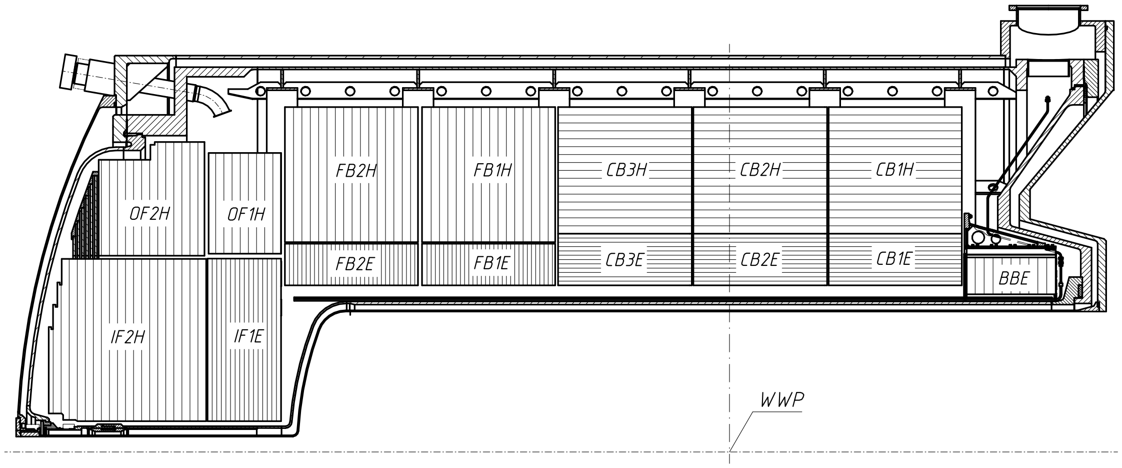

Abbildung 5: vertikaler Schnitt durch das LAr-Kalorimeter

Das Kalorimeter ist nicht kompensierend, d. h., daß Verlußte an unsichtbarer Energie bei hadronischen Schauern nicht kompensiert werden [9, 25]. Dadurch liefern elektromagnetische und hadronische Schauer, die von Teilchen gleicher Energie ausgelöst wurden, unterschiedliche Signale. Aufgrund der feinen Granularität des Kalorimeters ist jedoch durch die unterschiedlichen Schauerprofile eine Trennung und dadurch eine nachträgliche Korrektur möglich.

Die Energieauflösung beträgt $\sigma(E)/E \approx 12\%/\sqrt{E/\text{ GeV}}$ für Elektronen und $\sigma(E)/E \approx 50\%/\sqrt{E/\text{ GeV}}$ für Pionen.

Für Triggeranwendung auf Triggerstufe 1 ist das LAr-Kalorimeter in zum nominellen Vertex zeigende Bereicht, den sog. *big towers*, unterteilt.

Um die Messung der Energie und des Polarwinkels in Rückwärtsrichtung gestreuter Elektronen zu verbessern, wurden die bestehenden Detektoren im Rückwärtsbereich durch neue Detektoren ersetzt. Diese sind der Siliziumspurdetektor BST, die planare Driftkammer BDC und das Spaghetti-Kalorimeter SpaCal.

Seit dem Winter-*shutdown* 1994/95 ersetzt das SpaCal das *backward electromagnetic calorimeter* (BEMC).

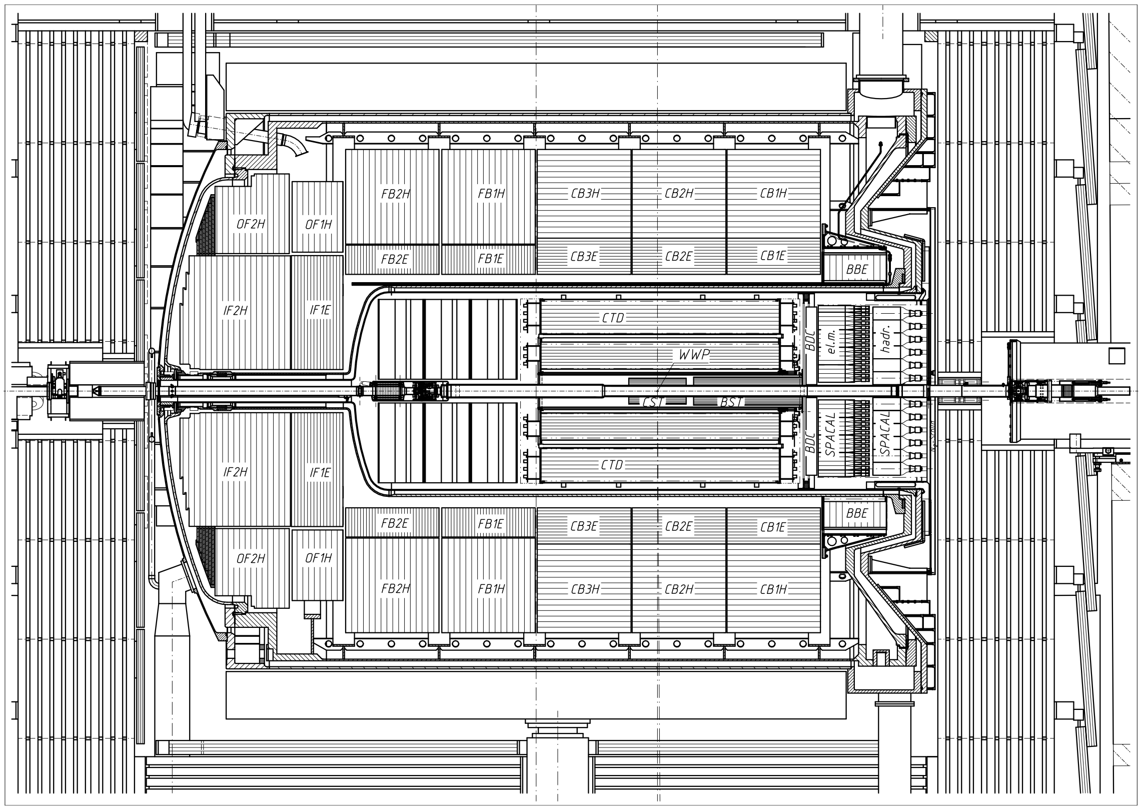

Abbildung 6: vertikaler Schnitt durch den H1-Detektor

Das SpaCal deckt den rückwärtigen Bereich $155° < \vartheta < 177.5°$ ab und verbessert Energie- und Ortsauflösung. Es besteht aus zwei ca. eine Strahlungslänge [30] tiefen Schichten. Die erste Schicht mißt elektromagnetische Schauer. Die zweite dient sowohl zur Messung von Ausläufern elektromagnetischer Schauer aus dem elektromagnetischen Teil, als auch zur Messung des hadronischen Energieflußes. Beide Teile bestehen aus Blei als Absorbermaterial und darin gelagerten optischen Szintillationsfasern. Der elektromagnetische und hadronische Teil unterscheiden sich im Verhältnis von Blei zu Fasern und in der Einteilung und Größe der Zellen zu denen die Fasern zusammengefaßt werden.

Direkt am Strahlrohr sind vier ringförmige Szintillatoren angebracht. Ein Veto auf Energiedepositionen in diesen Szintillatoren gewährleistet, daß die gesamte Energie

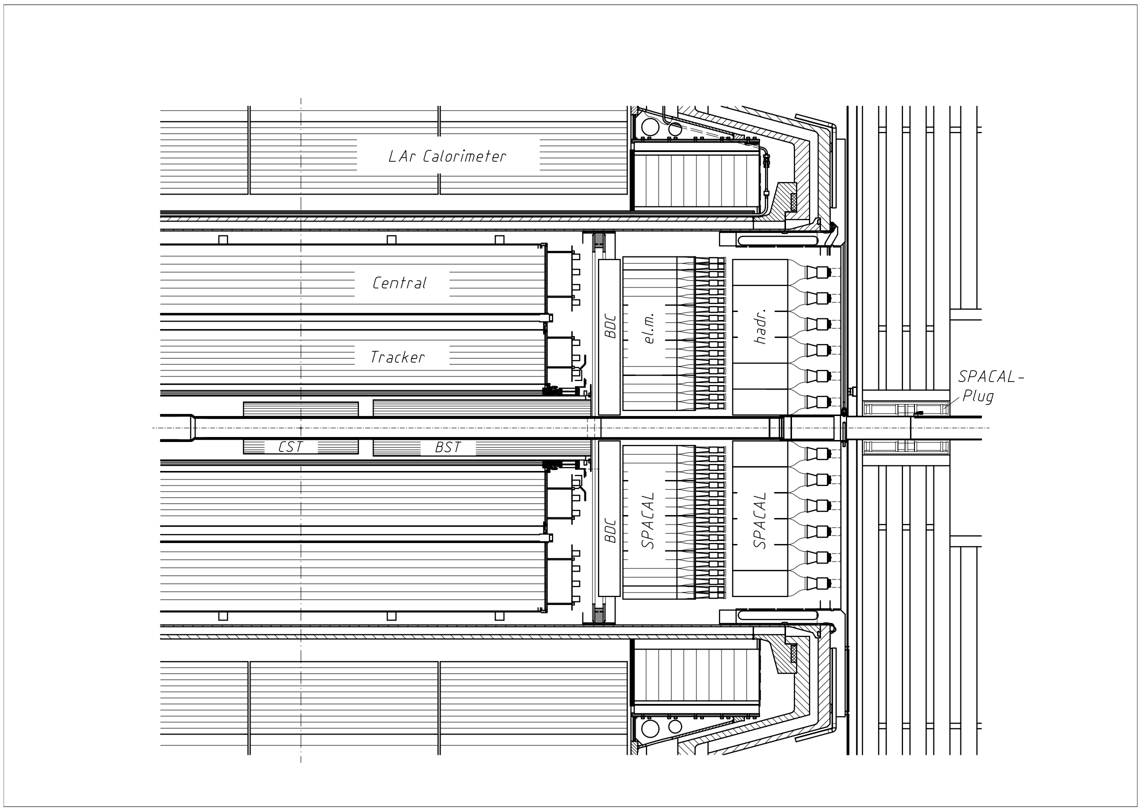

Abbildung 7: vertikaler Schnitt durch rückwärtigen Bereich des H1-Detektors

des gestreuten Elektrons erfaßt wird [17].

Der hadronische Teil des SpaCal ermöglicht mittels *time of flight* ein Veto auf Protonstrahluntergrundereignisse. Schnelle Signale werden auch für die Trigger benutzt. [2, Seite 345]

Für weitere Detektorkomponenten sei auf [2] verwiesen.

2.2 Untergrund

Die üblichen Quellen von Untergrundereignissen an Beschleunigerexperimenten finden sich auch bei HERA: [2, Seite 325]

- Synchrotronstrahlung des Elektronenstrahls
- Protonstrahl-Gas-Wechselwirkungen in der Strahlröhre (Vakuum 10^{-9} hPa)
- Gestreute Protonen treffen die Strahlröhre und andere Materie um den Beschleuniger. Dadurch werden Teilchenschauer erzeugt.
- Strahlhalomyonen
- kosmische Myonen

Die geringe radiale und zeitliche, d. h. entlang des Strahls, Ausdehung der Teilchenbündel und die Durchkreuzung der Strahlen mit einem Winkel von 0° führt zu einer Wechselwirkungszone von ca. -50 cm $< z <$ 50 cm für Elektron-Proton-Wechselwirkungen.

Bei typischen Vakuumbedingungen in der anfänglichen Datennahmeperiode betrug der Druck des hauptsächlich aus Wasserstoff und Kohlenmonoxid bestehenden Restgases in der Strahlröhre 1 bis $2 \cdot 10^{-9}$ hPa. Unter Annahme von Stickstoff als durchschnittlichen Repräsentanten für Restgasbestandteile und einem Wirkungsquerschnitt von ca. 200 mb für Protonen bei 820 GeV (Schwerpunktenergie $\sqrt{s} = 39$ GeV) wird eine Proton-Stickstoff-Wechselwirkung bei jedem 10^4-ten *bunch crossing* auf einer Wechselwirkungslänge von 1 m bei Designluminosität erwartet [2].

Dem gegenüber steht eine Elektron-Proton-Wechselwirkung bei jedem 10^5-ten *bunch crossing*. Die Rate der Physikereignisse von 200 Hz wird durch Photoproduktionsereignisse dominiert, während der Strahl-Gas-Untergrund eine Rate von ca. 1 kHz liefert. [2]

Die Rate der Untergrundereignisse (ca. 100 kHz) übersteigt die Rate der Elektron-Proton-Wechselwirkungen um mehrere Größenordnungen.

Der Strahl-Gas-Untergrund ist nicht auf die Wechselwirkungszone beschränkt, sondern kann durch Wechselwirkungen auf dem ganzen Weg der Protonen durch den Detektor entstehen. Eine teilweise Erkennung von Strahl-Gas-Wechselwirkungen jenseits des SpaCals ist durch Flugzeitmethoden möglich.

Weitere Untergrundereignisse entstehen auch durch Protonen des Strahlhalos, die Strahlrohr- oder Detektorkomponenten treffen.

Eine weitere Quelle von Untergrundereignissen ist die Synchrotronstrahlung des Elektronenstrahls durch die Fokussierungmagnete. Diese kann jedoch durch Kollimatoren größtenteils unterdrückt werden.

Auch Myonen aus kosmischer Strahlung tragen zum Untergrund bei. Diese treffen den Detektor mit einer Rate von ca. 1 kHz. Durch Anwendung einer Vertexbedingung kann die Rate leicht reduziert werden.

Länge eines Protonbündels	1.4 ns
Abstand zum nächsten Satellitenbündel	5 ns
Flugzeit zum rückwärtigen ToF	6 ns
Flugzeit zum *barrel muon system*	20 ns
bunch crossing Intervall	96 ns
längste Driftzeit in der CJC	1 μs
Integrationszeit der LAr-Vorverstärker	1.5 μs
Verzögerung des Level 1 Triggers	2.5 μs
front end readout time	$\approx$ 1 ms

Tabelle 1: Zeitskala bei HERA und H1

3 Trigger

Mit Trigger (engl. Auslöser) wird das System bezeichnet, das den Datenaufzeichnungsprozeß auslöst. Die Ausgangsrate ist durch die Bandbreite der Übertragung auf Massenspeicher begrenzt. An den Trigger wird die Anforderung gestellt, innerhalb dieser Rate mit möglichst hoher Effizienz interessante Physikereignisse auszuwählen.

Die durch den Protonstrahl erzeugten zahlreichen Untergrundereignisse sind oft nur schwierig von den interessierenden Physikereignissen zu unterscheiden. Daher wird ein intelligentes Triggersystem mit hoher Zurückweisungsrate benötigt.

Beim H1-Experiment wird für die *online*-Datenfilterung ein vierstufiges Triggersystem verwendet.

Der Level-1-Trigger (L1) und der Level-4-Trigger (L4) sind seit dem Start der Datennahme im Jahr 1992 in Betrieb. Der Trigger auf Level 2 (L2) arbeitet seit dem Ende der HERA Datennahmeperiode 1995. Trigger-Level-3 (L3) wird derzeit nicht benutzt.

L1 und L2 sind *hardware*-Triggerstufen. L3 und L4 sind *software*-Triggerstufen.

Für die Designwerte der Raten vor bzw. nach den einzelnen Triggerstufen siehe Abbildung 8.

3.1 Level 1

3.1.1 Triggerelemente des Level 1

Die Kollisionsfrequenz der Elektron- und Protonbündel von HERA beträgt im H1-Experiment 10.4 MHz. Dies fordert die Speicherung aller Daten der *front-end readout*-Systeme des Experiments in Pipelines. Deren Länge setzt wiederum strenge Beschränkungen auf die Entscheidungszeit des Triggers der ersten Stufe.

Das L1-System wird durch die HERA-*clock* getakted und stellt für jedes *bunch crossing* nach einer Verzögerung von 2.3 μs eine Triggerentscheidung ohne Totzeit zu verursachen zur Verfügung.

Die Daten der Subdetektorsysteme werden in *front-end pipelines* gespeichert, um eine schnelle „Ja/Nein“-Information über die allgemeinen Eigenschaften des Ereignisses zu erzeugen. Diese boolsche Informationen sind die L1-Triggerelemente (L1TE).

Typische Eigenschaften, welche zur Bildung der L1TE verwendet werden, sind Multiplizität und Impulse geladener Spuren in den Driftkammern oder lokalisierte Energiedepositionen (*cluster*) in den Kalorimetern.

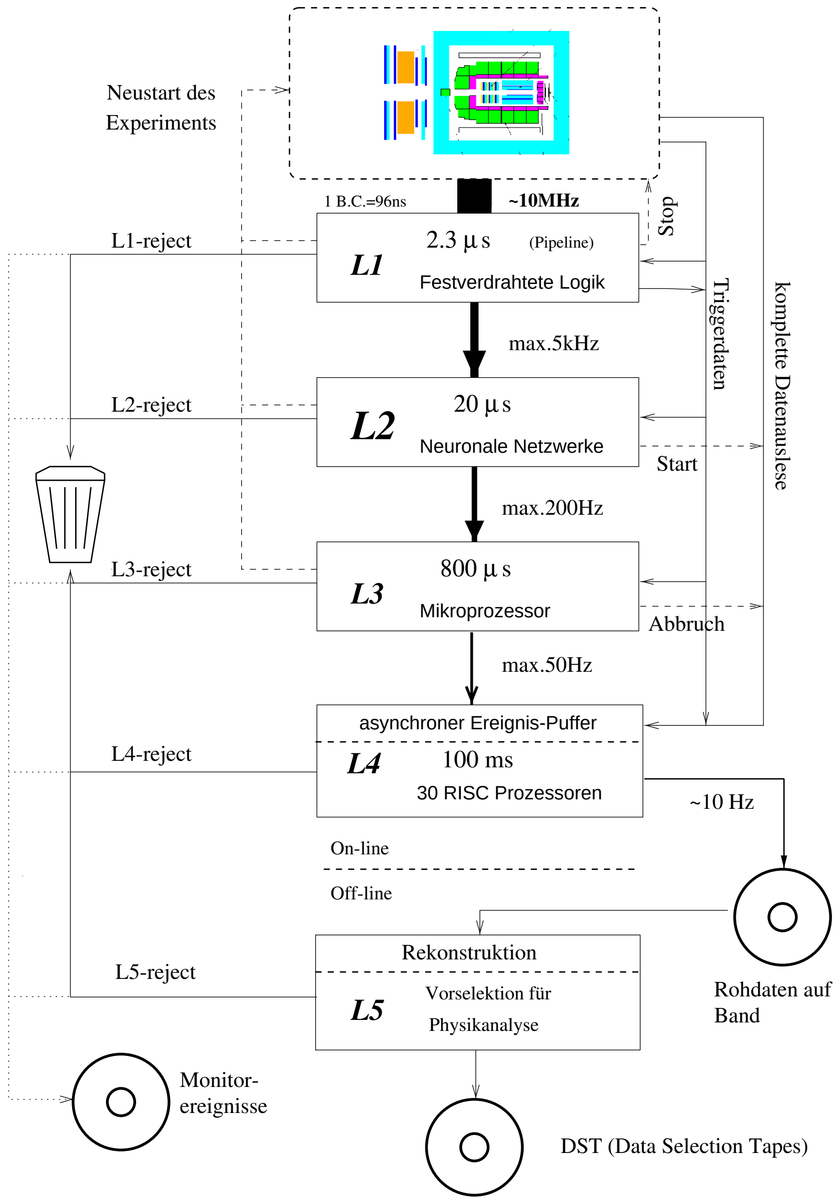

Abbildung 8: Das Trigger- und Datennahmesystem des H1-Experiments. Signale sind gestrichelt, Datenflüsse durchgezogen und gepunktet dargestellt. Die angegebenen Raten sind maximale Designwerte (aus [32]).

Die Wartezeit auf die L1-Triggerentscheidung und damit die Länge der *front-end pipeline* wird durch die Zeit der Signalerzeugung in den Subdetektoren, die Laufzeiten in den Kabeln und die Verarbeitungszeiten in den Triggersubsystemen bedingt. Die maximale Driftzeit in den Spurkammern beträgt ca. 11 BC . Die Ladungssammlung und Integration im LAr-Kalorimeter benötigt 11 BC. Dadurch ist die LAr-Triggerinformation, einschließlich der Summation aller Energien, die letzte für die L1-Entscheidung verfügbar werdende Information.

Von den Triggersubsystemen werden bis zu 192 L1TE (zur Zeit 128) an die *level 1 logic of the central trigger* (CTL1) geschickt. Dort werden sie einzeln synchronisiert und verbunden, um sicher zu stellen, daß die Information des selben *bunch crossings* für die L1-Entscheidung herangezogen wird. [48]

3.1.2 Subtrigger des Level 1

Die L1-Triggerelemente werden durch die CTL1 durch logische Verknüpfungen zu 128 L1-Subtriggern (*raw subtrigger* L1ST) kombiniert. Diese Verknüfungen erfolgen durch Wertetabellen (RAM *lookup tables*), wodurch größtmögliche Flexibilität gewährleistet ist. [45]

Um eine brauchbare Zusammenstellung von Ereignissen zu erhalten und die Gesamtrate von L1 zu beschränken, wird die Rate einzelner L1ST durch *prescaling* herabgesetzt. Dazu erhält jeder L1ST eine Zählervariable. Diese wird mit einem *prescale*-Wert initialisiert und bei jeder Erfüllung ihrer zugehörigen L1ST-Bedingung um 1 erniedrigt. Bei Erreichen der 0 wird die erfüllte L1ST-Bedingung weitergeleitet und die Zählervariable wieder mit dem *prescale*-Wert initialisiert.

Zusätzlich kann für jeden L1ST eine Fensterbedninung (*active gate*) aufgrund von Strahlinformation verlangt werden. So kann beispielsweise für einen Physiktrigger die Bedingung, daß beide Bündel des *bunch crossings* gefüllt sind, oder für eine Monitortrigger die Bedingung, daß Bündel nicht gefüllt sind, gesetzt werden.

Dieser Mechanismus war insbesondere in der Anfangsphase von HERA wichtig, da zu diesem Zeitpunkt nur ein geringer Bruchteil (ca. 10) der möglichen 210 Bündel gefüllt waren.

Um die Zeitentwicklung der Strahleigenschaften während einer Beschleunigerfüllung zu berücksichtigen und die Totzeit des Experiments möglichst gering zu halten, werden mehrere Mengen von Subtriggerbedingungen und *prescale*-Faktoren für die *run*-Phasen 1, 2, 3 und 4 gebildet.

Auf diese Weise werden die L1 *actual subtriggers* erzeugt.

artificial triggers [12, Seite 6] sind z. B. *random trigger* für Überwachungszwecke. Informationen über die für ein Ereignis gesetzten *artificial subtriggers* (*NIM1*, *NIM2*, *NIM13*, *rear input*, *software trigger*, *bunch trigger*, *random trigger*, *manual trigger*) befinden sich in der HEAR-Bank

3.1.3 L1keep

Falls das ODER der 128 *actual* L1ST erfüllt ist, wird das L1keep-Signal erzeugt.

Das L1keep-Signal wird zum *central trigger* gesendet, welcher für die Verteilung aller Zeit- und Kontrollsignale zu den Subsystemen verantwortlich ist.

Zu diesem Zeitpunkt werden die *front-end pipelines* angehalten und die Totzeit des Experiments beginnt.

3.1.4 Subdetektortrigger

Im Hinblick auf die Verwendung von Subdetektortriggerdaten als Eingabegrößen für den *background encapsulator* auf L2 (siehe Abschnitt 5.6) seien hier kurz die Subdetektortrigger z-Vertex-Trigger und DCRPHI-Trigger dargestellt.

z-Vertex Trigger Der z-Vertex-Trigger beruht vollständig auf den digitalisierten Signalen der *multiwire proportional chambers* (MWPC) (*central inner proportional chamber* (CIP), *central outer proportional chamber* (COP)) und der ersten Doppellage der *forward proportional chamber* (FPC)).

Während CIP und COP zylindrische Kammern um die Strahlachse mit Signaldrähten parallel zur Strahlachse, ist die FPC eine planare Kammer in Vorwärtsrichtung, die senkrecht zur Strahlachse ausgerichtet ist. Jede Kammer besteht aus zwei unabhängigen Lagen. Insgesamt werden 1920 Signale direkt im Detektor vorverstärkt, 35 m zum Elektroniktrailer außerhalb der H1-Experimentes weitergeleitet, geformt, diskriminiert und digitalisiert zu einer einheitlichen Breite von 96 ns und synchronisiert mit der globalen HERA-Uhr. Diese Signale gelangen in die z-Vertex-Triggerhardware, in der alle 96 ns eine volle Triggerentscheidung erzeugt wird. Das entspricht einem Datenfluß von 1.9 GByte/ s. Die digitalisierten und synchronisierten Signale werden in einer *pipeline* gespeichert. Wenn der *central trigger* das L1keep-Signal sendet, wird die *pipeline* gestoppt und ausgelesen.

Das Ziel des z-Vertex-Triggers ist die Onlinerekonstruktion der z-Koorindate des primären Wechselwirkungsvertexes. Diese wird durchgeführt durch Erzeugung von *rays* aus der Koinzidenz von vier (optional drei) Padsignalen, welche durch eine gerade, zur z-Achse zeigenden Linie verbunden werden können. Jedem *ray* wird auf eindeutige Weise entlang der z-Achse ein Bereich zugeordnet, welcher einen *bin* des z-Vertex-Histogramms definiert. Das z-Vertex-Histogramm besteht aus 16 *bins*,

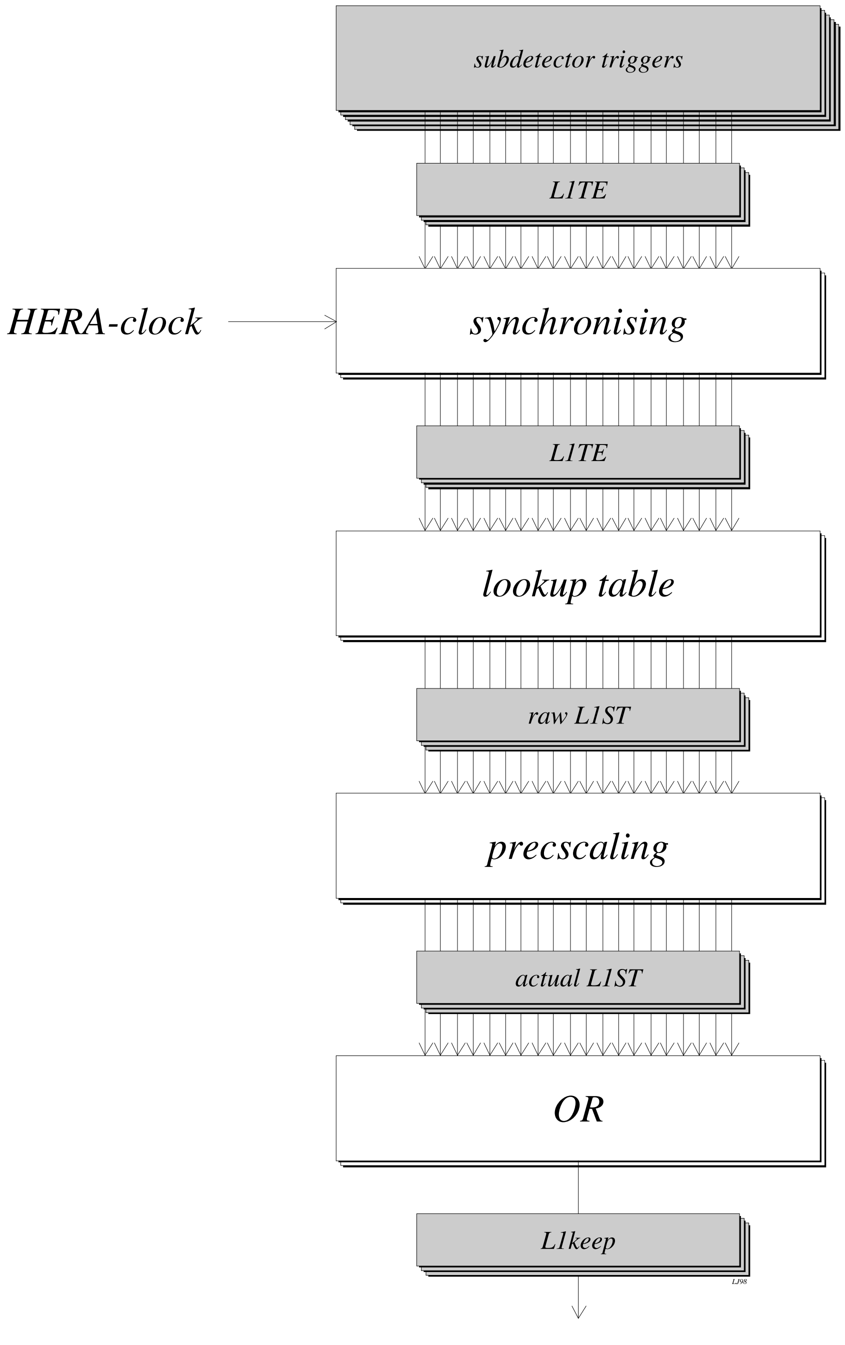

Abbildung 9: vereinfachte Darstellung der L1-Entscheidungsfindung

welche je 54.9 mm breit sind. So deckt das z-Vertex-Histogramm eine Bereich von 87.84 cm um den nominellen Wechselwirkungspunkt ab. Jeder *active ray* trägt mit einem Eintrag zum z-Vertex-Histogramm bei. Es wird erwartet, daß der *bin* mit den meisten Einträgen die Position des Wechselwirkungspunktes der Elektron-Proton-Kollision angibt.

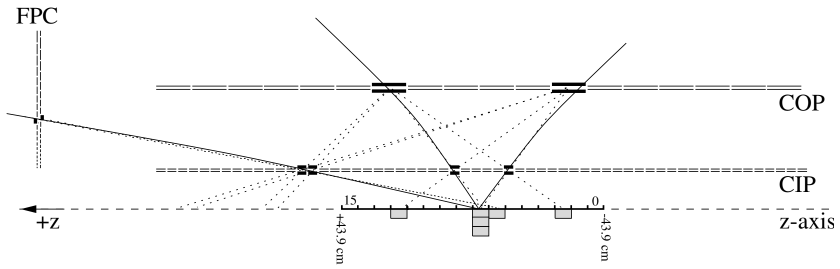

Abbildung 10: Prinzip des z-Vertex-Triggers in der rz-Ansicht: Eine gerade Spur durch 4 *pads* zur z-Achse trägt mit einem Eintrag zum z-Vertex-Histogramm bei.

Alle *active rays* werden mit einer 16-fachen Segmentierung in φ und einer 14-fachen Unterteilung in ϑ zu *big-ray*-Kandidaten in Gruppen zusammengefaßt. Die Unterteilung in (ϑ, φ) stimmt mit der Unterteilung des Kalorimeters in *big tower* überein.

Die *big-ray*-Information soll nach Vorverarbeitung durch *clustering*-Algorithmen der in der Entwicklung befindlichen DDB2 (siehe Abschnitt 3.2.6) aufbereitet den Netzen des L2 in Zukunft zur Verfügung gestellt werden.

Für die Triggerelemente des z-Vertex-Triggers und weitere Details sei auf [4] verwiesen.

DC$r\varphi$-Trigger Ziel des *drift chamber* $r\varphi$ *triggers* ist es, Spuren vom Wechselwirkungspunkt mit Transversalimpuls $p_t > 420$ MeV/ c in der $r\varphi$-Projektion zu erkennen und ihnen das korrekte *bunch crossing* zuzuordnen.

Dazu erhält der DC$r\varphi$-Trigger die Informatinen von 7 bzw. 3 Lagen der CJC1 bwz. CJC2. Diese Signale werden diskriminiert und synchronisiert mit einer Frequenz von 10.4 MHz bzw 20.8 MHz, um eine bessere Vertexschärfe zu erstellen. Bei einer typischen Driftgeschwindigkeit von 50 μs/ ns der Elektronen im Kammergas der CJC ergeben sich dadurch Driftabschnitte von 5 mm bzw. 2.5 mm. Auf diese Weise baut der Trigger ein digitales Bild der Kammer für die diskriminierten Signale auf und sucht durch Vergleich mit programmierten Masken (*roads*) nach Spuren.

Die für den DC$r\varphi$-Trigger für seine Entscheidungsfindung verbleibende Zeit nach Eintreffen des Signals mit der längsten Driftzeit von 1.1 μs beträgt 1 μs. Da der

Trigger aber für jedes *bunch crossing*, d. h. alle 96 ns eine Entscheidung treffen muß, hat die Entscheidungsfindung in einer „gepipelineten“ Art zu erfolgen.

Der DC$r\varphi$-Trigger bestimmt die Multiplizität der Spuren geladener Teilchen getrennt für positiv bwz. negativ geladene Teilchen in den Impulsbereichen $420\ \mathrm{MeV/\ c} < p_t < 800\ \mathrm{MeV/\ c}$ und $800\ \mathrm{MeV/\ c} < p_t$. Zusätzlich werden auch Entscheidungen zur Ereignistopologie in der $r\varphi$-Ebene erzeugt.

Seit 1993 ist φ in 45 Segmente unterteilt.

Um Triggerelemente für die CTL1 zu erhalten werden z. B. Schwellkriterien auf die verschiedenen Multiplizitäten angewendet. Für weitere Einzelheiten des DC$r\varphi$-Triggers, wie die Bestimmung von Referenzpunkten, den *football*-Algorithmus und die Bildung weiterer L1TE, sei auf [5, 44, 14, 34] verwiesen.

3.2 Level 2

Das L1keep-Signal wird durch die nachfolgenden Triggerstufen validiert.

Die maximale Entscheidungszeit für L2 beträgt 20 μs, welche aus der Forderung nach maximal 10% Totzeit bei Designluminosität resultiert.

Ein *bunch crossing* nach L1keep beginnt die Auslese der Eingabedaten für den L2-Trigger. Daher werden dem L2-Trigger die unvorverarbeiteten Daten der unabhängigen L1-Trigger-Subsysteme, wie z. B. den Spurkammern, Kalorimeter und Myonsystem, zur Verfügung gestellt (Tabelle 2) [26, 33].

Während auf der ersten Triggerstufe die Triggerentscheidung aus Koinzidenzen der lokalen Triggerbedingungen der einzelnen Subdetektoren gebildet wird, kann die zweite Triggerstufe diese Daten in lokaler Auflösung gemeinsam betrachten und somit korrelieren. Zusätzlich sind die L1-Subdetektortriggerentscheidungen und statistische Angaben (Spurzahl, Vertexhistogramm) des Ereignisses in L2 verfügbar. [18] Dadurch können schon frühzeitig gesuchte Ereignistopologien erkannt werden.

3.2.1 Triggerelemente des Level 2

Diese Daten werden von zwei unabhängigen *hardware*-Systemen (L2NN, L2TT) zu *raw level 2 triggerelements* (*raw* L2TE) verarbeitet. Ein drittes, in [12] erwähntes, System L2HH (Hamburg) wird zur Zeit nicht verwendet und ist ein Projekt für die Zukunft.

Die Systeme L2 *neural network* (L2NN)[2] und L2 *topological trigger* (L2TT)[3] erzeugen

[2] siehe Abschnitt 3.2.6
[3] siehe Abschnitt 3.2.7

MWPC	Vertexposition, (ϑ, φ)-Karte der *big-rays*
central drift chamber	45 φ-Sektoren für Spuren positiv/negativ geladener Teilchen mit hohem/niedrigem Impuls
SpaCal	globale Energie und Karten von Clustern
LAr-Kalorimeter	globale Energiesummen und *big tower*-Energien
Zentral- und Vorwärtsmyonsystem	Karte der Hits
central trigger	L1TE, L1ST, Kontrollinformation

Tabelle 2: Übersicht über die von den Subdetekoren an das L2-System gesendeten Daten

jeweils maximal 16 L2TE und senden diese zur *central level 2 decision logic* (CLT2). [49] Weitere 8 L2TE sind für das zusätliche L2HH-System vogesehen.

Die L2-Systeme müssen aktiv eine *accept* oder *reject* Entscheidung für jedes Triggerelement bilden. Dazu werden zusätliche *strobe*-Signale für die Validierung von L2TE zur CTL2 gesendet. Um die mögliche unterschiedliche Berechnungszeit der neuronalen Netze und damit die unterschiedliche Zeit der Entscheidungsübermittlung zu berücksichtigen, sendet das L2NN-System für jedes Triggerelement ein *strobe*-Signal. Da die Triggerelemente des L2TT-Systems gleichzeitig zur CTL2 gesendet werden, wird für das L2TT-System nur ein einziges *strobe*-Signale benötigt.

Die *level 2 trigger elements* gelangen zum *level 2 central trigger*. Dort wird die Zeitvalidierung überprüft. Falls ein Triggerelement nicht gültig ist, wird ein *default*-Wert eingesetzt. Die Setzung des *default*-Wertes kann auch erzwungen werden. Dadurch können fehlende oder verklemmte Triggerelemente angemessen behandelt werden. Die so erhaltenen *level 2 trigger elements* werden dann von der *level 2 central trigger logic* zu *unvalidated raw level 2 subtriggers* verarbeitet.

Die Triggerelemente müssen zusätzlich innerhalb des Zeitintervals von L1keep bis zwei *bunch crossings* vor dem Zeitpunkt, zu dem die L2-Entscheidung erwartet wird, bei der CTL2 eintreffen, um genügend Zeit für die weiteren Berechnungen zu haben.

Zusätzlich zu den durch *strobe* validierten Triggerelementen (*actual* L2TE) gibt es noch 8 *artificial outputs* (*artificial triggers* [12, Seite 12]), welche direkt über den Zustand interner Register gesetzt werden. Dadurch können *actual* L1ST auch ein L2keep liefern (siehe Abschnitt 3.2.4).

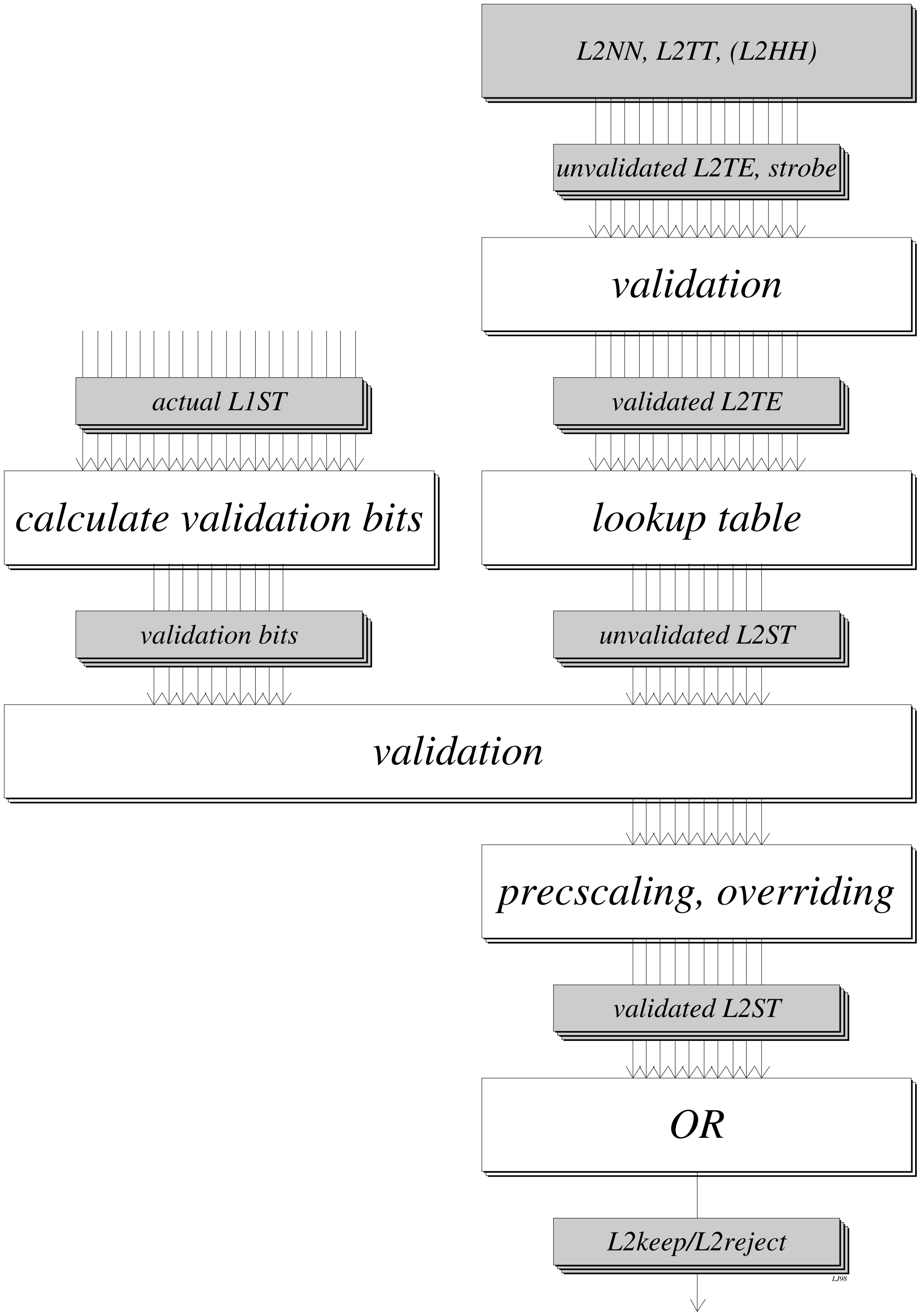

Abbildung 11: vereinfachte Darstellung der L2-Entscheidungsfindung

Anzahl	L2-System
16	L2NN
16	L2TT
8	L2HH
8	*artificial outputs*

Tabelle 3: Verteilung der L2TE auf die L2-Systeme

3.2.2 unvalidierte Subtrigger des Level 2

Aus den L2TE werden durch logische Verknüpfungen, welche durch Wertetabellen (*look up tables*) realisiert sind, 48 *unvalidated level 2 subtriggers* (L2ST) gebildet.

Die logischen Verknüpfungen von L2TE $\rightarrow$ L2ST sind für den L2NN trivial; beim L2TT werden sie stark genutzt.

3.2.3 *validation bits* für die Subtrigger des Level 2

Jedem L2ST ist eine Menge von L1ST zugeordnet („L1ST-Cocktail“). [46] Ein *unvalidated* L2ST wird zu einem *validated* L2ST, falls mindestens ein L1ST aus dem ihm zugeordneten L1ST-Cocktail gefeuert hat.

Seien mit S_j^1 der Vektor boolscher Variablen der *actual* L1ST (einschließlich der *artificial triggers* und der *active gates*) und mit M_{ij} die Matrix boolscher Variablen, welche die Zuordnung L2ST $\leftrightarrow$ L1ST-Cocktail enthält, bezeichnet, dann werden die 48 *validation bits* durch

$$V_i = \sum_j M_{ij} \cdot S_j^1 \quad (i \in \{0, \ldots, 47\})$$

bestimmt. [4]

Der *unvalidated* L2ST 40 liefert immer den Wert 1, d.h. die Entscheidung „ja“. Durch den Validierungsschritt können die L1ST des zugehörigen L1ST-Cocktails den Wert des *validated* L2ST 40 bestimmen. Von dieser Möglichkeit wird für L1ST Gebrauch gemacht, die keinem anderen L2ST zugeordnet sind und/oder deren Entscheidung „so gut ist“, daß diese L1ST auch ein L2keep liefern sollen.

Die Matrixmultiplikation ist der zeitraubendste Schritt der L2-Entscheidungsberechnung.

[4]Bei boolschen Operationen entsprechen Multiplikation und Addition dem logischen UND bzw. ODER

Für die 4 *excess bits* des Vektors V_i, welche auch Adreßleitungen der *look up tables* für die Bildung der *unvalidated* L2ST steuern können und somit L1-Subtriggern die Möglichkeit geben, die Erzeugung der *unvalidated* L2ST zu verändern, sei auf [33] verwiesen.

3.2.4 validierte Subtrigger des Level 2

Durch das logische UND aus *raw* L2ST und *validation bit* werden L2ST validiert.

Positive Entscheidungen können analog zum *prescale* von L1ST durch einen 8-*bit-count-down*-Zähler mit einem *prescale* versehen werden. Zusätzlich gibt es für negative Entscheidungen einen *override*-Zähler, so daß ein kontrollierter Bruchteil der Zurückweisungsentscheidungen für Überwachungszwecke übergangen werden kann.

Für jeden L2ST werden ein *prescale bit* P_i und ein *override bit* O_i zur Verfügung gestellt, die jeweils gesetzt sind, wenn der zugehörige Zähler den Wert 0 erreicht. Durch die logische Verknüpfung

$$\mathrm{L2ST}_i^{validated} = \mathrm{L2ST}_i^{unvalidated} \cdot V_i \cdot P_i + V_i \cdot O_i \quad (i \in \{0, \ldots, 47\})$$

erhält man die *validated* L2ST.

Ein gesetztes Level-2-Subtriggerbit kann ausgeschaltet werden durch *prescaling*. Ebenso kann es aber auch durch *override* angeschaltet werden. *Override* tritt immer dann auf, wenn der *override counter* Null erreicht hat. Dieser Zähler wird mit dem *override gap* initializiert und bei Erreichen der Null wieder auf dieses zurückgesetzt. Er wird jedesmal um Eins verringert, wenn das Level-2-Subtriggerbit aus ist.

3.2.5 L2keep, L2reject

Durch das logische ODER der *validated* L2ST wird die Entscheidung des L2-Triggers gebildet und ein L2keep- bzw. L2reject-Signal gesendet.

Das L2keep-Signal startet die Auslese der *front-end*-Daten.

Durch das L2reject-Signal werden die *pipelines* wieder in Betrieb genommen und die Totzeit endet.

3.2.6 *Level 2 Neural Network Trigger*

Der *level 2 neural network trigger* (L2NN) basiert auf mehreren (bis zu 16, derzeit 11) schnellen CNAPS/VME-Parallelrechnern, auf denen hauptsächlich *feed forward neural networks* (FFNN) (siehe Abschnitt 4.2.2) implementiert sind.

Die Triggerinformation der einzelnen Subdetektoren gelangt über den L2-Bus zu einer speziellen Bus-*interface*- und Vorverarbeitungskarte, dem sog. *data distribution board* (DDB). Durch dieses werden die Eingabedaten für den L2NN aufbereitet und in 8-Bit-Worten, den sog. L2-Größen [13], an den CNAPS-Parallelrechner weitergeleitet. Für die Datenvorverarbeitung stehen auf dem DDB einfache Algorithmen, wie die Auswahl von *high* oder *low byte* aus einem 16-Bit-Wort, die Summe von *high* und *low byte*, die Position eines Bits in einem Wort oder die Anwendung einer Wertetabelle, zur Verfügung. [26]

Eine Beschreibung der für diese Arbeit verwendeten, von dem DDB aufbereiteten „L2-Größen" befindet sich in Abschnitt 5.6.

Zur Zeit ist eine neue Vorverarbeitungskarte (DDB2) in Entwicklung, welche *clustering*-Algorithmen auf Bitfeldern bzw. Zahlenmappen anwenden können soll, um Jets aus Spurkammerdaten bzw. Kalorimeterdaten zu erkennen.

Die Kombination von je einem CNAPS-Parallelrechner und einem DDB wird *trigger box* genannt.

Derzeit sind im L2NN 11 von den maximal 16 *trigger boxes* eingebaut.

Bei der Implementation von FFNN auf einem CNAPS-Parallelrechner kann die Berechnung für jedes Neuron einer Schicht durch einen eigenen Prozessor übernommen werden. Die im H1-Experiment eingebauten CNAPS-Parallelrechner besitzen 64 Prozessorknoten. Für *offline*-Anwendungen (Training, Netzauswertungen, ...) stehen CNAPS-Parallelrechner mit 128 und 512 Knoten zur Verfügung. Die verwendbare Knotenzahl in der ersten Schicht der *online* Entscheidungsberechnung ist dadurch begrenzt, daß die Daten der ersten Schicht sequentiell zugeführt werden müssen.

Das Netzergebnis wird wieder zurück an das DDB und von diesem an die CTL2 (siehe Abschnitt 3.2.1) weitergeleitet.

Datenvorverarbeitung und Rechenzeit auf dem CNAPS-Parallelrechner dürfen zusammen nicht länger als 20 μs dauern.

Das Laden und die Kontrolle des L2NN erfolgt durch Software auf einer Themis VME-Sun *workstation*. Eine Beobachtung (*monitoring*) des Datenflusses innerhalb des L2-Systems wird durch die *spy card* ermöglicht.

3.2.7 *Level 2 Topological Trigger*

Der *level 2topological trigger* (L2TT) basiert auf einer 16×16 Matrix, welche die Detektorgeometrie in (ϑ, φ)-Koordinaten repräsentiert.

Die L2-Daten werden vorverarbeitet in boolsche Projektionen auf diese Matrix. Maximal 150 solcher Projektionen können während der L2-Entscheidungszeit vorverarbeitet und sequentiell analysiert werden. Jede dieser Projektionen wird auf topologische Eigenschaften untersucht. Für jede Projektion wird der „Abstand zum Untergrund“ für dieses Ereignis untersucht. Diese werde mit *look up tables* auf 16 sog. Maschinen, welche interessante Topologien repräsentieren, abgebildet. Durch den Vergleich der Summe der Abstände mit einer Schwelle für jede Maschine werden die L2TT-Triggerelemente erzeugt.

3.3 Level 3

Der L3-Trigger bildet seine Entscheidung aufgrund von *software*-Algorithmen, welche auf einem RISC-Prozessor laufen.

Aufgrund der langen Entscheidungszeit ist die Eingangsrate stark beschränkt. Deshalb wird L3 derzeit nicht für die Selektion von Physikdaten benutzt.

3.4 Level 4

Die Auslese und Formatierung der Daten der Subdetektorsysteme erfolgt parallel. Diese Daten werden dann in *multi-event buffers* gespeichert. Die formatierten Daten werden von der *central data aquisition* (CDAQ) geholt. Durch einen bestimmten Prozessor, dem sogenannten *event bilder*, werden die Daten für ein ganzes Ereignis zusammengestellt.

Ausreichendes *buffering* der Daten soll eine weitere Vergrößerung der Totzeit durch diesen Schritt vermeiden, vorausgesetzt die Bandbreite wird nicht überschritten.

Der L4-Trigger arbeitet auf diesen Daten asynchron. Dabei wird durch je einen der bis zu 36 RISC-Prozessoren der Prozessorfarm eine vereinfachte Version der *offline*-Rekonstruktion durchgeführt. Die Berechnungszeit ist von den Ereignisdaten abhängig, sollte aber für ein typisches Ereignis ca. 100 ns betragen.

Für Ereignisse mit vielen Signalen in den Spurkammern ist die Rekonstruktion von Spurbruchstücken erschwert, wodurch sich die Berechnungszeit verlängert. Falls diese zu groß wird, laufen die vorgeschalteten Puffer voll und die Totzeit des Experiments steigt.

Die durch L4 selektierten Ereignisse werden mit einer typischen Rate von 5 bis 10 Hz auf Band geschrieben (*raw data tapes*). Die Obergrenze von 10 Hz ist durch die Bandbreite der Übertragung auf den Massenspeicher von ca. 1.2 MByte bestimmt.

3.5 Level 5

Innerhalb weniger Tage, nachdem die Daten nach L4 auf Band geschrieben worden sind, werden sie einer vollständigen Rekonstruktion unterzogen. Aus den Detektordaten werden physikalische Objekte, wie Spuren aus Anpassungen von Helices an Energiedepositionen in den Spurkammern oder *clustern* aus der Zusammenfassung von benachbarten Kalorimeterzellen mit einer Energiedeposition, gebildet. Die Daten werden auf *physics output tapes* (POT1) geschrieben.

Die Ereignisse werde in Klassen (L5Class) eingeteilt (*offline*-Trigger Stufe 5).

Nach Datenkomprimierung werden diese Daten auf Festplatte für die Physikanalyse als *data selection tapes* (DST1) zur Verfügung gestellt.

Nach Abschluß der Datennahmeperiode werden mit teilweise verfeinerter Kalibrierung der Detektoren die Daten der *raw data tapes* noch einmal rekonstruiert und POT2 (POT3) bzw. DST2 (DST3) erzeugt.

3.6 *Transparent Runs*

Für Überwachungszwecke und Effizienzüberprüfungen von Triggern wird ein kleiner Bruchteil in der Größenordung von 1% der durch Trigger verworfenen Ereignisse dennoch auf Band gespeichert (vgl. den *override*-Mechanismus des L2-Triggers in Abschnitt 3.2.4).

Zusätzlich dazu gibt es spezielle sog. *transparent runs* bei denen die Entscheidung einer Triggerstufe nicht berücksichtigt wird, sondern alle Ereignisse von dieser Triggerstufe durchgelassen werden. Je nach „transparent" geschalteter Triggerstufe spricht man von L4-*transparent* bzw. L2/L4-*transparent*. Bei letzterem sind beide Triggerstufen transparent geschaltet. Alle Ereignisse eines L4-*transparent runs* nach L4 gelangen auch durch L5 auf POT, d.h. L4-*transparent* bedeutet auch L5-*transparent* [11].

Bei *transparent runs* spiegelt die Ereignisverteilung nach der transparent geschalteten Triggerstufe diejenige vor dieser wider. Dies ist insbesondere für die Selektion von Trainingsdaten für das Training der Netze des L2NN wichtig (siehe Abschnitt 5.2).

4 Neuronale Netze

Nor aine skild in Loupes of fingring fine,
Might in their divers cunning ever dare
With this so curios Networke to compare.
Edmund Spenser [3, Seite 193, Kapitel über Double- and Multiloop Knots]

Neuronale Netze bestehen aus Berechnungseinheiten (Knoten, Neuronen, *nodes*) und Verbindungen (Kanten, *edges*).

Die Knoten besitzen mehrere unabhänginge Eingänge, aber nur einen Ausgang, d. h. die Information fließt in eine Richtung. Der Informationstransport zwischen den Knoten erfolgt über gerichtete Informationskanäle, die Kanten.

Allgemein kann die Auswertung eines neuronalen Netzes durch eine Funktion

$$\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$$

mit $n, m \in \mathbb{N}$ beschrieben werden. Diese Funktion wird im weitere Netzausgaben genannt.

Neuronale Netze werden eingesetzt, wenn

1. die Berechnung der Funktion $\mathcal{N}$ zeitkritisch ist. Da die Auswertung neuronaler Netze hochgradig parallel erfolgen kann, ist durch sie eine sehr schnelle Berechnung von $\mathcal{N}$ möglich.
   Der Einsatz speziell für die Auswertung von FFNN ausgelegter *hardware*, wie z. B. dem CNAPS-Parallelrechner, ermöglicht Berechnungszeiten von wenigen Mikrosekunden.

2. die Funktion $\mathcal{N}$ unbegkannt ist und die Netzparameter nicht durch einen *fit* bestimmt werden können, da zu wenig Daten vorliegen. In diesem Bereich eignen sich neuronale Netze wegen ihrer Fähigkeit zur Generalisierung, d. h. daß gut trainierte Netze unbekannten Eingabedaten „vernünftig“ Ausgabewerte zuordnen. Dazu ist die Anzahl der freien Parameter zu beschränken, um ein Interpolieren zu vermeiden und Generalisierung zu ermöglichen.

3. unbekannte Daten klassifiziert werden sollen.

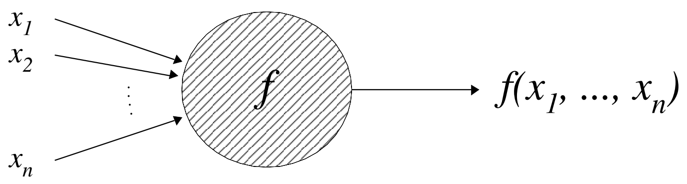

Abbildung 12: allgemeiner Knoten: $n \in \mathbb{N}$, $x \in \mathbb{R}^n$ und Aktivierungsfunktion $f : \mathbb{R}^n \to \mathbb{R}$

4.1 Knoten

Of Knots, it is necessary that I speak ...
A Naval Repository, 1762 [3, Seite 1]

Allgemein kann ein Knoten wie in Abbildung 12 durch eine Funktion

$$f : \mathbb{R}^n \to \mathbb{R} \quad \text{mit} n \in \mathbb{N}$$

beschrieben werden. f heißt Aktivierungsfunktion.

Eine übliche Unterteilung der Aktivierungsfunktion ist

$$f = \psi \circ \varphi,$$

wobei $\varphi : \mathbb{R}^n \to \mathbb{R}$ die Integrationsfunktion und $\psi : \mathbb{R} \to \mathbb{R}$ die Transfer- oder Ausgabefunktion ist.

Der Ausgabewert eines Neurons wird auch als seine Aktivierung bezeichnet.

Die Entwicklung verschiedener mathematischer Modelle (siehe z. B. [35]) für biologische Neuronen führte zum *einfachen Perzeptron*, dessen Integrationsfunktion sich wie folgt beschreiben läßt:

$$\varphi_1(x_1, \ldots, x_n) = \sum_{j=1}^{n} w_j x_j - \vartheta. \tag{1}$$

Dabei sind die Gewichte $w_1, \ldots, w_n \in \mathbb{R}$ und die Schwelle $\vartheta \in \mathbb{R}$.

Die Funktion φ_1 vermittelt eine Teilung des $\mathbb{R}^n$ in zwei „Halbräume“

$$\mathbb{H}_s := \{x \in \mathbb{R}^n | \varphi_1(x) = s\} \quad (s \in \{0,1\}),$$

die durch die Ebene $< x, w, - > \vartheta = \sum_{j=1}^{n} w_j x_j - \vartheta = 0$ mit dem Normalenvektor w gegeneinander abgegrenzt werden.

Angelehnt an die Biologie wird für die Ausgabefunktion die Stufenfunktion

$$\Theta(x) = \begin{cases} 0 & (x < 0) \\ 1 & (x > 0) \end{cases} // // \quad (x \in \mathbb{R} \setminus \{0\})$$

verwendet. Aus später ersichtlichen Gründen (siehe Abschnitt 4.3.1) und um die Anwendung eines nachträlichen Schwellenkriteriums zu ermöglichen, wird jedoch meist eine stetig differenzierbare Ausgabefunktion bevorzugt. Die Sigmoidfunktion

$$s(x) = \frac{1}{1 + \mathrm{e}^{-x}} \in (0,1) \quad (x \in \mathbb{R}) \tag{2}$$

hat zudem die schöne Eigenschaft, daß

$$s'(x) = s(x)(1 - s(x)) > 0 \quad (x \in \mathbb{R}). \tag{3}$$

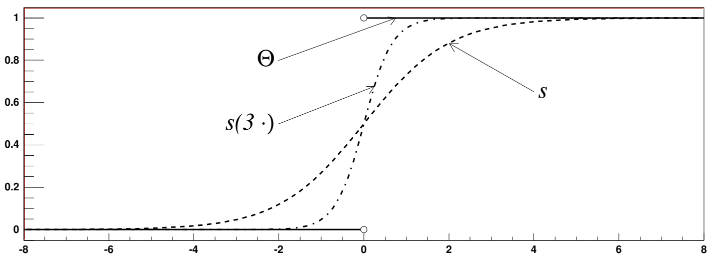

Abbildung 13: Sigmoidfunktion und Stufenfunktion

Die „Steilheit“ der Stufe $s(\beta\cdot)$ kann zusätzlich noch über den Parameter $\beta > 0$ gesteuert werden. β wird in Analogie zur Fermiverteilung als inverse Temperatur bezeichnet. Für den Grenzwert niedriger Temperatur geht die Sigmoidfunktion $s(\beta\cdot)$ fast überall in die Stufenfunktion über:

$$\lim_{\beta \to \infty} s(\beta x) = \Theta(x) \quad (x \neq 0).$$

Als Transferfunktion kann auch ein Vielfaches der Gaußfunktion

$$\varphi_{\text{Gauß}}(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \quad (x \in \mathbb{R})$$

verwendet werden. Mit Hilfe des Parameters $\sigma > 0$ kann die Breite variiert werden.

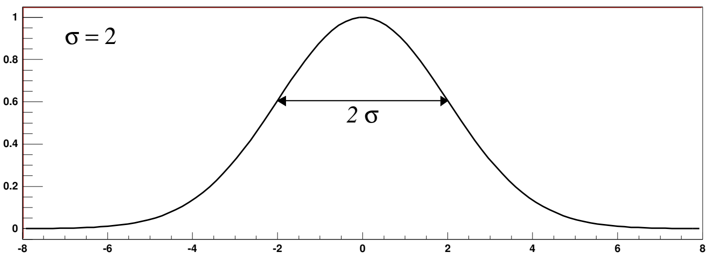

Abbildung 14: Gaußfunktion mit Standardabweichung σ

Für höhere Dimensionen wird in dieser Arbeit die Gaußglocke

$$\varphi_{\text{gg}}(x) = \prod_{j=1}^{n} \exp\left(-\frac{(x_j - w_{ij})^2}{2\sigma_j^2}\right) = \exp\left(-\sum_{j=1}^{n} \frac{(x_j - w_{ij})^2}{2\sigma_j^2}\right) \quad (x \in \mathbb{R}^n) \qquad (4)$$

mit Mittelpunkt $w \in \mathbb{R}^n$ und „Breite“ $\sigma \in \mathbb{R}^n$ verwendet.

4.2 Netztopologie

Neuronale Netze sind gerichtete Graphen bestehend aus Knoten und Kanten. Topologisch können zwei Typen unterschieden werden: Falls der Graph keine Zyklen enthält, wird das Netz *vorwärtsgerichtet* genannt, andernfalls *rekursiv*.

4.2.1 rekursive Netze

Bei rekursiven Netzen erhält also mindestens ein Knoten eine irgendwie geartete Weiterverarbeitung seiner Ausgabe als Eingabe. Für eine *software*-Simulation solcher Netze ist die Definition einer Zeiteinheit notwendig.

Da rekursive Netze im weiteren Verlauf der Arbeit nicht auftreten, sei im Hinblick auf diese auf die Literatur verwiesen [35].

4.2.2 vorwärtsgerichtete Netze

In vorwärtsgerichteten Netzen (*feed forward neural network*, FFNN) können die Knoten in einer Schichtarchitektur angeordnet sein. Dabei ist die Ausgabe eines Knotens einer Schicht mit je einem Eingabekanal jedes Knotens der nächsten Schicht verbunden. In Abbildung 15 ist die allgemeine Topologie eines vorwärtsgerichteten Netzes mit Schichtarchitektur dargestellt.

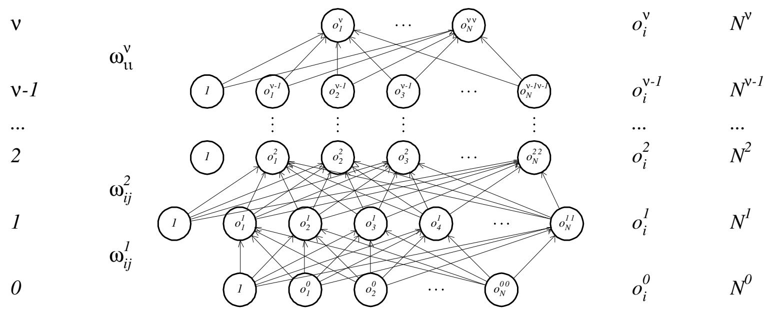

Abbildung 15: vorwärtsgerichtetes Netz mit Schichtarchitektur (FFNN): Mit ν sei die Anzahl der Schichten, mit N^j die Anzahl der Knoten innerhalb der Schicht j ($j \in \{1, \ldots, \nu\}$) bezeichnet. ω_{ij}^k heißt das Gewicht der Kante von Knoten i der Schicht k zu Knoten j der Schicht $k+1$. o_j^k ist die Aktivierung des Knoten j in der Schicht k.

Dieser Spezialfall eines zyklenfreien Netzes ist in der Anwendung von FFNN der am häufigsten anzutreffende Fall. Auch das menschliche Gehirn ist im wesentlichen ein fünf- bis zehnlagiges FFNN [23]. Für eine kurze Beschreibung des Aufbaus biologischer Neuronen aus Dendriten, Zellkörper und Axon, die elektro-chemische Signalleitung im Axon und den Informationstransport an den Synapsen durch chemische Transmitterstoffe sei auf [35] verwiesen. In [43] wird unter anderem auch auf den Aufbau und die Entwicklung des menschlichen Gehirns und den Prozess des Lernens eingegangen.

Bei den üblicherweise als FFNN bezeichneten Netzen ist die Aktivierung $o_j^k (k > 0)$ von der Form f_2 (siehe Abschnitt 4.1).

Die Schicht 0 wird als Eingabeschicht, die Schicht ν als Ausgabeschicht bezeichnet. Dazwischenliegende Schichten werden versteckte Schichten genannt.

Bei den in dieser Arbeit zur Anwendung kommenden Netze ist $\nu = 2$, o_j^1 von der Form f_3, $N^2 = 1$ und o_1^2 von der Form $s \circ \varphi_1$ (siehe Abbildung 16).

Die Motivation für diese beiden Spezialfälle wird im nächsten Abschnitt klar werden.

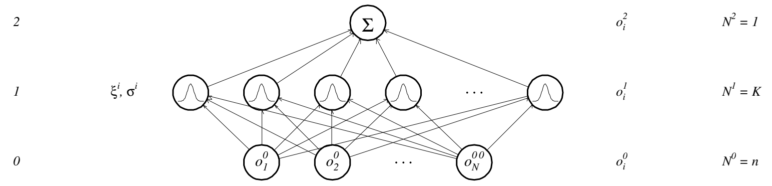

Abbildung 16: Aufbau des FFNN für den *background encapsulator*

4.3 Training

In diesem Abschnitt wird auf die Bestimmung der Netzparameter aufgrund von Daten eingegangen. Eine übliche Art die Gewichte und Parameter der Integrationsfunktion zu bestimmen ist Training. Trainingsdaten können in folgenden Formen vorliegen:

$$\mathbb{T}_A = \{(x^1, t^1), \ldots, (x^N, t^N)\} \subset \mathbb{R}^n \times \mathbb{R}^m$$

$$\mathbb{T}_B = \{x^1, \ldots, x^N\} \subset \mathbb{R}^n$$

Im Fall $\mathbb{T}_A$ ist das Ziel, die Netzparameter eines Netzes so zu bestimmen, daß die Netzausgabe $\mathcal{N}$ den Trainingsdaten entspricht; aber auch nicht in der Trainingsdatenmenge enthaltenen Daten, die diesen ähnliche Daten sind, sollen gut wiedergegeben werden. Das Netz soll fähig sein, zu generalisieren.

Im Fall $\mathbb{T}_B$ ist es Ziel, daß das Netz den „Bereich" der Trainingsdaten $\mathbb{T}_B$ erkennt, d. h. beispielsweise die Ausgabe

$$\begin{array}{ll} \mathcal{N}(x) \geq 0 & (x \in \text{„Bereich" von } \mathbb{T}_B) \\ \mathcal{N}(x) < 0 & (x \notin \text{„Bereich" von } \mathbb{T}_B) \end{array}$$

liefert.

Diese beiden sehr unterschiedlichen Problemstellungen, werden durch unterschiedliche Algorithmen behandelt.

Je nach Art, in der die Trainingsdaten vorliegen, eignen sich unüberwachte oder überwachte Trainingsalgorithmen. Einige dieser Algorithmen werden in den folgenden Abschnitten dargestellt.

4.3.1 Überwachtes Lernen

Hier liegen die Trainingsdaten in der Form

$$\mathbb{T} = \{(x^1, t^1), \ldots, (x^N, t^N)\} \subset \mathbb{R}^n \times \mathbb{R}^m$$

mit $n, m \in \mathbb{N}$ vor. Dabei ist t^j der Zielwert (*target*) für die Eingabe x^j ($j \in \{1, ..., N\}$).

Für diese Trainingsdaten kann die Netzausgabe $\mathcal{N}$ des zu trainierenden Netzes dur die Energiefunktion

$$E_{x,t}^{\mathcal{N}} := \frac{1}{2}|\mathcal{N}(x) - t|_2^2 \geq 0 \quad ((x,t) \in \mathbb{R}^n \times \mathbb{R}^m$$

bewertet werden. Je weiter die Energiefunktion $E_{x,t}^{\mathcal{N}}$ von der 0 abweicht, um so weiter weicht die Netzausgabe $\mathcal{N}$ von dem *target*-Wert t ab.

Das Ziel des Netztrainings, die Funktion $\mathcal{N}$ an die Trainingsdaten anzupassen. kann durch die Minimierung der nach unten beschränkten Energiefunktion $E_{x,t}^{\mathcal{N}}$ erreicht werden.

Sei die Netztopologie durch

$$\begin{aligned}
o_0^n &:= 1 \quad (n \in \{0, \ldots, \nu = 1\} \\
o_j^n &:= s(\sum_{\alpha=1}^{N^{n-1}} w_{\alpha j}^n o_j^{n-1} + w_{0j}^n 1) \\
&= s(\sum_{\alpha=0}^{N^{n-1}} w_{\alpha j}^n o_j^{n-1}) \quad (j \in \{1, \ldots, \mathbb{N}^n\}, n \in \{1, \ldots, \nu\})
\end{aligned}$$

gegeben (siehe Abbildung 15).

Sei $(x,t) \in \mathbb{T}$ für das folgende fest: Dann ist $/NE_w(x)$ als Funktione von w beliebig oft differenzierbar und damit auch $E_{x,t}(\cdot)$. Wegen

$$E_{x,t}(w + \Delta w) \approx E_{x,t}(w) + < \Delta w, \left(\frac{\partial E_{x,t}}{\partial w}\right)(w) > \quad \text{für kleines } |\Delta w|$$

ist

$$E_{x,t}(w - \left(\frac{\partial E_{x,t}}{\partial w}\right)(w)) \approx E_{x,t}(w) - \underbrace{< \left(\frac{\partial E_{x,t}}{\partial w}\right)(w), \left(\frac{\partial E_{x,t}}{\partial w}\right)(w) >}_{\geq 0} \quad \text{für kleines } |\Delta w|$$
$$\tilde{<} \quad E_{x,t}(w).$$

Dies legt eine schrittweise Gewichtsänderung der Form

$$w := w + \lambda \cdot \left(\frac{\partial E_{x,t}}{\partial w}\right)(w)$$

mit dem Lernparameter $\lambda_t > 0$ nahe, da dadurch bei geigneter Wahl von λ_t bei jedem Schritt $E_{x,t}(\cdot)$ verkleinert wird. Mit

$$\begin{aligned} \delta_l^\nu &:= -o_l^\nu(1 - o_l^\nu)(o_l^\nu - t_l) \\ \delta_l^n &:= -o_l^n(1 - o_l^n) \sum_{\beta=1}^{N^n} \delta_\beta^{n+1} w_{l\beta}^{n+1} \quad (n \in \{1, \ldots, \nu - 1\}) \end{aligned}$$

erhält man nach einfacher Rechnung[5]

$$\frac{\partial E_{x,t}}{\partial w_{kl}^n} = -\delta_l^n o_k^{n-1} \quad (n \in \{1, \ldots, \nu\}).$$

In Abbildung 17 ist das Trainingsergebnis für ein zweidimensionales Problem dargestellt.

Um das Hängenbleiben in lokalen Mininma zu vermeiden, gibt es mehrer heurisitische Ansätze. Beim Gewichtsänderungsschritt kann auch ein gewisser Anteil, der über den sogenannten Impulsparamter festgelegt wird, der vorhergehenden Gewichtsänderung berücksichtigt werden. Auch der Lernparameter λ kann mit der Zeit, d. h. der Zahl der Trainingsschritte, oder in Abhängigkeit von der Gewichtsänderung verändert werden. [35]

Der *backpropagation*-Algorithmus kann auf beliebige vorwärtsgerichtete Netze verallgemeinert werden. Dabei „merken“ sich die Neuronen ihre Aktivierung beim Auswertungsschritt. Durch rückwärtsgerichtetes Durchschreiten des Netzes kann die Änderung der Gewichte bestimmt werden, da die Sigmoidfunktion die Eigenschaft 3 hat. Näheres dazu siehe [35, Kapitel 7].

[5] welche man verkürzen kann, wenn man sich zuerzt die partiellen Ableitungen $\frac{\partial o_j^n}{\partial w_{kl}^m}$ mit $m \in \{0, \ldots, n\}$ vorbereitet.

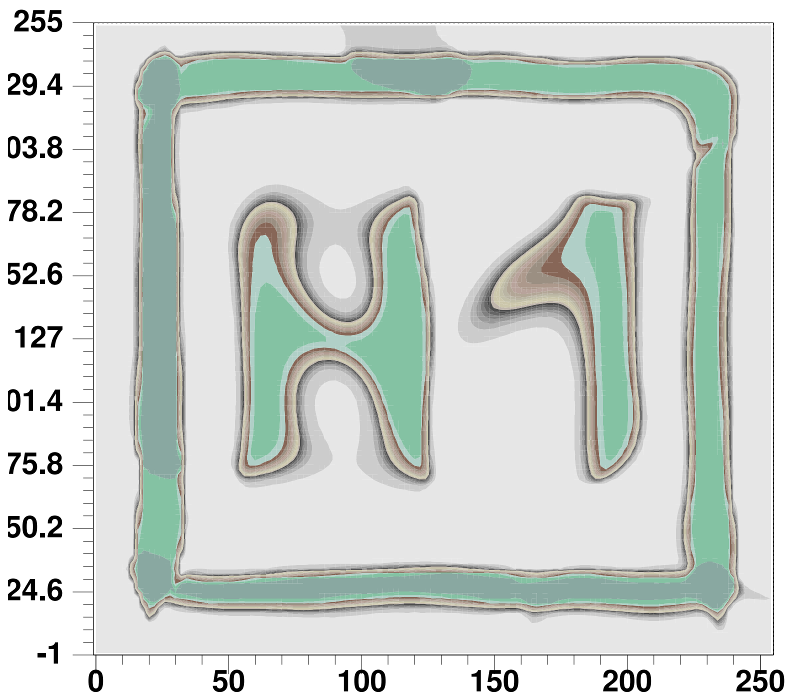

Abbildung 17: Netzausgabe eine vierlagigen FFNN, das mit dem *backpropagation*-Algorithmus auf dem CNAPS-Parallelrechner (Programm `bpexp2` [40, 41, 39]) trainiert wurde. Die Eingabeschicht enthält 2 Knoten, die beiden versteckten Schichten 30 bzw. 10 und die Ausgabeschicht 1 Knoten. Zum Training wurde Daten mit *taget*-Werte 255 bzw. 0 aus dem Bereich der Buchstaben und des Rahmens bzw. aus dem Komplement angeboten.

4.3.2 Unüberwachtes Lernen

Aus den vorliegenden Trainingsdaten

$$\mathbb{T} = \{x^1, \ldots, x^N\} \subset \mathbb{R}^n, N \in \mathbb{N}$$

sollen die Parameter des Netzes so bestimmt werden, daß für die Netzausgabe $\mathcal{N}$ für ein $c \in R$ gilt

$$\begin{array}{ll} \mathcal{N}(x) \geq c & (x \in \text{„Bereich" von } \mathbb{T}) \\ \mathcal{N}(x) < c & (x \notin \text{„Bereich" von } \mathbb{T}). \end{array}$$

In zwei (teilweise sogar noch in drei) Dimensionen ist eine Eingrenzung des „Bereichs" von $\mathbb{T}$ durch optisch, per Hand gesetzte Ebenen noch möglich. Dazu wird der „Bereich" von $\mathbb{T}$ in konvexe Teile zerlegt. Jeder von diesen kann dann durch Ebenen eingeschlossen werden. Jede dieser Ebenen kann durch einen Knoten mit Integrationsfunktion (1) [6] und der Stufenfunktion Θ realisiert werden. Diese Knoten bilden die erste Schicht des zu konstuierenden FFNN.

Die Entscheidung, ob ein Punkt innerhalb eines so durch Ebenen eingeschlossenen Bereichs liegt, ist ein logisches UND der Entscheidungen für die einzelnen Ebenen. Dieses UND kann durch Summation der Knotenentscheidungen (0 oder 1) für die Ebenen und Vergleich mit einer Schwelle, z. B. Knotenzahl-0.5, realisiert werden. Diese Berechnung kann aber wiederum durch einen Knoten mit Integrationsfunktion (1) und der Stufenfunktion Θ als Transferfunktion realisiert werden. Auf diese Weise erhält man die zweite Schicht des zu konstruierenden FFNN.

Um den ganzen „Bereich" von $\mathbb{T}$ zu erkennen, ist noch ein ODER aus den Entscheidungen für die einzelnen konvexen Teilbereiche zu berechnen. Dies ist durch Summation der Einzelentscheidungen für die konvexen Teilbereiche und Vergleich auf „ größer als 0" realisierbar; also wieder durch einen Knoten mit Integrationsfunktion (1) und der Stufenfunktion Θ als Transferfunktion. Der ODER-Knoten bildet die dritte Schicht des zu konstruierenden FFNN.

Im Fall von Trainingsdaten, die sich durch den Ausgabewert klassifizieren lassen, kann für jede dieser Klassen ein solches dreilagiges Netz konstruiert werden und anschließend deren Entscheidung (0 oder 1) mit dem für die entsprechende Klasse gewünschten Ausgabewert multipliziert werden. Da die Klassen disjunkt sind, ist

[6] Durch ihr Vorzeichen unterscheidet diese Integrationsfunktion den Halbraum, der die Trainigsdaten enthält, von dem, der sie nicht enthält.

höchstens eine Netzentscheidung der Teilnetze 1. Summation der gewichteten Teilentscheidungen ergibt also die gewünschte Gesamtentscheidung. Dieser letzte Schritt kann aber wiederum durch einen Knoten mit Integrationsfunktion (1) und der Stufenfunktion Θ als Transferfunktion realisiert werden, welcher die vierte Schicht des zu konstruierenden FFNN bildet.

Je nach Form können ganze konvexe Bereiche oder Teile davon durch Ellipsen beschrieben werden. In diesem Fall eignen sich Knoten mit Aktivierungsfunktion (4). Mit solchen Knoten können beschränkte konvexe Bereiche, für die in obigem Ansatz zwei Schichten nötig waren, in einer Schicht erkannt werden.

Die folgenden Konstruktionsschritte erfolgen analog.

Dieser zweite Ansatz hat den Vorteil, daß ganze konvexe Bereicht durch einen Knoten abgedeckt werden können.

Für eine Triggeranwendung ist auch die Beschränktheit des Gebiets, welches durch einen Knoten abgedeckt wird, wichtig. Hier werden im Gegensatz zur Eingrenzung durch Ebenen, unbegrenzte Bereiche ausgeschlossen.

Für die Eingrenzung durch Ebenen werden mindestens eine Ebene mehr als die Dimension des Raumes benötigt. Da bei der Realisierung des Auswertealgorithmuses auf dem CNAPS-Parallelrechner nur begrenzt viele Prozessorknoten (siehe Abschnitt 7) zur Verfügung stehen, ist die Zahl der realisierbaren Ebenen und damit die Zahl der konvexen Bereiche gering.

Deswegen wird im Weitern die zweite Methode mit den Knoten verwendet, die elliptischen Bereiche abdecken.

Daher kommt man zu einer FFNN-Topologie, wie sie in Abbildung 16 dargestellt ist.

Für die Bestimmung der Netzparameter (Mittelpunkte/Positionen und „Breiten“ der Gaußglocken) werden folgende Algorithmen verwendet:

`saf`	*simple and fast*
`kohonen`	Kohonenalgorithmus für ein- und zweidimenstionale Gitter
`gcs`	*growing cell structure*
`gng`	*growing neural gas*

Zuerst wird auf das Training der Positionen der Gaußglocken und anschließend auf das der „Breiten“ der Gaußglocken eingegangen.

4.3.3 Training der Knotenpositionen nach dem Altorithmus `saf`

Für die Erkennung des „Bereichs“ von $\mathbb{T}$ wird eine **feste** Anzahl K von Knoten verwendet. Ein Knoten k hat die Eigenschaft Position und „Breite“ der Gaußkurve

$$k = (\xi_1, \ldots, \xi_n, \sigma_1, \ldots, \sigma_n) \in \mathbb{R}^n \times \mathbb{R}^n.$$

Mit $\mathbb{K}$ sei die Menge der Knoten bezeichnet

$$\mathbb{K} := \{k^1, \ldots, k^n\}.$$

Die Menge der Knotenpositionen und die Menge der Trainingsdaten $\mathbb{T}$ sind Teilmengen eines beide umfassenden Raumes. Im Gegensatz dazu sind der Raum der Gewichte und der Raum der Trainingseingabedaten bei Netzen, die nach dem *backpropagation*-Algorithmus trainiert werden, im Allgemeinen verschieden.

Die Knotenpositionen werden nach dem Prinzip *winner takes all* trainiert. Zu einem Trainingsdatum $x \in \mathbb{T}$ wird der nächstliegende Knoten k^{W} (*winner*)

$$|k^{\mathrm{W}}.\xi - x| \leq |k.\xi - x| \quad (k \in \mathbb{K})$$

bestimmt. Dieser Knoten wird dann in Richtung des Trainingsdatums x verschoben. Die Stärke dieser Verschiebung wird über den Lernparameter $\alpha \geq 0$ bestimmt.[7]

$$k^{\mathrm{W}}_{(t+1)}.\xi = k^{\mathrm{W}}_{(t)}.\xi + \alpha(x - k^{\mathrm{W}}_{(t)}.\xi)$$

Eine bezüglich der Trainingsdaten günstige Lage eines Knotens wird durch einen Trainingsschritt dieser Art verbessert. Daher spricht man auch von Lernen durch Verstärkung.

Ziel ist es, durch wiederholte Anwendung dieses Trainingsschrittes die Knotenpositionen in die Zentren von Anhäufungen (*cluster*) der Trainingsdaten zu bewegen (Clusteringalgorithmen).

Ein Nachteil dieses einfachen Trainingsschrittes ist, daß von den Trainingsdaten weit entfernte Knoten niemals *winner* werden und dadurch untrainiert bleiben.

Für den Algorithmus `saf` wird jedem Knoten zusätzlich noch ein Zähler hinzugefügt

[7] Im folgenden wird eine Pseudonotation zwischen mathematischer und programmiersprachlicher (C++) verwendet. Der Punkt . wird verwendet um auf Komponenten zuzugreifen. Um Verwechslungen zu vermeiden wird das Skalarprodukt durch $< \cdot, \cdot >$ dargestellt. Der Allquantor $\bigwedge$ wird im Folgenden teilweise nicht mathematisch, logisch sondern im Sinn einer `for`-Schleife angewendet. Gleichsam ist die Implikation $\Longrightarrow$ nicht logisch, sondern im Sinne von `if ...then ...` zu verstehen. Mathematisch exakt müßten zusätzlich Zeitindizes verwendet werden. Der Übersichtlichkeit halber bleiben diese teilweise weg.

$$k^{\text{saf}} = (\xi, \sigma, \text{counter}) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{N}_0,$$

der die Anzahl der diesen Knoten betreffenden Trainingsschritte, d. h. die Anzahl der der Trainingsschritte, bei denen dieser Knoten als *winner* hervorging, speichert. Nach einer einstellbaren Anzahl von Trainingsschritten wird jedem Knoten $k \in \mathbb{K}$ mit $k.\text{counter} \leq \text{Schwelle}$ eine zufällige neue Position $k.\xi$ zugewiesen.

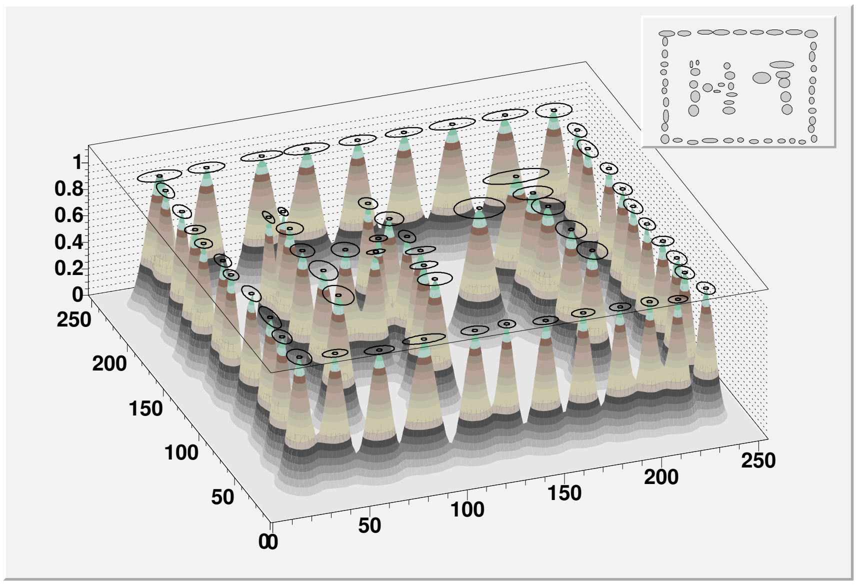

Abbildung 18: H1-Beispiel trainiert mit `saf`

Eine weitere Möglichkeit, untrainierte Knoten zu vermeiden, ist die Einführung von Nachbarschaftsbeziehungen. Von dieser wird bei den folgenden Algorithmen Gebrauch gemacht.

4.3.4 Training der Knotenpositionen nach dem Altorithmus `kohonen`

Dem Kohonenalgorithmus liegt eine **feste** Anzahl K von Knoten der Form

$$k = (\text{id}, \xi, \dots) \in \mathbb{N}_0 \times \mathbb{R}^n \times \dots$$

zugrunde. Mit $\text{id} \in \mathbb{N}_0$ sei jedem Knoten eine eindeutige Nummer zugeordnet, so daß Knoten über diese Nummer identifiziert werden können.[8] Für diese Knoten wird

[8]Für Algorithmen mit fester Knotenzahl kann einfach $k^i.\text{id} := k^i (i \in \{1, \dots, K\})$ gesetzt werden. Die Einführung von id ist bei Algorithmen mit zeitlich veränderlicher Knotenzahl wichtig. Um eine einheitliche Notation verwenden zu können, wird id schon hier eingeführt.

eine Nachbarschaftsbeziehung über ein ein-, zwei- oder mehrdimensionales Gitter vermittelt.

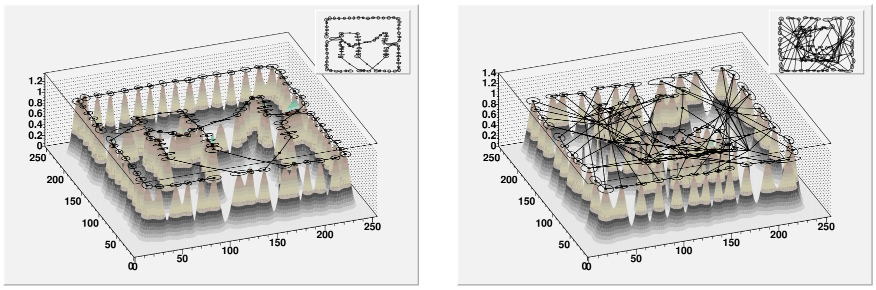

Abbildung 19: Beispiele kohonen-1, kohonen-2: Kohonenalgorithmus mit ein- und zweidimensionalen Gitter auf die Trainingsdaten des H1-Beispieles angewendet

Diese Gitter können unabhängig von ihrer Dimension durch eine Menge[9] $\mathbb{E}$ von Kanten (*edges*) realisiert werden. Eine Kante

$$e = \{k^0.\mathrm{id}, k^1.\mathrm{id}\} \in \mathbb{E}$$

wird durch die Identifikationsnummern der durch sie verbundenen Knoten $k^0, k^1 \in \mathbb{K}$ bestimmt. Durch diese Kanten können Nachbarschaftsbeziehungen erklärt werden. Seien $k^0, k^1 \in \mathbb{K}$, dann ist durch

$$a(k^0, k^1) := \min\{m \in \mathbb{N}_0 \quad | \quad \begin{array}{l}\text{Knoten } k^1 \text{ kann unter Verwendung von } m \\ \text{Kanten von Knoten } k^0 \text{ erreicht werden.}\}\end{array}$$

die Entfernung einer Nachbarschaftsbeziehung definiert. Es gilt:

$$a(k^0, k^1) = a(k^1, k^0) \quad (k^0, k^1 \in \mathbb{K})$$

$$a(k^0, k^1) = 0 \Longrightarrow k^0 = k^1 \quad (k^0, k^1 \in \mathbb{K})$$

[9] Die Formulierung der Algorithmen mit Mengenschreibeweise spiegelt sich auch im Quellcode nieder. Siehe hierzu auch den Klassenbaum in Anhang A.4.

Knoten $k^0, k^1 \in \mathbb{K}$ mit $a(k^0, k^1) = 1$ heißen nächste Nachbarn.

Im Trainingsschritt wird zu einem Trainingsdatum $x \in \mathbb{T}$ der *winner*-Knoten $k^{\mathrm{W}} \in \mathbb{K}$ bestimmt. Zusätzlich zum Lernparameter $\alpha^{\mathrm{W}} = \alpha^0 \geq 0$ für den *winner*-Knoten werden weitere Lernparameter $\alpha^b \geq 0$ $(b \in \mathbb{N})$ für die Knoten mit Nachbarschaftsgrad b eingeführt. Der Trainingsschritt für die Knotenpositionen ist

$$k_{(t+1)}.\xi = k_{(t)}.\xi + \alpha^{a(k^{\mathrm{W}},k)}(x - k_{(t)}.\xi) \quad (k \in \mathbb{K}).$$

Um eine die Topologoie der Trainingsdaten wiederspliegelnde Topologie in den Knoten zu erhalten, d. h. um zu erreichen, daß räumlich nahe beieinander liegende Knoten auch nahe bezüglich der Nachbarschaftsbeziehungen beieinander liegen, ist

$$\alpha^b \geq \alpha^{b+1} \quad (b \in \mathbb{N}_0)$$

zu wählen (*range decreasing*). Für „großes“ b wird meist $\alpha^b = 0$ gewählt.

Um anfänglich größere Positionsänderungen bei den Trainingschritten zu erhalten und später nur noch kleinere Korrekturen anzubringen, werden die Lernparameter zeitabhängig gewählt: $\alpha^b(t)$, wobei t den Trainingsschritt angibt. (*cooling schedule*)

Nachteile des Kohonenalgorithmus liegen in

1. der festen Anzahl der Knoten,
2. der festen Gittertopologie,
3. der mit der Dimension d exponentiell steigende Bedarf 2^d an Kanten für einen „inneren“ Knoten.

Insbesondere die feste Gittertopologie bereitet Schwierigkeiten bei aus mehreren getrennten Clustern bestehenden Trainingsdaten, da hier lange, clusterübergreifende Kanten nötig werden. Dazu muß jedoch der Lernparameter für die Nachbarn stark verkleinert werden, so daß der Vorteil des Kohonenalgorithmus, der in den Nachbarschaftsbeziehungen besteht, unterdrückt wird.

Die Dimension des Gitters muß nicht mit der Dimenstion der Trainingsdaten übereinstimmen. Durch Wählen einer niedrigeren Dimension kann durch das Training eine Dimensionsreduktion durchgeführt werden.

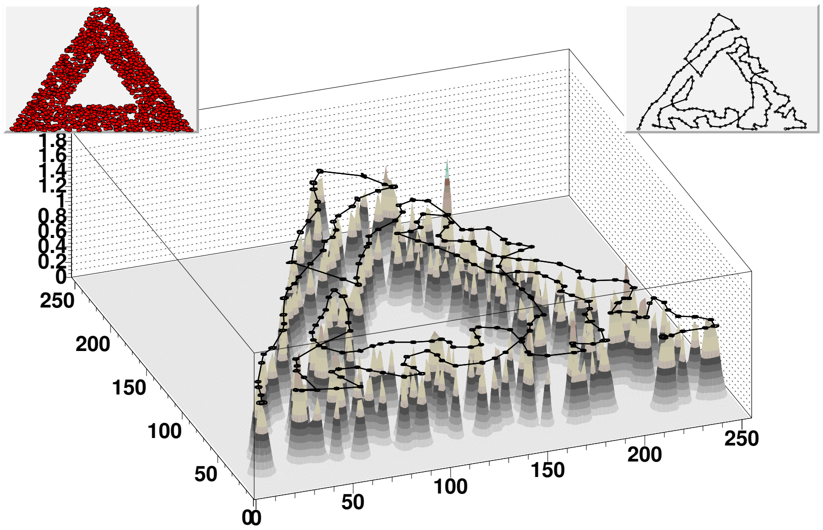

Abbildung 20: Kohonen 1 Trainingsdaten (links oben) aus dem Gebiet eines Dreieckrahmens: Reduktion der Dimension

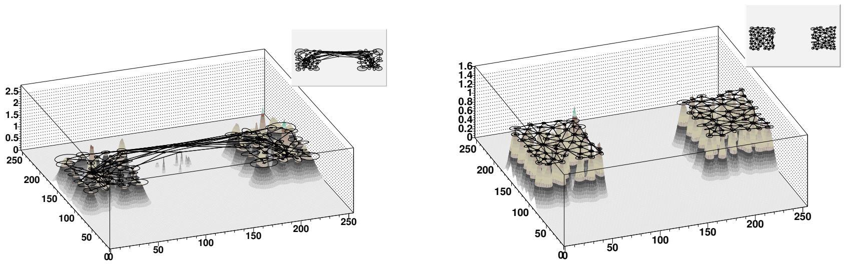

Abbildung 21: Algorithmus `kohonen` mit zweidimensionalen Gitter und Algorithmus `gcs` angewendet auf Trainingsdaten, die aus zwei getrennten Clustern bestehen

4.3.5 Training der Knotenpositionen nach dem Altorithmus gcs

Beim Algorithmus `gcs`, der sich an den Algorithmus *growing cell structure* [37, 28, 15] anlehnt, werden die Eigenschaften — Nachbarschaftsbeziehung und Training der Nachbarn — des Kohonenalgorithmuses durch eine sich zeitlich ändernde und durch das Training bestimmte Knotenzahl sowie durch eine andere Topologie ergänzt. Die Knoten sind nicht mehr in einem topologisch starren, kubischen Gitter angeordnet, sondern befinden sich an den Ecken von Simplizes.

Die Nachbarschaftsbeziehung zwischen den Knoten wird über die Kanten der Simplizes definiert. Da Knoten mehreren Simplizes angehören können, können auf diese Weise Strukturen aufgebaut werden. Ein Zusammenhang dieser Strukturen ist nicht zwingend, so daß dieser Algorithmus mehrere in sich zusammenhängende und untereinander nicht zuammenhängende Strukturen für aus mehreren Clustern bestehende Trainingsdaten aufbauen kann.

Die für diesen Algorithmus verwendeten Knoten sind von der Form

$$k = (\mathrm{id}, \xi, \sigma, \mathrm{counter}) \in \mathbb{N}_0 \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}.$$

Für die Repräsentation der Simplexstruktur bzw. der Nachbarschaftsbeziehungen wird die Menge der Simplices $\mathbb{S}$ definiert ($d \in \mathbb{N}$) :

$$\mathbb{S} = \{\{k^1.\mathrm{id}, \ldots, k^d.\mathrm{id}\}, \ldots\}.$$

Wie beim `kohonen`-Algorithmus muß auch hier die Dimension der Simplices nicht mit der Dimension der Trainingsdaten übereinstimmen. Bei Wahl einer niedrigeren Dimension kann durch das Training eine Dimensionsreduktion erreicht werden.

Aus der Menge der Simplices läßt sich die Menge der Kanten

$$\mathbb{E} = \bigcup_{s \in \mathbb{S}} \bigcup_{i \in \mathbb{S}} \bigcup_{\substack{i \in \mathbb{S} \\ i \neq j}} \{\{i\} \cup \{j\}\}$$

gewinnen.

Der `gcs`-Algorithmus startet mit einem Simplex, dessen $d+1$ verschiedenen Knoten zufällig positioniert sind.

Im Trainingsschritt für ein Trainingsdatum $x \in \mathbb{T}$ wird für die Struktur von Simplizes der *winner*-Knoten $k^{\mathrm{W}} \in \mathbb{K}$ bestimmt. Die Knotenpositionen und counter werden folgendermaßen trainiert:

	Ecken	Kanten
Simplex	$d+1$	$\frac{(d+1)d}{2}$
Würfel	2^d	$d2^{d-1}$

Tabelle 4: Das exponentielle Anwachsen der Ecken- bzw. Kantenzahl beim Würfel steht dem linearen bzw. quadratischen beim Simplex gegenüber. (Dimension $d \in \mathbb{N}$)

$$\begin{aligned}
k^{\mathrm{w}}.\xi_{(t+1)} &= k^{\mathrm{w}}.\xi_{(t)} + \alpha^{\mathrm{w}} \cdot (x - k^{\mathrm{w}}.\xi_{(t)}) \\
k^{\mathrm{n}}.\xi_{(t+1)} &= k^{\mathrm{n}}.\xi_{(t)} + \alpha^{\mathrm{n}} \cdot (x - k^{\mathrm{n}}.\xi_{(t)}) \quad (k^{\mathrm{n}} \in \mathbb{K} \text{ mit } a(k^{\mathrm{w}}, k^{\mathrm{n}}) = 1) \\
k^{\mathrm{w}}.\mathrm{counter}_{(t+1)} &= k^{\mathrm{w}}.\mathrm{counter}_{(t)} + 1 \\
k.\mathrm{counter}_{(t+1)} &= \beta \cdot k.\mathrm{counter}_{(t)} \quad (k \in \mathbb{K})
\end{aligned}$$

Mit den Lernparameteren $\alpha^{\mathrm{w}}, \alpha^{\mathrm{n}} \geq 0$ und der Zerfallskonstante $\beta \in (0,1)$. Der counter einer Knoten soll die Information enthalten, wie oft dieser Knoten „in letzter Zeit" *winner* geworden ist. Um lang überholte counter-Information aus lange zurückliegenden Trainingsschritten „aussterben" zu lassen, wird die Zählerzerfallskonstante β verwendet.

Wenn ein Knoten $k^0 \in \mathbb{K}$ besonders häufig *winner* wird, d. h. sein counter eine festzulegende Schwelle überschreitet, bietet es sich an, an dieser Stelle einen weiteren Knoten einzusetzen, da hier besonders viele Trainingsdaten vorliegen. Um die „Spannung" der langen Kante zum enferntesten topologischen Nachbarknoten $k^1 \in \mathbb{K}$ zu verringern, bietet es sich an, den neuen Knoten k^n auf der Hälfte dieser Kante einzusetzen:

$$k^n_{(t+1)}.\xi = \frac{1}{2}(k^0_{(t)}.\xi + k^1_{(t)}.\xi).$$

Die counter der drei Knoten k^0, k^1, k^n werden aufgeteilt:

$$k^0_{(t+1)}.\mathrm{counter} = k^1_{(t+1)}.\mathrm{counter} = k^n_{(t+1)}.\mathrm{counter} = \frac{1}{3}(k^0_{(t)}.\mathrm{counter} + k^1_{(t)}.\mathrm{counter}).$$

Um die Simplexstruktur beizubehalten müssen alle Simplices, die k^0 und k^1 zu ihren Eckpunkten zählen, geteilt werden:

$$\begin{aligned}
&\bigwedge_{s \in \mathbb{S}} ((k^0.\mathrm{id} \in s \wedge k^1.\mathrm{id} \in s) \Longrightarrow \\
&\quad (\mathbb{S}_{(t+1)} = (\mathbb{S}_{(t)} \setminus s) \cup \{\{(s \setminus k^0.\mathrm{id}) \cup k^n.\mathrm{id}\}\} \cup \{\{(s \setminus k^1.\mathrm{id}) \cup k^n.\mathrm{id}\}\}))
\end{aligned}$$

$$\mathbb{K}_{(t+1)} = \mathbb{K}_{(t)} \cup \{k^n\}$$

Auf diese Weise ist ein Anwachsen der Knotenzahl möglich.

Wenn jedoch ein Knoten $k^0 \in \mathbb{K}$ „in letzter Zeit“ nicht trainiert worden ist, also durch den Zerfallsprozess sein counter unter eine Schwelle gefallen ist, kann der Knoten durch folgenden Mechanismus entfernt werden: Durch das Entfernen des Knotens k^0 werden alle Simplizes $s \in \mathbb{S}$ mit $k^0.\mathrm{id} \in s$ „ungültig“, d. h. die Simplexstruktur wird zerstört. Diese Simplizes müssen ebenfalls entfernt werden. Dadurch können aber Knoten entstehen, die nicht mehr über Kanten mit anderen Knoten verbunden sind. Um die Simplexstruktur zu bewahren, müssen diese Knoten ebenfalls entfernt werden. Dies ist in folgenden Formeln zusammengefaßt:

$$\begin{aligned}
\mathbb{S}_{(t+1)} &= \mathbb{S}_{(t)} \setminus \underbrace{\{s \in \mathbb{S}_{(t)} | k^0.\mathrm{id} \in s\}}_{=:\mathbb{D}_{(t)}} \\
\mathbb{K}_{(t+1)} &= \mathbb{K}_{(t)} \setminus \left(\left\{ k \in \mathbb{K}_{(t)} | \bigvee_{s \in \mathbb{D}_{(t)}} (k.\mathrm{id} \in s) \right\} \setminus \left\{ k \in \mathbb{K}_{(t)} | \bigvee_{s \in \mathbb{S}_{(t)} \setminus \mathbb{D}_{(t)} = \mathbb{S}_{(t+1)}} (k.\mathrm{id} \in s) \right\} \right) \\
&= \left\{ k \in \mathbb{K}_{(t)} | \bigvee_{s \in \mathbb{S}_{(t+1)}} (k.\mathrm{id} \in s) \right\}
\end{aligned}$$

Auf diese Weise wird ein Abnehmen der Knotenzahl realisiert. Durch die Entfernung ganzer Simplizes kann die Simplexstruktur in mehrere, nicht zusammenhängende Strukturen aufbrechen und so getrennte Cluster der Trainingsdaten abdecken.

Ein Zusammenwachsen von einmal getrennten Simplexstrukturen ist jedoch mit diesem Algorithmus nicht möglich. Nur ein Ausdehnen der einen und Aussterben einer anderen Simplexstruktur kann diesen Vorgang nachahmen.

Die Möglichkeit neue Verbindugen aufzubauen, wird im folgenden Algorithmus verwendet.

4.3.6 Training der Knotenpositionen nach dem Altorithmus gng

Der an den Algorithmus *growin neural gas* [28] angelehnte Algorithmus gng baut wieder auf Knoten und Kanten auf. Die Kanten vermitteln die Nachbarschaftsbeziehung. Kanten werden zwischen *winner* und dem nächsten Nachbarn eingebaut.

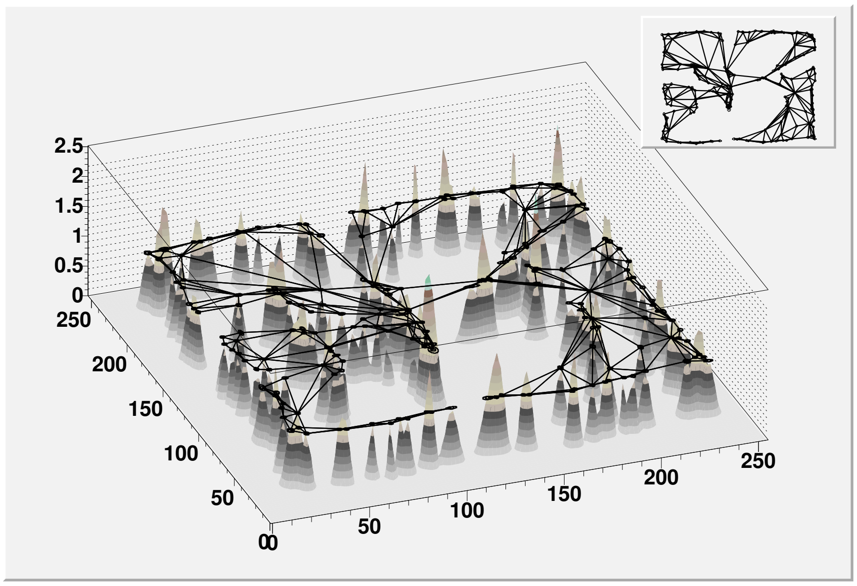

Abbildung 22: gcs-Algorithmus auf die H1-Beispieltrainingsdaten angewendet

Gegenüber den strukturbildenden Elementen des Kohonen- oder gcs-Algorithums haben die Kanten zusätzlich die Eigenschaft des Alters. D. h. falls eine Nachbarschaftsbeziehung zwischen zwei Knoten nicht mehr vorliegt, altert die Kante und stirbt bei Ereichen eines Schwellalters aus. So entstandene unverbundene Knoten werden entfernt, wodurch eine Abnahme der Knotenzahl realisiert wird.

Eine Zunahme der Knotenzahl wird, wie beim gcs-Algorithmus über ein Schwellkriterum an den counter eines Knotens implementiert. Die Kante zum örtlich weitest enfernten, topologischen nächsten Nachbarn einens die counter-Schwelle überschreitenden Knoten wird gespaltet und ein neuer Knoten eingesetzt. Die counter der drei beteiligten Knoten werden wie beim gcs-Algorithmus auf die Knoten verteilt.

Durch das Einsetzen von neuen Kanten auf diese Weise kann sich die topologische Struktur der lokalen Dimension der Trainingsdaten anpassen.

Der Algorithmus startet mit zwei zufällig positionierten Knoten und einer Kante zwischen diesen beiden.

Beim Trainingsschritt wird zu einem Trainingsdatum $x \in \mathbb{T}$ der *winner*-Knoten $k^{\mathrm{W}} \in \mathbb{K}$ und der örtlich zweitnächste Knoten $k^{\mathrm{S}} \in \mathbb{K}$ bestimmt. Falls die Kante $e = \{k^{\mathrm{W}}.\mathrm{id}, k^{\mathrm{S}}.\mathrm{id}\}$ in $\mathbb{E}$ enthalten ist, wird deren Alter auf 0 gesetzt, andernfalls wird eine solche Kante mit Alter 0 erzeugt. Anschließend wird das Alter aller Kanten ausgehend vom *winner*-Knoten, $e \in \mathbb{E}$ mit $k^{\mathrm{W}}.\mathrm{id} \in e$, um 1 erhöht.

Kanten, deren Alter eine Schwelle überschreitet, werden entfernt, wie auch die durch das Entfernen der Kante entstandenen unverbundenen Knoten.

Der Zähler des *winner*-Knotens wird erhöht:

$$k^{\mathrm{w}}.\mathrm{counter}_{(t+1)} = k^{\mathrm{w}}.\mathrm{counter}_{(t)} + 1$$

Falls dieser Zähler eine Schwelle überschreitet, wird nach der oben beschriebener Methode ein neuer Knoten eingesetzt.

Anschließend wird der Zerfallsschritt für die Zählervariable der Knoten durchgeführt:

$$k.\mathrm{counter}_{(t+1)} = \beta k.\mathrm{counter}_{(t)} \quad (k \in \mathbb{K})$$

und das Training für die Knotenposition des *winner*-Knoten und seiner nächsten topologischen Nachbarn durchgeführt:

$$\begin{aligned} k^{\mathrm{w}}.\xi_{(t+1)} &= k^{\mathrm{w}}.\xi_{(t)} + \alpha^{\mathrm{w}}(x - k^{\mathrm{w}}.\xi_{(t)}) \\ k^{\mathrm{n}}.\xi_{(t+1)} &= k^{\mathrm{n}}.\xi_{(t)} + \alpha^{\mathrm{n}}(x - k^{\mathrm{n}}.\xi_{(t)}) \quad (k^{\mathrm{n}} \in \mathbb{K} \text{ mit } a(k^{\mathrm{w}}, k^{\mathrm{n}}) = 1). \end{aligned}$$

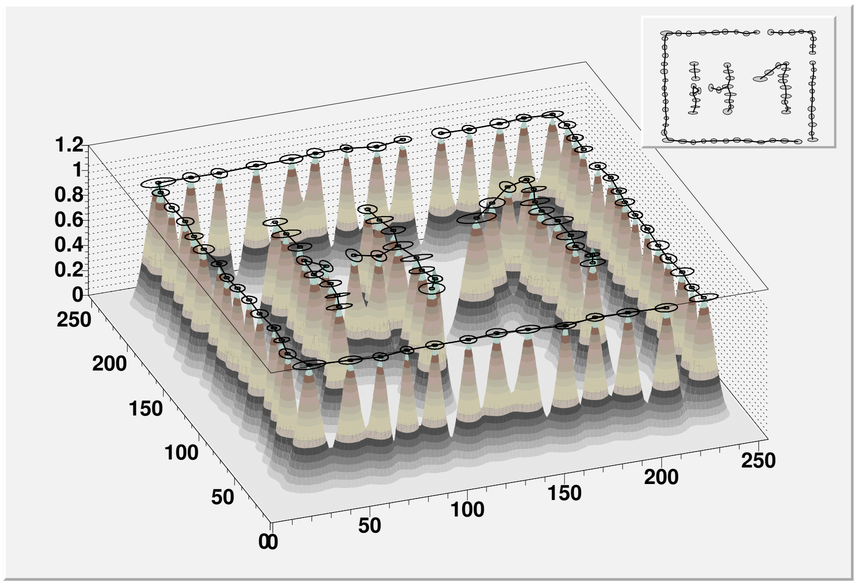

Abbildung 23: gng-Algorithmus auf die H1-Beispieltrainingsdaten angewendet

4.3.7 Training der Knotenausdehung

Bislang wurde nur das Training der Knotenpositionen betrachtet. Für das Trainig der Breiten der Gaußkurven werden im folgenden die Möglichkeiten

- *offline*-Radientraining
- semi-*offline*-Radientraining
- *online*-Radientraining

betrachtet. Für die Breite der Gaußkurven werden die Knoten zusätzlich mit dem Parameter σ ausgestattet:

$$k = (\ldots, \sigma, \ldots) \in \ldots \times [0, \infty)^n \times \ldots .$$

***Offline*-Radientraining** Sei $k \in \mathbb{K}$ und

$$\mathbb{X}_k := \left\{ x \in \mathbb{T} | \bigwedge_{k' \in \mathbb{K}} \left(|x - k.\xi| \leq |x - k'.\xi| \right) \right\}$$

die Teilmenge der Trainingsdaten, für die der Knoten k der *winner*-Knoten ist, d. h. die bei einer Voroni-Tessalierung im Einzugsbereich des Knotens k liegen.

Unter Annahme einer gaußförmigen Verteilung der Trainigsdaten in jeder Koordinatenrichtung um die Knotenposition erhält man für die Breite der Gaußglocke

$$k.\sigma_i^2 =< (x_i - \underbrace{< x >_{\mathbb{X}_k}}_{\approx k.\xi_i})^2 >_{\mathbb{X}_k} \approx < (x_i - k.\xi_i)^2 >_{\mathbb{X}_k} \quad (i \in \{1, \ldots, d\}).$$

Der Nachteil dieses Verfahrens besteht darin, daß zur Radienbestimmung ein zusätzlicher vollständiger Durchlauf durch die Trainingsdaten nach dem Training der Knotenpositionen erforderlich ist.

Da beim Netztraining in regelmäßigen Abständen Auswertungen auf Testdaten erfolgen, um den Fortschritt des Netztrainings beobachten zu können, wird mindestens vor jedem Testschritt ein solcher zusätzlicher Durchgang durch die Trainingsdaten erforderlich.

Semi-*offline*-Radientraining Nach Anfangsinitialisierung

$$k.\sigma_i = 0 \quad (k \in \mathbb{K}, i \in \{1, \ldots, d\})$$

wird jeweils für den *winner*-Knoten

$$k^{\mathrm{W}}.\sigma^2_{i\,(t+1)} = k^{\mathrm{W}}.\sigma^2_{i\,(t)} + (x_i - k^{\mathrm{W}}.\xi_{i(t)})^2$$

berechnet. Nach wählbarer Trainingschrittzahl wird ein Normierungsschritt durchgeführt:

$$k^{\mathrm{W}}.\sigma^2_{i\,(t+1)} = \frac{k^{\mathrm{W}}.\sigma^2_{i\,(t)}}{k^{\mathrm{W}}.\mathrm{counter}_{(t)}} \quad (k \in \mathbb{K}, i \in \{1, \ldots, d\}).$$

Im Unterschied zum *offline*-Ansatz, der die Radienbestimmung für die letzte Position der Knoten durchführt, werden hier auch Trainingsdaten vor der vorläufigen Endposition der Knoten verwendet. Zu Beginn der Trainings können dadurch Unterschiede zwischen diesen beiden Methoden auftreten. Da das Training für die Knotenpositionen in eine stationäre Phase übergehen soll, d. h. sich die Knotenpositionen kaum mehr ändern, wird der Unterschied zwischen den beiden Methoden mit fortschreitendem Training immer geringer.

Durch das semi-*offline*-Verfahren wird ein zusätzlicher Durchgang durch die Trainingsdaten für das Radientraining vermieden. Dies bedeutet Zeitersparnis. Da die Knotenzahl gegenüber der Anzahl der Trainingsdaten verhältnismäßig gering ist, fällt der zusätzliche Durchlauf über die Knoten für den Normierungsschritt wenig ins Gewicht.

Das Ergebnis der Radientrainings steht jedoch nur nach Trainingsschrittintervallen, zu denen ein Normierungschritt durchgeführt worden ist, zur Vefügung; daher die Bezeichnung „semi-*offline*“.

***Online*-Radientraining** Beim *online*-Radientraining wird für den *winner*-Knoten $k^{\mathrm{W}} \in \mathbb{K}$ zum Trainingsdatum $x \in$ T

$$k^{\mathrm{W}}.\sigma^2_i = k^{\mathrm{W}}.\sigma^2_i + \alpha^\sigma \cdot ((x_i - k^{\mathrm{W}}.\xi_i)^2 - k^{\mathrm{W}}.\sigma^2_i)$$

mit dem Lernparameter $\alpha^\sigma \in (0, 1)$ berechnet. Es wurde die Beobachtung gemacht, daß durch einen Trainingsschritt dieser Art, die Variable in den Schwerpunkt (Mittelwert) der Trainingsdaten läuft. Dies wird auch für das Training der Knotenpositionen erwartet, bei dem sich die Knoten in die Schwerpunkte von Cluster bewegen sollen.

Bei dieser Methode steht nach jedem Trainingschritt die Information über die Ausdehnung der Graußglocken der einzeln Knoten zur Verfügung; daher der Name „*online*-Radientraining“.

Da jedoch zu Beginn der Trainings auch Trainingsdaten zum Radientraining verwendet werden, die später nicht mehr diesem Knoten zugeordnet werden oder durch Verschiebung der Knotenposition nicht mehr aktuelle Abstandsdaten verwendet werden, muß bis zur stationären Phase des Knotenpositionstrainings abgewartet werden, um sinnvolle Breitenparameter zu erhalten.

Ein zusätlicher Durchgang über die Trainingsdaten oder die Knoten ist bei diesem Verfahren nicht erforderlich.

In Tabelle 5 ist zusammengefaßt, welche Radientrainingsmethoden für die Knotenpositionstrainingsalgorithmen verwendet werden. Die Zuordnung ist gewachsener Art. Begonnen wurde mit den Algorithmen `gcs` und `gng` mit *online*-Radientraining. Da bei diesen Algorithmen sich die Knotenzahl bei jedem Trainingsschritt ändern kann, ist es sinnvoll die Radien der Knoten aktuell zur Verfügung zu haben. Der Algorithmus `saf` wurde als „Kontrollalgorithmus“ entwickelt. Da hier die Knotenzahl zeitlich fest ist, kann hier das übersichtliche semi-*offline*-Radientraining verwendet werden. Die Entwicklung des hier nur für Demonstrationszwecke verwendeten Algorithmus `kohonen`, war nach der Breitstellung der Verwaltung von Knoten und Nachbarschaftsbeziehungen für die Algorithmen `gcs` und `gng` ein Leichtes. Da die Knotenzahl hier ebenfalls zeitlich konstant ist, kann ebenfalls das semi-*offline*-Radientraining verwendet werden.

`saf`	semi-*offline*-Radientraining
`kohonen`	semi-*offline*-Radientraining
`gcs`	*online*-Radientraining
`gng`	*online*-Radientraining

Tabelle 5: Zusammenstellung der verwendeten Radientrainingsmethoden für die Trainingsalgorithmen der Knotenpositionen

4.3.8 kombiniertes Radien- und Positionstraining

Bei den oben besprochenen Algorithmen haben die Radien keinen Einfluß auf das Training der Knotenpositionen. Eine Möglichkeit besteht darin, jeden Knoten mit einer über seinen Radius bestimmten, lokalen Norm auszustatten. Dadurch kann die bestehende Ausdehnung eines Knotens die Suche nach dem *winner*-Knoten beeinflussen. Ein unkontrolliertes Anwachsen eines einzelnen Knoten ist zu unbinden.

Das kombinierte Training befindet sich in der Entwicklungsphase.

4.4 Netzauswertung

Die bei den oben beschriebenen Trainingsalgorithmen verwendeten Kanten dienen nur der Definition von Nachbarschaftsbeziehungen. Sie sind zu unterscheiden von den in Abschnitt 4 eingeführten Kanten, die den Informationstransport zwischen den Knoten bei der Netzauswertung durchführen. Die Kanten für die Definition der Nachbarschaftsbeziehungen haben keine Bedeutung bei der Netzauswertung.

Das Resultat der verschiednen Trainingsalgorithmen ist eine Menge $\mathbb{K}$ von Knoten der Form

$$(\xi, \sigma, \ldots) \in \mathbb{R}^n \times \mathbb{R}^n \times \ldots$$

Die Netzausgabe wird gemäß

$$\mathcal{N}(x) = \sum_{k \in \mathbb{K}} \exp\left(-\sum_{j=1}^{n} \frac{(x_j - k.\xi_j)^2}{2k.\sigma_j^2}\right) \quad (x \in \mathbb{R}^n)$$

berechnet. Um eine binäre Netzentscheidung zu erhalten, wird ein Schwellenkriterium mit Schwelle $c \in \mathbb{R}$

$$\tilde{\mathcal{N}}_c(x) = \Theta(\mathcal{N}(x) - c) \quad (x \in \mathbb{R}^n)$$

angewendet.

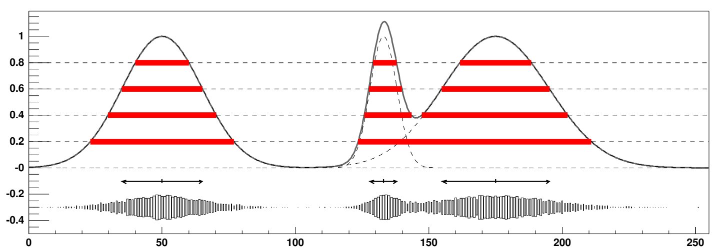

Abbildung 24: Netzausgabe und Schwellenkriterium: Das *box*-Histogramm unten gibt die Verteilung der Trainingsdaten an. Die Position und Breite der drei Knoten sind durch die Pfeile gekennzeichnet. Die binäre Netzentscheidung $\tilde{\mathcal{N}}_c$ ist für verschiedene Schwellen durch die breiten Balken verdeutlicht.

4.4.1 Netzeffizienz

Zur Bestimmung der Güte eines Netzes werden Testdaten verschiedener Klassen verwendet.

Seien $\mathbb{P}^{\geq}, \mathbb{P}^{<} \subset \mathbb{R}^n$ Klassen von Testdaten mit erwarteter Netzausgabe $\mathcal{N} \geq$ bzw. $<$ c.

Durch

$$\begin{aligned}\mathrm{Eff}_c^{\geq} &:= \frac{|\{x \in \mathbb{P}^{\geq} | \mathcal{N}(x) \geq c\}|}{|\mathbb{P}^{\geq}|} \\ \mathrm{Eff}_c^{<} &:= \frac{|\{x \in \mathbb{P}^{<} | \mathcal{N}(x) < c\}|}{|\mathbb{P}^{<}|}\end{aligned}$$

wird die Effizienz der entsprechenden Klasse definiert. In dem hier vorliegenden Anwendungsbereich ist

$\mathbb{P}^{\geq}$ eine Teilmenge der *big events* und
$\mathbb{P}^{<}$ eine Teilmenge der Physikereignisse.

Falls zum Test des Netzes beide Klassen $\mathbb{P}^{\geq}, \mathbb{P}^{<}$ zur Verfügung stehen, kann ein zweidimensionaler Effizienplot

$$\mathbb{G} := \{(x, y) \in \mathbb{R}^2 | \bigvee_{c \in \mathbb{R}} (x = \mathrm{Eff}_c^{\geq} \wedge y = \mathrm{Eff}_c^{<})\} \tag{5}$$

erstellt werden.

Eine hohe Untergrunderkennung und zugleich hohe Physikerkennung ziehen den Effizienzplot weit in die obere rechte Ecke.

Für eine Entscheidung nach dem *prescale*-Prinzip (siehe z. B. Abschnitt 3.1.2) und zufällig verteilten Daten erhält man für den Effizienzplot $\mathbb{G}^{prescale}$ eine Gerade:

$$\mathbb{G}^{prescale} \approx \{(x, y) \in [0, 1]^2 | x + y = 1\}.$$

Beim Netztraining werden in wählbaren Trainingsschrittintervallen Testdaten durch das Netz ausgewertet. Dadurch ist es möglich, den Fortschritt des Trainings zu beobachten und dieses entsprechend abzubrechen oder fortzusetzen. Um eine solche Testauswertung durch eine Zahl ϵ aus dem Intervall $[0, 1]$ zu bewerten, wird

$$\epsilon(\mathbb{G}) = \max\{z \in [0, 1] | \bigvee_{(x,y) \in \mathbb{G}} (z = (x + y - 1))\}$$

definiert. Geometrisch kann diese Größe als der maximale Abstand von $\mathbb{G}$ von der Geraden $x + y = 1$ skaliert auf das Intervall $[0, 1]$ interpretiert werden.

Die Größe von $\epsilon(\mathbb{G})$ stellt insofern ein Maß für die Netzgüte dar, daß bei „kleinem“ $\epsilon(\mathbb{G})$ (siehe z. B. 8) eine hohe Netzeffizienz nicht möglich ist. Für „großes“ „kleinem“ muß zusätzlich $\mathbb{G}$ direkt überprüft werden.

5 Selektion der Trainings- und Testdaten

5.1 Anforderungen an die Trainingsdaten

Folgende Anforderungen werden an die Trainigsdaten gestellt:

1. big events
2. mit Sicherheit Untergrundereignisse
3. Daten, wie sie der L2-Trigger sieht

5.2 POT2-Daten als Ausgangspunkt für die Selektion der Trainingsdaten

Für das Training eines Netzes für den L2-Trigger werden Trainingsdaten in der Form benötigt, wie sie später die Triggerbox als Eingabe erhält. Insbesondere sind auch solche Ereignisse zu berücksichtigen, die auf höheren Triggerstufen verworfen werden. Daher bilden Daten aus L2-L4-*transparent runs* (siehe Abschnitt 3.6) die Grundlage der Trainingsdatenselektion.

H1-Realdaten liegen in den Formen *raw data*, POT und DST vor (siehe Abschnitt 3.5). Da auf DST Bänke mit L4-Informationen, wie z. B. die Positionen rekonstruierter Vertices, nicht zur Verfügung stehen, scheidet DST aus. Da jedoch zur Selektion der Trainingsdaten rekonstruierte Größen verwendet werden müssen[10], kann *raw data* ebenfalls nicht verwendet werden.

Als Ausgangspunkt für die Selektion der Trainingsdaten werden daher POT2-Daten aus dem Jahr 1997 von L2-L4-*transparent runs* verwendet. Diese sind in Tabelle 6 zusammengestellt.

Die Bestimmung eines Selektionskriteriums und damit eine Definition der Eigenschaft *big event* ist Inhalt des folgenden Abschnittes.

5.3 Bestimmung des Schnittes auf die rekonstruierten Daten

In diesem Abschnitt wird schrittweise zu dem Selektionskriterium für *big events* hingeführt.

[10]Andernfalls könnte man einen L2-Trigger konstruieren und müßte kein Netz trainieren.

run-Nummer	Datum	Phase	Ereigniszahl	Zahl der BE
180061	16.03.1997	3	14997	5998 / 1669
181003	22.03.1997	4	12880	3578 / 801
181401	25.03.1997	3	20047	5844 / 1510
184602	18.04.1997	2	11419	3034 / 381
192496	19.06.1997	3	4145	2512 / 1093
197809	20.08.1997	4	2749	2196 / 325
Summe			66237	23162 / 5779

Tabelle 6: Die Tabelle enthält die Liste der zur Trainingsdatenselektion verwendeten *runs*. Alle *runs* besitzen die Qualität *good*, d.h. alle wichtigen Detektorsysteme (CJC1, CJC2, LAr-Kalorimeter, Myondetektoren, BDC, Lumi-System, SpaCal) sind in Betrieb. Die Forderung nach Qualität *good* und Phase 2, 3, oder 4 schränken die Zahl der L2/L4-*transparent runs* aus der Datennahmeperiode 1997 stark ein. Die Ereigniszahl gibt die Anzahl der betrachteten Ereignisse an. Kleine Bruchteile am Anfang oder Ende von *runs* mit relativ wenigen Ereignissen auf angrenzenden POT-Bändern wurden teilweise nicht verwendet. In der Spalte „Zahl der BE“ (*big events*) gibt die erste Zahl die Gesamtzahl der *bigevents* und die zweite die Zahl der zum Training verwendete Ereignisse an. (siehe Abschnitt 5.3)

Da in Phase 1 die Hochspannung der Spurkammern erst hochgefahren wird und sich die L1-Subtriggerbedinungen und *prescale*-Faktoren von Phase 1 stark von denen in Phase 2, 3, 4 unterscheiden, werden für die Selektion der Trainingsdaten nur Daten aus den Phasen 2, 3, und 4 verwendet. Die Subtriggerbedingungen und *prescale*-Faktoren der Phasen 2, 3, 4 sind dagegen untereinander jeweils vergleichsweise ähnlich. [47]

$$\mathrm{cut}_{\mathrm{P}} \quad = \quad (\mathrm{phase} \in \{2,3,4\}) \tag{6}$$

Big events sind durch ihre lange, zu Problemen führende Rechenzeit auf L4 gekennzeichnet. Deshalb wäre ein schönes Kriterium für die Erkennung *big events* die Zeitdauer der Entscheidungsfindung von L4. Von dieser konnte jedoch aus folgenden Gründen kein Gebrauch gemacht werden:

Die Zeit der *online*-Enscheidung wird nicht gespeichert [29], so daß nur noch die Entscheidungszeit der *offline*-L4-Simulation (Programm `filter`) zur Verfügung steht. Das Programm `filter` ist ausgelegt für eine *offline*-L4-Simulation und zum Test sogenannter *finder*, welche auf Triggerstufe 4 besonders interessante Ereignisklassen „retten“ sollen.

Die L4-Entscheidung konnte ebenfalls nicht zur Datenselektion herangezogen werden, da bei L2-L4-*transparent runs* die *online*-L4-Entscheidung (BOS-Bank: DMIS) immer „ja“ (d. h. das Ereignis wird akzeptiert) ist. [29]

Entscheidend für die Nichtverwendbarkeit der *offline*-Entscheidungszeit ist, daß das Programm `filter` als Eingabe *raw data tapes* benötigt; wegen der in Abschnitt 5.2 aufgeführten Gründe, werden jedoch ausschließlich POT-Daten für die Trainingsdatenselektion verwendet.

Im Vergleich zu *raw data tapes* existieren bzw. fehlen auf POT einige BOS-Bänke, so daß die *offline*-L4-Simulation diese nicht Erzeugen muß bzw. die Eingabedaten fehlen. Dadurch weicht die *offline*-L4-Entscheidungszeit in unabsehbarer Weise von derjenigen bei *raw data tapes* als Eingabe bzw. der *online*-L4-Entscheidungzeit ab. [29]

Um eine L4-Entscheidung zu erhalten, müßte wieder *offline* das Programm `filter` angewendet werden, was jedoch die gleichen Probleme wie oben ergibt.

Da die L4-Rechenzeit nicht zur Verfügung steht, wird als Kriterium die Anzahl der Signale in der zentralen Spurkammer CJC verwendet, welche die Rechenzeit der L4-Rekonstruktion starkt beeinflußt. Dabei gilt nach [29] [11]

$$\text{big event} \Longrightarrow \#(\text{CJC-hits}) > 500. \tag{7}$$

Die Rekonstruktionen auf L4 bzw. L5 ermitteln Positionen von Vertexkandidaten. Diese Informationen werden in die BOS-Bänke CJV4 bzw. DVER geschrieben. Dazu wird folgende Bezeichnung eingeführt: Ein Ereignis hat eine *validierte* z-Koordinate, wenn die Bänke CJV4 und DVER existieren, eine von Null verschiedene Länge haben und die DVER-Bank mindestens einen Eintrag von Typ *primary vertex* besitzt, andernfalls hat dieses Ereignis keine *validierte* z-Koordinate.

Da später ein zusätzliches Kriterium auf die z-Koordinate des rekonstruierten Vertex angewendet wird, werden zunächst Ereignisse betrachtet, die keine *validierte* z-Koordinate besitzen: Die Verteilung der Anzahl der Signale der CJC für Ereignisse mit **nicht**-*validierter* z-Koordinate in Abbildung 25 ergibt, daß diese Ereignisse **keine** *big events* sind.

Der Schnitt kann also jeweils ohne Verlust von *big events* erweitert werden zu

$$\begin{aligned} \text{cut}_{\text{PV}} \quad = \quad & (\text{phase} \in \{2,3,4\}) \wedge \\ & (\text{validierte } z\text{-Koordinate}) \end{aligned} \tag{8}$$

und wegen (7) zu

[11] Die Richtung des Implikationspfeiles ist beabsichtigt.

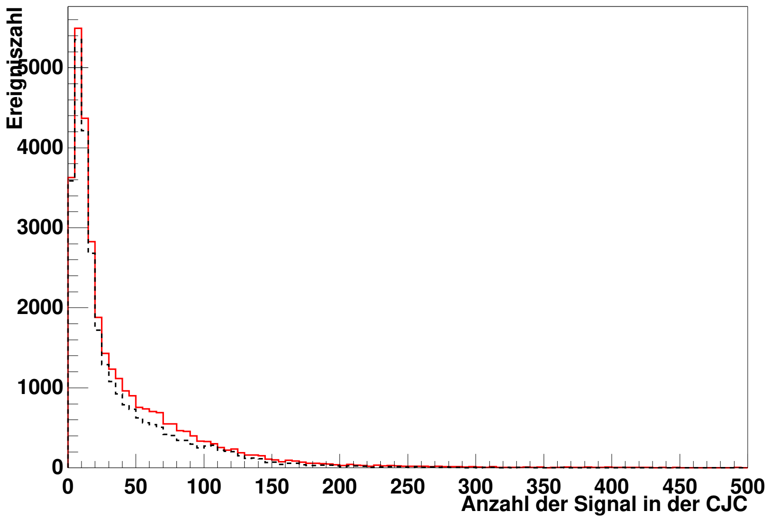

Abbildung 25: Verteilung der Anzahl der Signale der CJC für Ereignisse aus Phase 2, 3 oder 4 der betrachteten L2-L4-*transparent runs* mit nicht-validierter z-Koordinate. Das durchgezogene Histogramm enthält die Ereignisse, zu denen keine L4-z-Vertex-Information zur Verfügung steht, und das gestrichelte diejenigen Ereignisse, zu denen keine L5-z-Vertex-Information vorhanden ist. mit nicht validierter L5-z-Koordinate. Diese Ereignisse sind also wegen der geringen Zahl der Signale in der CJC **keine** *big events*.

$$
\begin{aligned}
\mathrm{cut}_{\mathrm{PVH}} \quad = \quad & (\text{phase} \in \{2,3,4\}) \wedge \\
& (\text{validierte } z\text{-Koordinate}) \wedge \\
& (\#(\text{CJC-hits}) > 500)
\end{aligned} \tag{9}
$$

Mit dem Schnitt (9) ergibt sich das punktierte Histogramm der z-Koordinate des auf Triggerstufe 4 rekonstruierten Vertex in Abbildung 26

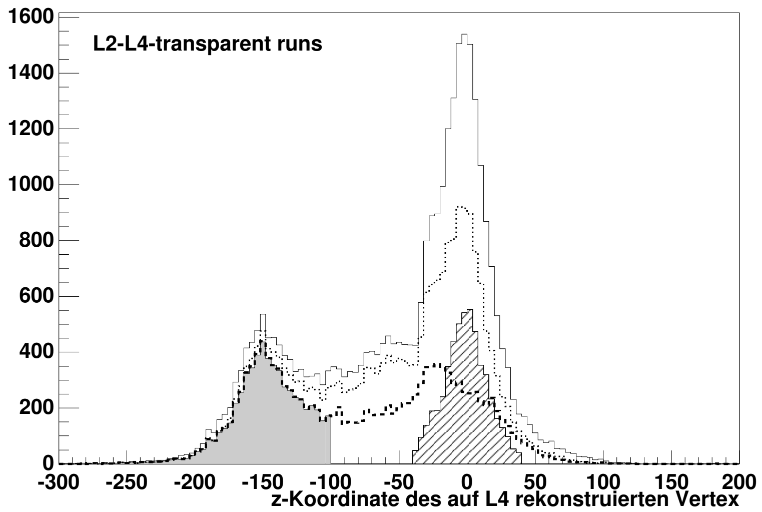

Abbildung 26: Verteilung der z-Koordinate des auf Triggerstufe 4 rekonstruierten Vetex für die Ereignisse der betrachteten L2-L4-*transparent runs*: Das durchgezogene Histogramm enthält die Ereignisse mit validierter z-Koordinate (32115 Ereignisse). Für das gepunktete Histogramm wurde zusätzlich der Schnitt #(CJC-hits) $>$ 500 angewendet (23162 Ereignisse) (Schnitt (9)). Mit dem zusätzlichen Schnitt $F < 0.22$ wird das gestrichelte Histogramm gewonnen (13880 Ereignisse). Weitere Anwendung von z-Koordinate Vertex $<$ -100 cm und des L1ST-Cocktail ergeben das grau gefüllte Histrogramm (5779 Ereignisse). Diese Ereignisse werden für das Training verwendet. Das schraffierte Histogramm enthält Vertexereignisse ($|z| < 40$ cm) mit #(CJC-hits) $<=$ 500, $F >= 0.22$ und mindestens einem *actual* L1ST aus dem L1ST-Cocktail des *background encapsulator* (3035 Ereignisse).

Da der einfache zusätzliche Schnitt (#(CJC-hits) $>$ 500) Physikereignisse bzw. Ereignisse, die der *background encapsulator* nicht verwerfen soll, nicht ausschließt (siehe Abbildung 27), muß differenzierter vorgegangen werden.

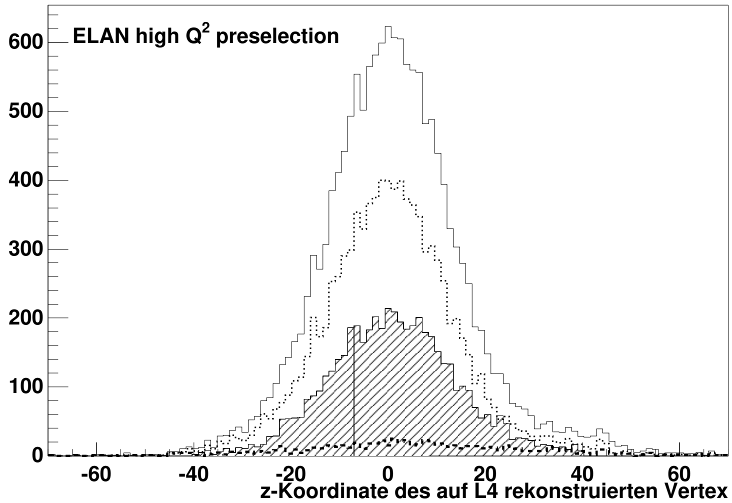

Abbildung 27: Verteilung der auf L4 rekonstuierten z-Vertexposition für Ereignisse aus Teilen der *ELAN high* Q^2 *preselection*: Das durchgezogene Histogramm enthält die Ereignisse mit validierter z-Koordinate (15220 Ereignisse). Für das gepunktete Histogramm wurde zusätzlich der Schnitt #(CJC-hits) $>$ 500 angewendet (9562 Ereignisse) (Schnitt (9)). Mit dem zusätzlichen Schnitt $F < 0.22$ wird das gestrichelte Histogramm gewonnen (822 Ereignisse). Das schraffierte Histogramm enthält Vertexereignisse ($|z| < 40$ cm) mit #(CJC-hits) $<=$ 500, $F >= 0.22$ und mindestens einem *actual* L1ST aus dem L1ST-Cocktail des *background encapsulator* (5066 Ereignisse).

Dazu wird der Bruchteil F der CJC-Signale für ein Ereignis betrachtet, welche für die Rekonstruktion von Spuren zum Primärvertex auf L5 verwendet werden:

$$F := \frac{\#(\text{verwendeten CJC-Signale für Primärvertexspurrekonstruktion})}{\#(\text{CJC-hits})}.$$

Wie aus Abbildung 28 zu ersehen ist, gibt es unter den durch den Schnitt (9) erfaßten Ereignissen der betrachteten L2-L4-*transparent runs* eine Anhäufung solcher, bei denen nur ein geringer Bruchteil der vorhandenen CJC-Signale für die Rekonstruktion der Spuren zum Primärvertex verwendet worden ist. Diese deutliche Anhäufung ist bei den nicht durch Schnitt (9) erfaßten Ereignissen der L2/L4-*transparent runs* und den Ereignissen aus Teilen der *ELAN high* Q^2 *preselection* nicht zu beobachten. Dies legte eine Erweiterung des Schnittes zu

$$\begin{aligned} \text{cut}_{\text{PVHF}} \quad = \quad & (\text{phase} \in \{2,3,4\}) \wedge \\ & (\text{validierte } z\text{-Koordinate}) \wedge \\ & (\#(\text{CJC-hits}) > 500) \wedge \\ & (F < 0.22) \end{aligned} \tag{10}$$

nahe.

In Abbildung 29 ist der Bruchteil der Ereignisse von den durch Schnitt (9) erfaßten Ereignisse, die durch den Schnitt (11) erfaßt werden, in Abhänigkeit der z-Koordinate des Vertex (L4) dargestellt

Anhand von Ereignissen aus Teilen der *ELAN high* Q^2 *preselection 1997* wird aus der Verteilung der z-Koordinate des Primärvertex (siehe Abbildung 27) eine Grenze bei normaler Strahllage abgeschätzt, unterhalb derer Physikereignisse aus Elektron-Proton-Kollisionen extrem selten auftreten:

$$z < -100 \text{ cm}.$$

Der zusätzliche, den *peak* in Proton-*upstream*-Richtung in Abbildung 26 mitnehmende Schnitt $z < -100$ bewirkt, daß der zu

$$\begin{aligned} {\text{cut}_{\text{Untergrund}}}^{raw} \quad = \quad & (\text{phase} \in \{2,3,4\}) \wedge \\ & (\text{validierte } z\text{-Koordinate}) \wedge \\ & (\#(\text{CJC-hits}) > 500) \wedge \\ & (F < 0.22) \wedge \\ & (z\text{-Koordinate Vertex} < -100 \text{ cm}) \end{aligned} \tag{11}$$

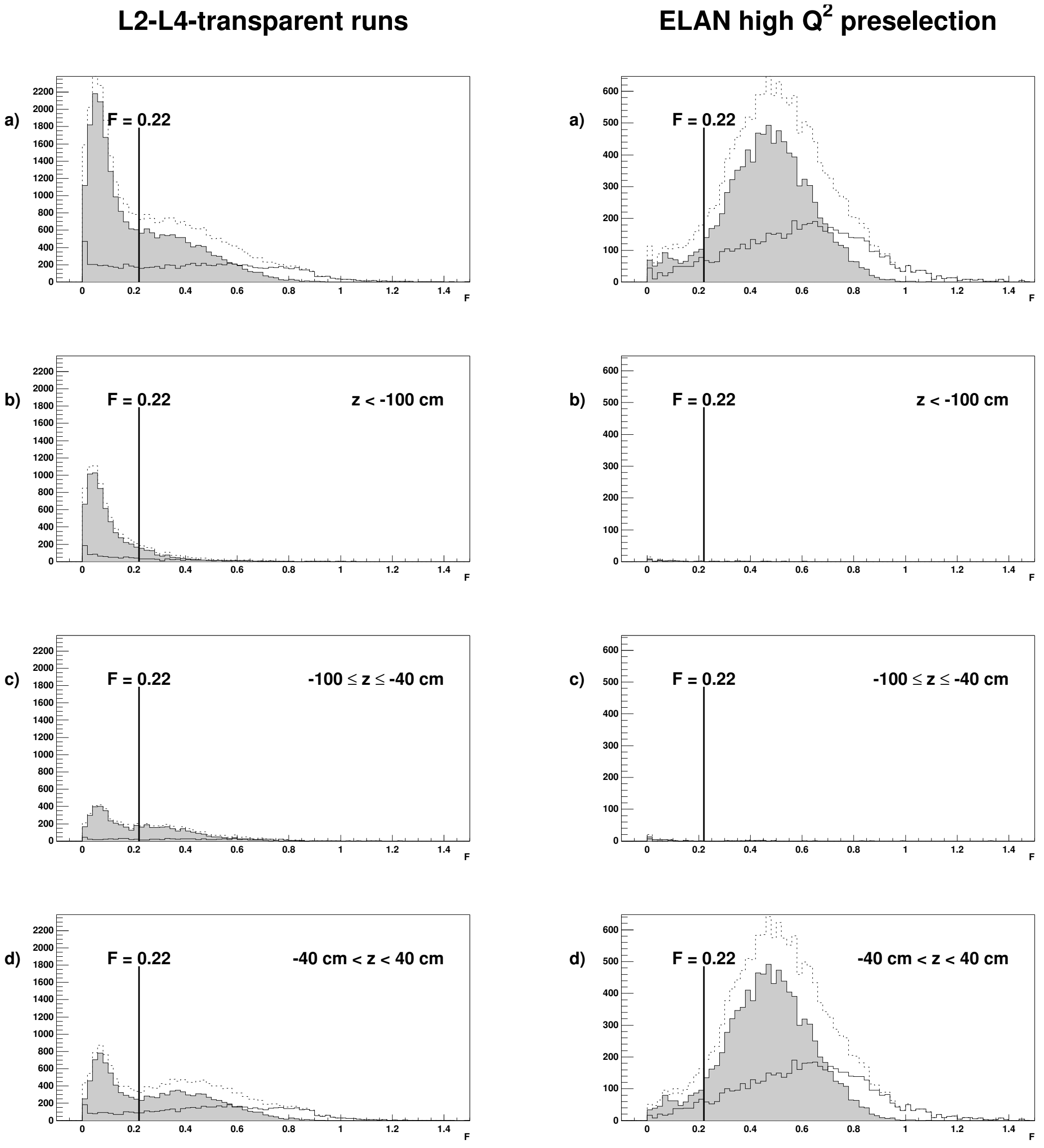

Abbildung 28: Verteilung der Größe F für Ereignisse aus Phase 2, 3, 4 mit *validierter* z-Koordinate aus den betrachteten L2-L4-*transparent runs* bzw. aus Teilen der *ELAN high* Q^2 *preselection*. In der Darstellung a) wird das gepunktete Histogramm durch diese Ereignisse gebildet. Für das grau gefüllte bzw. das leere Histogramm wurde zusätzlich die Bedingung $\#(CJC - hits) > 500$ (*big event*-Kandidaten) bzw. $\#(CJC - hits) <= 500$ (keine *big events*) angewendet. Für die weiteren Darstellungen b), c) und d) wurden die dort angegebenen Einschränkungen an die z-Koordinate (L4) jeweils zusätzlich angewendet.

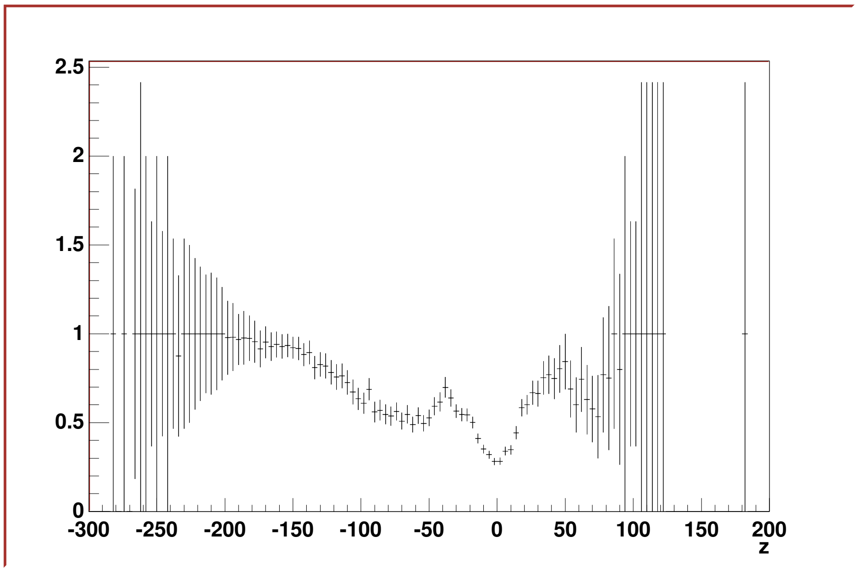

Abbildung 29: Anteil (mit Fehler) der durch den Schnitt zusätzlichen Schnitt $F < 0.22$ erfaßten Ereignisse des Schnittes (9) in Abhängigkeit von z: Der Großteil der durch den Schnitt (9) erfaßten *proton upstream* Ereignisse werden auch durch den Schnitt (11) erfaßt. Die großen Fehlerbalken für großes $|z|$ sind in der dort geringen Ereigniszahl begründet.

erweiterte Schnitt eine höchst reine, an *big events* stark angereicherte Zusammenstellung von Untergrundereignissen erfaßt.

Um den *background encapsulator* nicht von den anderen L2TE/ST abhängig zu machen, wird die auf POT-Daten zur Verfügung stehende L2-Entscheidung nicht zur Selektion der Trainingsdaten verwendet.

5.4 Testdaten

Für die Bestimmung des L1ST-Cocktails, für die Auswahl von zum Training geeingeten L2-Größen und zum Test trainierter Netzt wird eine weitere Klasse von Ereinissen benötigt, die sich von den *big events* unterscheiden und ähnlich zu Physikereignissen sind.

$$
\begin{aligned}
\mathrm{cut}_{\mathrm{Vertexereignis}}{}^{raw} \quad = \quad & (\mathrm{phase} \in \{2,3,4\}) \wedge \\
& (\text{validierte } z\text{-Koordinate}) \wedge \\
& \neg(\#(\text{CJC-hits}) > 500) \wedge \\
& \neg(F < 0.22) \wedge
\end{aligned}
\qquad (12)
$$

($|z$-Koordinate Vertex$| < -40$ cm)

Durch den Schnitt (12) werden Ereinisse aus der Region um den nominellen Vertex erfaßt, die keine *big events* sind und bei denen ein nicht geringer Anteil der CJC-Signale für die Rekonstruktion von Spuren zum Primärvertex verwendet werden.

In den Abbildungen 30 und 31 sind einige Ereignisse für verschiedene Klassen dargestellt. Durch die obere Darstellung in Abbildung 31 wird der zusätzliche Schnitt $z < -100cm$ in (11) in unterstüzt.

5.5 L1ST-Cocktail

Durch den *background encapsulator* soll ein weiteres L2TE des L2NN bereitgestellt werden. Es soll als Veto auf *big events* dienen. Dies kann im Schritt der Bildung von *unvalidated* L2ST aus logischen Kombinationen der L2TE erfolgen (siehe Abschnitt 3.2.2). Die gebildeten L2ST müssen dann durch eine L1ST aus dem ihnen zugeordneten L1ST-Cocktail validierte werden.

In diesem Abschnitt werden Untersuchungen zu einem L1ST-Cocktail für das Triggerelement *background encapsulator* ohne Kombination mit anderen L2TE unternommen.

Dazu werden zunächst von den derzeit 128 *actual* L1ST nur diejenigen betrachtet, von denen erwartet wird, daß sie Physikereignisse triggern. L1ST sind durch je ein Bit als *physics trigger* bzw. *monitoring trigger* klassifiziert. Um gegnüber eventuellen Änderungen dieser Klassifizierung während einer Datennahmeperiode möglichst unabhängig zu sein, werden im weiteren nur solche L1ST betrachtet, die als *physics trigger* klassifiziert sind und innerhalb der betrachteten L2-L4-*transparent runs* nicht als *monitoring trigger* klassifiziert sind. Weiter werden nur solche L1ST betrachtet, die in Phase 4 (siehe Abschnitt 3.1.2) den *prescale*-Faktor 1 besitzen, d. h. deren Rate nicht durch *prescaling* (siehe Abschnitt 3.1.2) reduziert wird.

In dieser Auswahl von L1ST sollten diejenigen enthalten sein, die in vernünftiger Rate Physikereignisse triggern. besitzen

Für die durch den Schnitt (11) erfaßten Ereignisse ist in Abbildung 32 die Verteilung der L1ST dargestellt, welche zu *L2Keep* beitragen, d. h. die L1ST je Ereignis sind ausgeschaltet, die in keinem L1ST-Cocktail eines L2ST enthalten sind, der zu *L2Keep* geführt hat (vgl. Abschnitt 3.2.3).

Die „boolschen“ Histogramme in Abbildung 32 bezeichnen

- L1ST hat in Phase 4 *prescale* 1

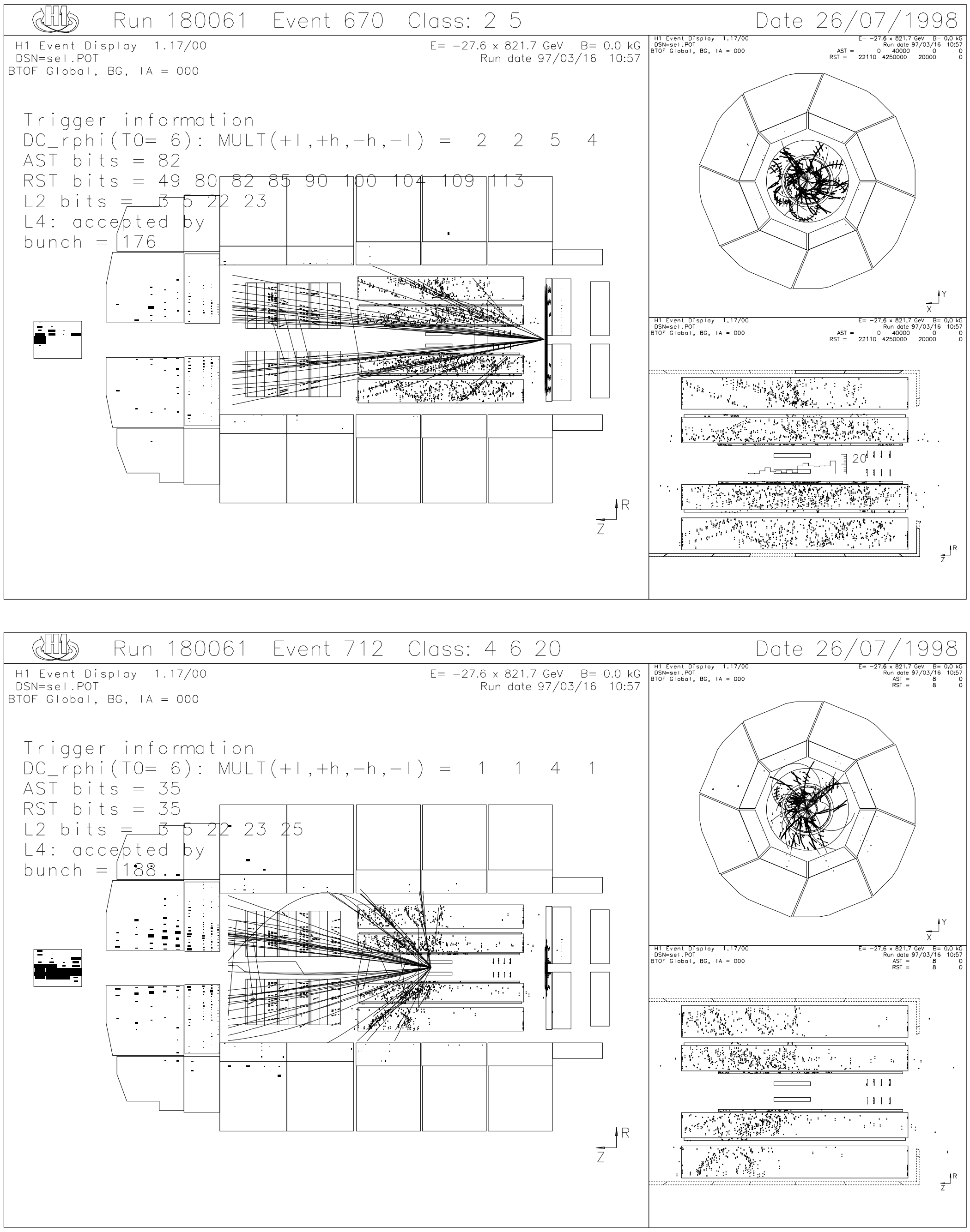

Abbildung 30: Die Darstellungen zeigen eine seitliche und radiale Ansicht des H1-Detektors. Dort sind die Signale in den Spurkammern, Energiedepositionen in den Kalorimetern, die auf L4 rekonstruierten Spurstücke und die auf L5 rekonstruierten, vertexgefitteten Spuren eingezeichnet. In der rechten unteren Teildarstellung sind die zentralen Spurkammern vergrößert dargestellt. Um den nominellen Wechselwirkungspunkt ist das z-Vertex-Histogramm eingezeichnet. Das Ereignis *run* 180061 *event* 670 fällt in die Klasse der zum Training verwendeten Untergrundereignisse. Nur durch den Schnitt $z < -100$ cm wird das Ereignis *run* 180061 *event* 712 aus der Klasse der zum Training verwendeten Untergrundereignisse ausgeschlossen.

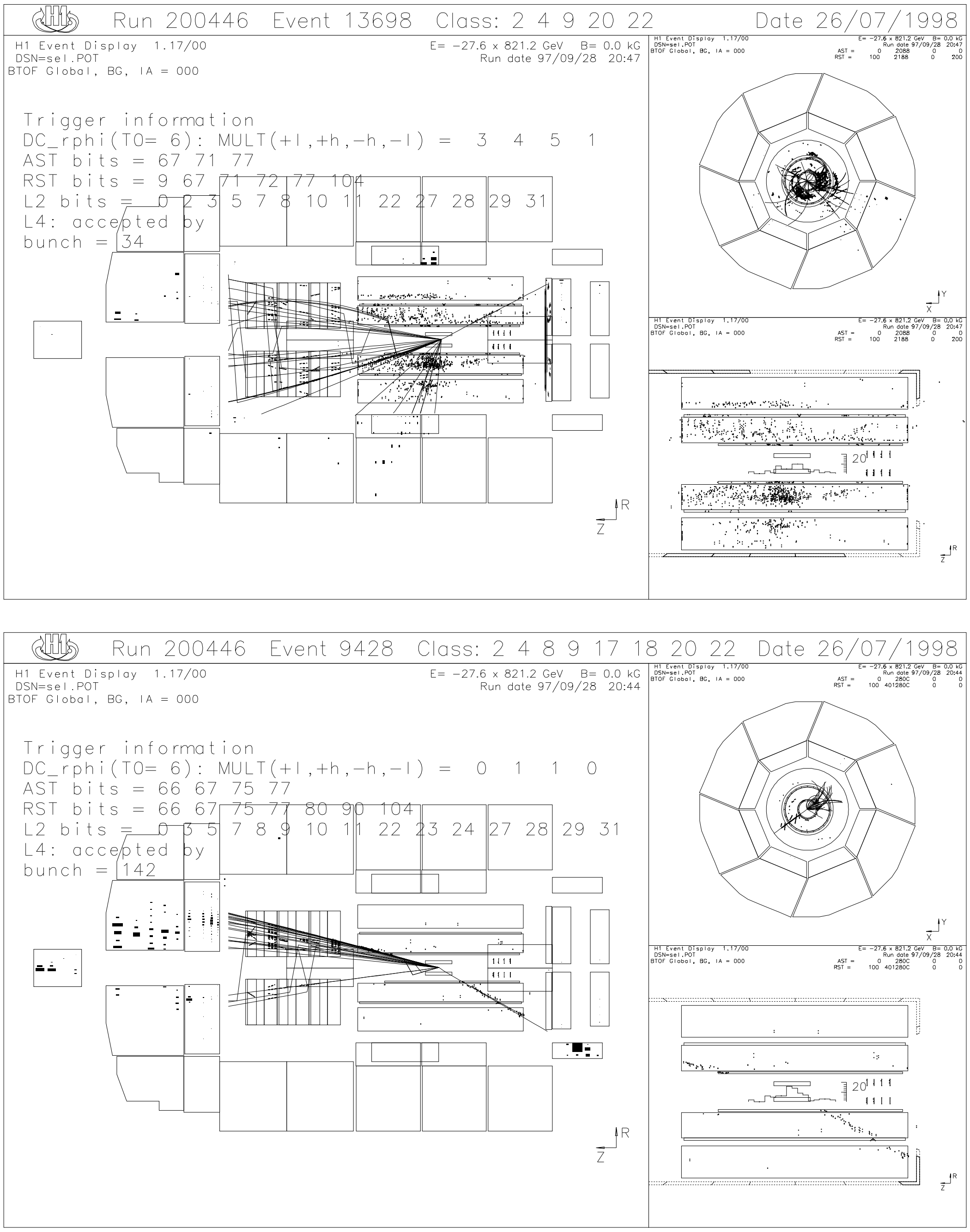

Abbildung 31: Beide Ereignisse sind aus den betrachteten Teilen der *ELAN high* Q^2 *preselection*. Bis auf den Schnitt $z < -100$ cm erfüllt das Ereignis *run* 200446 *event* 13698 die restlichen Bedingungen für ein zum Training verwendetes Untergrundereignis. Das Ereignis *run* 200446 *event* 9428 fällt in die Klasse der zum Testen verwendete Vertexereignisse. (siehe auch die Beschreibung zu Abbildung 30)

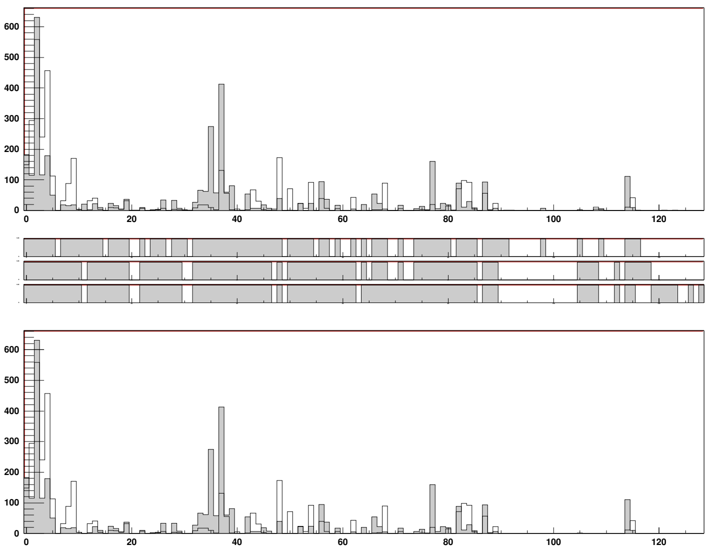

Abbildung 32: „*physics*“, actual L1ST (Beschreibung im Text)

- L1ST ist *physics trigger*
- L1ST ist nicht *monitoring trigger*.

Durch Multiplikation dieser drei Histogramme mit dem oberen L1ST-Histogramm in Abbildung 32 — diese Operation entspricht einem logischen UND — erhält man das untere Histogramm in Abbildung 32.

Die L1ST dieses Histogramms werden im folgenden weiter betrachtet.

Für die Auswahl des L1ST-Cocktails für den *background encapsulator* gibt es folgende Möglichkeiten:

1. so viele L1ST, beginnend mit denen mit der größten Rate, verwenden bis ca. 90% der durch den Schnitt (11) erfaßten Ereignisse selektiert werden. Die Rate von 90% wurde so hoch gewählt, da die Effizienz des *background encapsulators* später zusätzlich zu beachten ist.

2. nur eine kleine Anzahl (ca. 5 bis 10) von L1ST verwenden, zu denen die „getriggerte“ Physik bekannt ist. Dadurch müssen Untersuchungen der *background encapsulator*-Entscheidung nur für diese Physikklassen durchgeführt werden.

3. alle *physics trigger*

Für eine gewünschte *big event*-Rate von 90% ergibt die Abschätzung in Abbildung 33, deren Annahme im Allgemeinen nicht erfüllt ist, eine untere Grenze von über 30 für die Anzahl der benötigten L1ST. Damit besteht zwischen Möglichtkeit 1. und 3. kein größerer Unterschied mehr.

Um den *background encapsulator* möglichst allgemein zu halten, d. h. unabhänigig von *prescale* und Bildung der einzelnen L1ST, wird Möglichkeit 2. vorerst nicht gewählt.

Daher wird folgender L1ST-Cocktail gewählt

$$\text{L1ST-Cocktail} := \text{alle } \textit{physics trigger}.$$

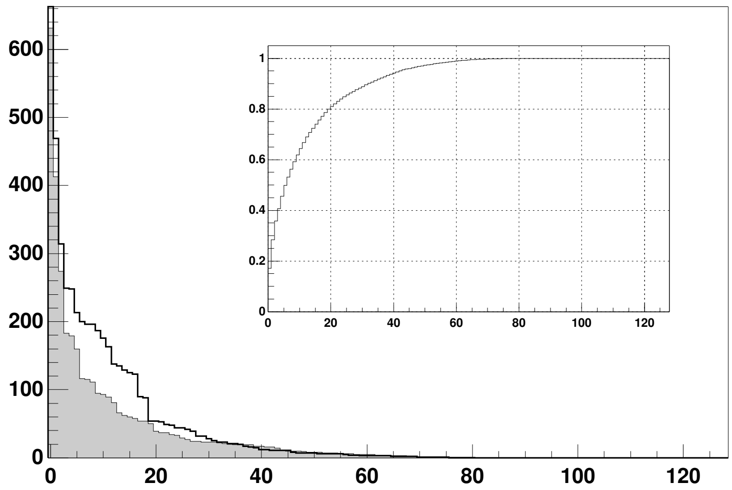

Abbildung 33: L1ST der Größe nach geordnet (graues Histogramm: *big events*; leeres Histogramm: Vertexereignisse): In der rechten oberen Teildarstellung ist das auf ein Maximum von 1 normierte Integral über das graue Histogramm dargestellt. Dieses ermöglicht die Anzahl der benötigten L1ST für den gewünschten Anteil von *big events* nach unten unter der Annahme, daß nur ein L1ST jeweils triggert, abzuschätzen.

Mit diesem L1ST-Cocktail stehen nun die Schnitte für die Ereignisklassen *big event* und Vergleichsdaten-Vertexereignis zur Verfügung:

$$\begin{aligned}\text{cut}_{\text{Untergrund}} \quad = \quad & (\text{phase} \in \{2,3,4\}) \wedge \\ & (\text{validierte } z\text{-Koordinate}) \wedge\end{aligned}$$

$$
\begin{aligned}
&(\#(\text{CJC-hits}) > 500) \wedge \\
&(F < 0.22) \wedge \\
&(z\text{-Koordinate Vertex} < -100 \text{ cm}) \wedge \\
&(\text{L1ST-Cocktail})
\end{aligned}
\qquad (13)
$$

$$
\begin{aligned}
\text{cut}_{\text{Vertexereignis}} = \; &(\text{phase} \in \{2,3,4\}) \wedge \\
&(\text{validierte } z\text{-Koordinate}) \wedge \\
&\neg(\#(\text{CJC-hits}) > 500) \wedge \\
&\neg(F < 0.22) \wedge \\
&(|z\text{-Koordinate Vertex}| < -40 \text{ cm}) \wedge \\
&(\text{L1ST-Cocktail})
\end{aligned}
\qquad (14)
$$

Ereignisse der Klassen $\text{cut}_{\text{Untergrund}}$ bzw. $\text{cut}_{\text{Vertexereignis}}$ bilden jeweils die das grau gefüllte bzw. schraffierte Histogramm in den Abbildungen 26 und 27.

5.6 Auswahl für das Training geeigneter L2 Größen

Diese Ereignisse werden später auch als Teil der Testdaten für die trainierten Netze verwendet.

Um aus der großen Anzahl (siehe [13]) von L2-Größen, diejenigen herauszufinden, welche sich für das Training eignen, werden folgende Möglichkeiten verwendet, den Unterschied der Verteilungen einer L2-Größe für die, durch die Schnitte $\text{cut}_{\text{Untergrund}}$ und $\text{cut}_{\text{Vertexereignis}}$ erfaßten Ereignisse, durch je eine Zahl zu bewerten:

1. Korrelation:

$$\varrho_{xy} = \sum_{j=1}^{N} \frac{(x_j - <x>)(y_j - <y>)}{N\sigma_x\sigma_y} = \frac{<xy> - <x><y>}{\sigma_x\sigma_y}$$

   Für $x = \pm y$ ist $\varrho_{xy} = \pm 1$.

2. Differenz der Mittelwerte der Verteilungen

3. Differenz der Streuung der Verteilungen

4. Integral des Quadrates der Differenz der beiden jeweils normierten Verteilungen

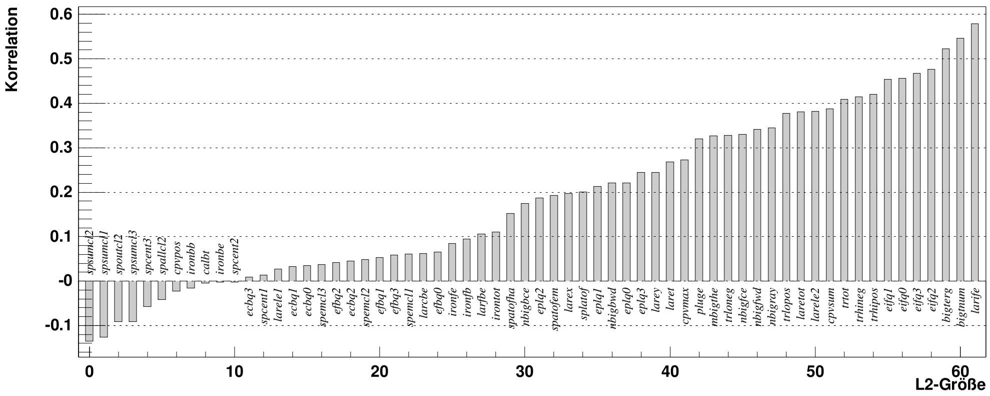

Abbildung 34: Korrelation zwischen L2-Größe und Ereignisklasse *big event* bzw. Vertexereignis

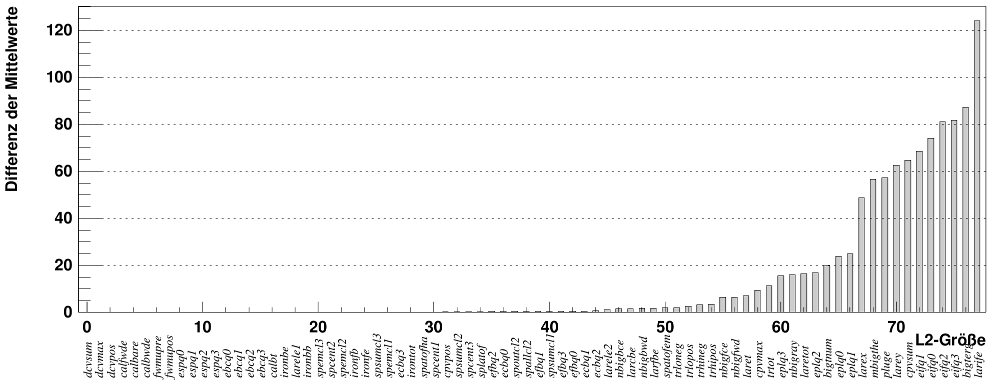

Abbildung 35: Differenz der Mittelwerte der Verteilungen der L2-Größe für die Ereignisklassen *big event* bzw. Vertexereignis

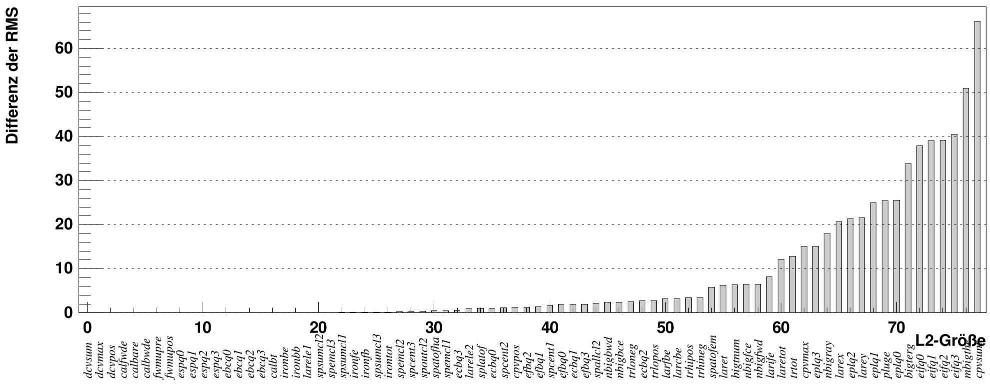

Abbildung 36: Differenz der Streuung der Verteilungen der L2-Größe für die Ereignisklassen *big event* bzw. Vertexereignis

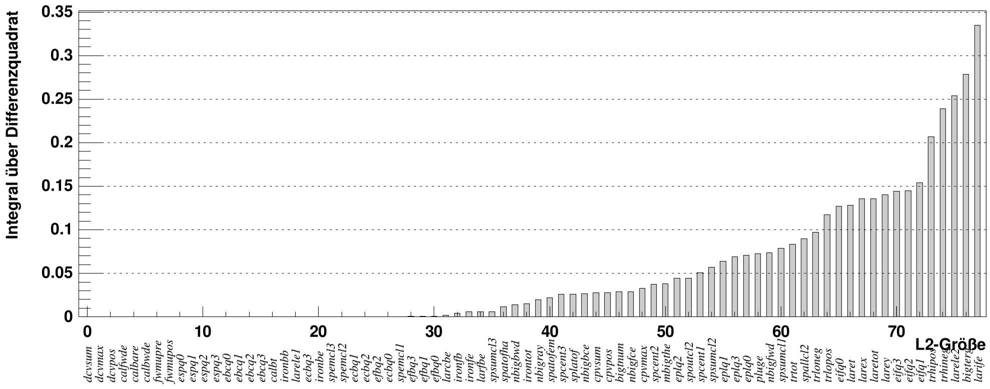

Abbildung 37: Integral über das Quadrat der Differenz der Verteilungen der L2-Größe für die Ereignisklassen *big event* bzw. Vertexereignis

Das Ergebnis für die verschiedenen Kriterien — jeweils der Größe nach sortiert — ist in den Abbildungen 34, 35, 36, 37 dargestellt.

Eine Zusammenstellung der sich als Netzeingabegrößen anbietenden L2-Größen befindet sich in Tabelle 7.

Diese legt nahe, die in Tabelle 7 aufgeführten L2-Größen für die Aufbereitung von Trainingsdaten zu verwenden. Eine Beschreibung der übrigen, hier weiter nicht verwendeten L2-Größen findet sich in [13].

L2-Größe	Beschreibung
bigtnum	Anzahl der *big tower* im LAr-Kalorimeter
cpvmax	maximaler Eintrag im z-Vertex-Histogramm (siehe Abschnitt 3.1.4)
cpvpos	Position des maximalen Eintrag im z-Vertex-Histogramm (siehe Abschnitt 3.1.4)
cpvsum	Summe der Einträge im z-Vertex-Histogramm (siehe Abschnitt 3.1.4)
larife	Energie im inneren Vorwärtsteil des LAr-Kalorimeters
laretot	totale (gewichtete) Enegie im LAr-Kalorimeter
mbigthe	mittlerer Winkel ϑ aller *big rays*
nbigray	Anzahl der *big rays*
trhineg	Anzahl der Spuren negativ geladener Teilchen mit hohem Impuls (siehe Abschnitt 3.1.4)
trhipos	Anzahl der Spuren positiv geladener Teilchen mit hohem Impuls (siehe Abschnitt 3.1.4)
trloneg	Anzahl der für Spuren negativ geladener Teilchen mit niedrigem Impuls (siehe Abschnitt 3.1.4)
trlopos	Anzahl der für Spuren positiv geladener Teilchen mit niedrigem Impuls (siehe Abschnitt 3.1.4)
trtot	Gesamtzahl der Spuren (siehe Abschnitt 3.1.4)

Tabelle 7: L2-Größen: Kandidaten für Netzeingabegrößen

5.7 Trainingsdaten

Aufgrund der vorhergehenden Abschnitte, wurden unter anderem folgende Sätze von L2-Größen für die Trainigsdaten verwendet:

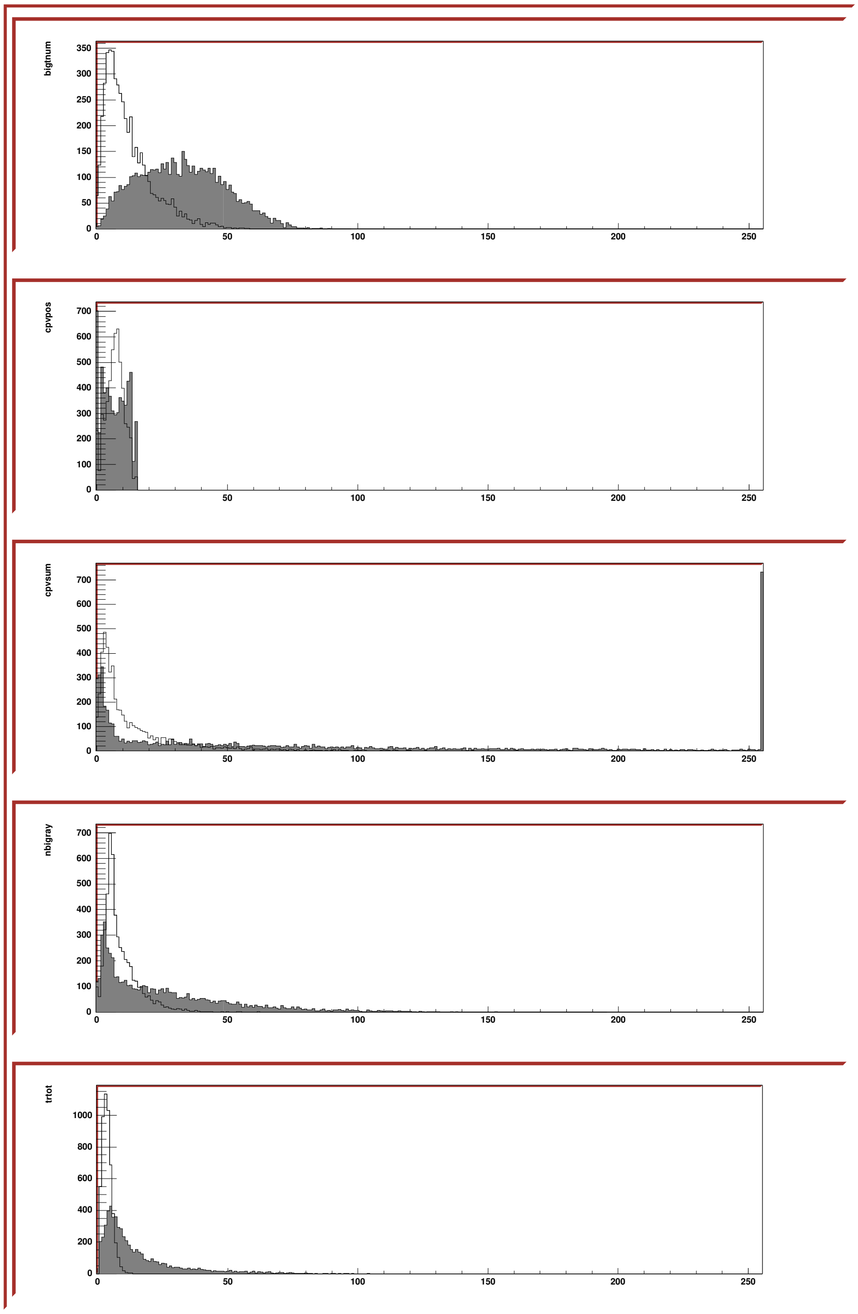

Abbildung 38: eindimensionale Verteilungen L2-Größen aus dem Satz A (siehe Abschnitt 5.7) und die Größe cpvpos: Das grau gefüllte Histogramm bilden die Untergrundereignisse, das leere Histogramm die Vertexereignisse. Eine effiziente Trennung mittels eindimensionaler Schnitte ist nicht möglich.

Satz A	:=	(bigtnum, cpvsum, nbigray, trtot)
Satz B	:=	(bigtnum, cpvmax, cpvpos, cpvsum, larife, laretot, nbigray)
Satz C	:=	(bigtnum, cpvmax, cpvpos, cpvsum, mbitthe, nbigray, trhineg, trhipos, trloneg, trlopos, trtot)

Ausgehend von den Ereignissen der betrachteten L2-L4-*transparent run* stehen 5779 Ereignisse der Klasse *big event* (Schnitt (13) und 3035 Ereignisse der Klasse Vertexereignis (Schnitt (14) zur Verfügung. Für jeden der Sätze A, B, C von L2-Größen werden Trainingsdaten (A, B, C) aus 4000 Ereignissen der Klasse *big event* und Testdaten (A, B, C) aus je 1500 Ereignissen der Klasse *big event* und Vertexereignis gebildet.

6 Netztraining

6.1 Training mit den Algorithmen `saf` und gng auf Realdaten

Mit den Trainingsdatensätzen A, B, C wurden Netze mit den Algorithmen `saf` und `gng` trainiert. Eine Zusammenstellung der Ergebnisse befindet sich in Tabelle 8. Netze mit weniger Paramtern, d. h. Knoten, scheinen besser generalisieren zu können.

Für Trainingsergebnisse mit „nichtverschwindendem“ $\epsilon(\mathbb{G})$ befinden sich die Effizienzplots $\mathbb{G}$ in Abbildung 39.

Trainings-datensatz	Algo-rithmus	max. Knotenzahl	max. Epochen	max. $\epsilon(\mathbb{G})$	in Trainings-schritt	in Epoche
A	saf	5	300	0.31	12000	3
A	saf	10	300	0.41	4000	1
A	saf	15	300	0.50	12000	3
A	saf	30	300	0.49	4000	1
A	saf	60	300	0.49	4000	1
A	gng	30	400	0.043	320000	80
A	gng	60	200	0.094	160000	40
B	saf	10	300	0.24	1076000	269
B	saf	15	300	0.0087	1076000	269
B	saf	30	300	0.005	2012000	503
B	saf	60	300	0.057	1536000	384
B	gng	30	400	0.03	732000	183
B	gng	60	200	0.073	148000	37
C	saf	15	300	0.207	44000	11
C	saf	30	300	0.30	1236000	309
C	gng	30	400	0.013	220000	55
C	gng	60	200	0.069	160000	40

Tabelle 8: Tabelle einger Trainings mit Angabe von maximalen $\epsilon(\mathbb{G})$ (siehe Abschnitt 4.4.1)

Die Netzentscheidung bzw. Effizienz für Ereignisse spezielle Physikklassen bleibt nocht zu untersuchen.

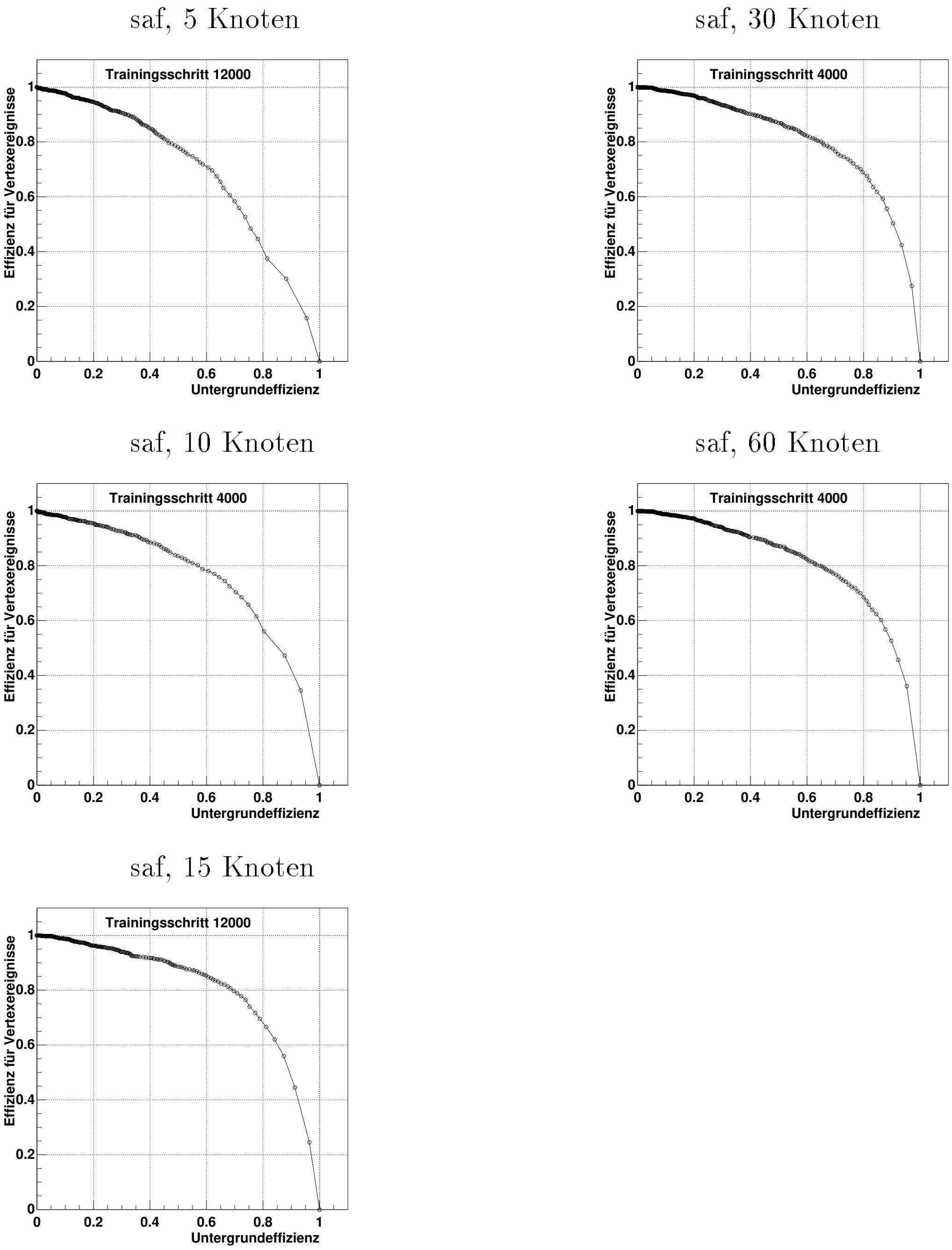

Abbildung 39: Effizienzplots zu einigen Trainings aus Tabelle 8

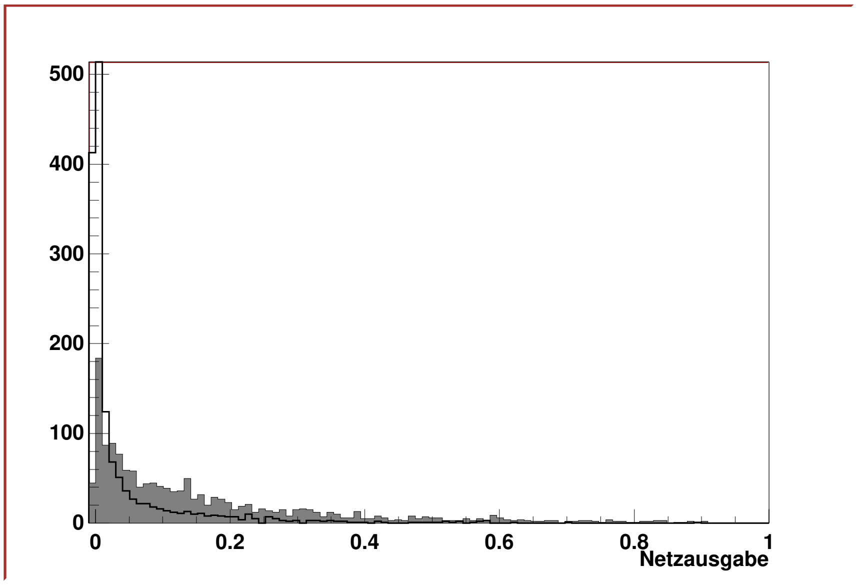

Abbildung 40: Netzausgabe für die Testdaten des mit `saf` trainierten Netzes mit 30 Knoten: Das graue Histogramm wird durch die Untergrundereignisse gebildet, das leere Histrogramm durch die Vertexereignisse.

6.2 Vergleichstraining auf dem CNAPS-Parallelrechner

Zum Vergleich wurden aus den Sätzen zum Training geeigneter L2-Größen auch Trainingsdaten für ein Zweiklassentraining gebildet. Je 2000 bzw. 1000 je Ereignisklasse *big event* und Vertexereignis wurden für die Trainings- bzw. Testdaten verwendet.

Für Trainings- bzw. Testdaten, die mit dem Satz A von L2-Größen gebildet wurden, und einem dreilagigem Netz mit 4 Knoten in der ersten Schicht, 20 in der versteckten Schicht und einem Ausgabeknoten sind die in Abbildung 41 dargestellten Ergebnisse nach 9968 Epochen mit dem Programm `bpexp2` [40, 41, 39] auf dem CNAPS-Parallelrechner erzielt worden.

Ein direkter Vergleich zu den Ergebnissen des Einklassentrainings ist jedoch nicht möglich, da das Zweiklassentraining durch die zweite Klasse wesentlich mehr Information in den Trainingsdaten erhält.

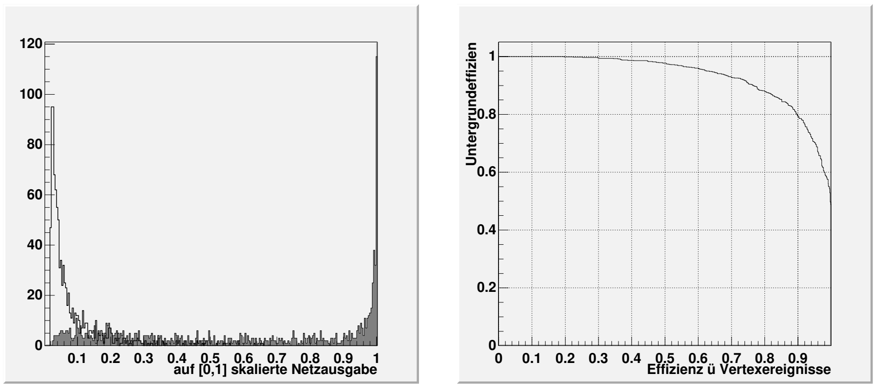

Abbildung 41: Links ist die auf $[0, 1]$ normierte Netzausgabe dargestellt. Dabei bildet die Netzausgabe für *big event*-Testdaten das grau gefüllte Histogramm, die Netzausgabe für Vertexereignisse das leere Histogramm. Rechts ist die Effizienzkurve ($\epsilon(\mathbb{G}) = 0.71$) dargestellt. Da in dieser Anwendung die Untergrundereignisse erkannt werden sollen, sind die Achsen gegenüber der üblichen Darstellung vertauscht.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde gezeigt, daß sich *big events* enkapsulieren lassen.

Erst nach Untersuchungen zur Netzeffizienz für interessante Physikereignisklassen kann jedoch der *background encapsulator* im L2NN in Betrieb genommen werden. Dazu ist noch der Auswertealgorithmus für die neuronalen Netze auf dem CNAPS-Parallelrechner zu implementieren. Die Hauptschwierigkeit in der Implementierung liegt bei der Berechnung des „elliptischen" Abstandsbegriffes

$$\sum_{j=1}^{d} \frac{(x_j - y_j)^2}{\sigma_j}.$$

Nach einer Abschätzung [42] kann diese jedoch in 7 Taktzyklen bei 20 MHz berechnet werden, wodurch ausreichend vielen Prozessorknoten (ca. 60) möglich sind.

Eine Steuerung der Triggerrate ist über die Veränderung der Ausgabeschwelle oder „Abschalten" einzelner Knoten möglich.

Ein Konkurrenzalgorithmus besteht im Trainingsprogramm `cluclu`, für welches die Netzauswertung für den CNAPS bereits implementiert ist.

Weitere Untersuchungen sollen die Trainingsergebnisse der alternativen Zugänge vergleichen.

Der Vorteil der in dieser Arbeit entwickelten Trainingsalgorithmen liegt in ihrer klaren und flexiblen Implementierung.

A Dokumentation

In diesem Abschnitt werden einige Programme beschrieben, die im Rahmen dieser Arbeit erstellt wurden:

Programm	Programmpaket	Aufgabe
`ntfill`	`ntfill`	Extraktion von POT-Daten und Speicherung in `Ntuple`
`l2sel`	`l2sel`	Datenanalyse und Bereitstellung von Trainings-/Testdatensätzen in Form von ASCII-Dateien
ROOT-`macros`	`l2sel`	Darstellung der von `l2sel` aufbereitetn Daten
`asc2bexx`	`bexx`	`perl`-Skript zur Konvertierung von Trainingsdaten
`saf` `gng` `gcs` `kohonen`	`bexx`	Trainingsalgorithmen
`stat-root`	`bexx`	Konvertierung der statistischen Information, wie z.B. Anzahl der Knoten bzw. Verbindungen und Netzergebnisse für Testdaten, in ROOT `TTree`s
`bexx-2d-root`	`bexx`	Konvertierung von Informationen über Netztopologie, Netzergebnisse und Trainings-/Testdaten ROOT-Objekte (`TTree`, `TH2F`) im Hinblick auf ein graphische Darstellung
ROOT-`macros`	`bexx`	Darstellung der von `stat-root` und `bexx-2d-root` aufbereiteten Daten

Diese Programme lassen sich wie folgt zu Paketen zusammenfassen:

Programmpaket	Sprache	*lines of code*
`ntfill`	FORTRAN 77	$\approx 3.9 \cdot 10^3$
`l2sel`	C++	$\approx 13.3 \cdot 10^3$
`bexx`	C++, perl	$\approx 26.7 \cdot 10^3$

A.1 `ntfill`

Der zentrale Punkt des Programms `ntfill` ist eine Schleife, in der jeweils die Daten zu einem Ereignis durch FPACK eingelesen werden und durch BOS-Bänke [6] zur

Verfügung gestellt werden. In dieser kann dann auf Ereignisinformationen, wie z. B. Anzahl der Signale in der CJC, ..., zugegriffen werden.

Daten, die für die Selektion von *big events* interessant sind, werden in Form eines `Ntuple`s in eine HBOOK-Datei geschrieben.

Grob können die Daten folgendermaßen eingeteilt werden:

- „organisatorische„ Daten: *run*-Nummer, *event*-Number, Phase, ...
- Triggerinformation
- Daten der Subdetektoren
- rekonstruierte Größen: Spuren, Vertexhypothesen

Eine detaillierte Beschreibung der im `Ntuple` gespeicherten Größen enthält folgende Tabelle:

module: ntc1nt		
ntc1Run	nccrun : run number	common BOSMDL
ntc1Evnt	nevent : run number	common BOSMDL
ntc10val	number of type-0-vertex .ge. 1	DVER : DST list of vertices
ntc10x	type of vertex = 0 : default run vertex X_VE : x (cm)	DVER : DST list of vertices
ntc10y	type of vertex = 0 : default run vertex Y_VE : y (cm)	DVER : DST list of vertices
ntc10z	type of vertex = 0 : default run vertex Z_VE : z (cm)	DVER : DST list of vertices
ntc11val	number of type-1-vertex .ge. 1	DVER : DST list of vertices
ntc11x	type of vertex = 1 : primary (z vertex from event) X_VE : x (cm)	DVER : DST list of vertices
ntc11y	type of vertex = 1 : primary (z vertex from event) Y_VE : y (cm)	DVER : DST list of vertices

ntc11z	type of vertex = 1 : primary (z vertex from event) Z_VE : z (cm)	DVER : DST list of vertices
ntc14val	number of type-4-vertex .ge. 1	DVER : DST list of vertices
ntc14x	type of vertex = 4 : alternative primary hypothesis X_VE : x (cm)	DVER : DST list of vertices
ntc14y	type of vertex = 4 : alternative primary hypothesis Y_VE : y (cm)	DVER : DST list of vertices
ntc14z	type of vertex = 4 : alternative primary hypothesis Z_VE : z (cm)	DVER : DST list of vertices
ntc1ldmi	number of rows of DMIS bank	DMIS
ntc1nhcj	sum of NHIT_CJC : total number of CJC hits	DMIS
ntc1nhra	sum of NHIT_RAD : total number of F-Radial hits	DMIS
ntc1nhpl	sum of NHIT_PLA : total number of F-Planar hits	DMIS
ntc1ldtn	number of rows of DTNV bank -> guess for number of non vertex fitted tracks	DTNV : DST Non-vertex fitted tracks, all tracks included
module: cjv4nt	CJC reconstruction from filter farm	CJV4
cjv4len	number of rows in CJV4 bank	CJV4
cjv4alen	min{cjv4len, 99}	CJV4
cjv4zver(cjv4alen)	z vertex coordinate	CJV4
cjv4hits(cjv4alen)	average number of hits per track	CJV4
cjv4ntr(cjv4alen)	number of tracks at z vertex	CJV4

module: dmis2nt	DST \miscellaneous\ data (scalar data or fixed length arrays). All data is for nominal (t0) bunch crossing if not stated brackets can be addressed as array elements.	95 Format:
dmislen	number of rows in DMIS bank	DMIS
dmisl4dc	KL4DEC : L4 decision word (1. word Y4TT bank) KL4DEC : L4 decision word (1. word Y4TT bank)	DMIS
dmiseel	E_ELET : Energy in electron tagger (GeV)	DMIS
dmieel44	E_EL44 : Energy in z=44m electron tagger (GeV)	DMIS
dmisseph	E_PHOT : Energy in photon tagger (GeV)	DMIS
dmisevet	E_VETO : Energy in veto counter (GeV)	DMIS
dmisnhcj	NHIT_CJC : total number of CJC hits	DMIS
dmisnhra	NHIT_RAD : total number of F-Radial hits	DMIS
mdirnhpl	NHIT_PLA : total number of F-Planar hits	DMIS
dmisnhfi	NHIT_IRO : number of forward IRON hits	DMIS
module: l1es2nt		
l1esnsub	number of level 1 subtrigger (max. NSUBTR)	h1util L1PRSC
l1espsfc(l1esnsub)	level 1 trigger prescale factors	h1util L1PRSC
l1ess1l1(l1esnsub)	level 1 actual subtriggers	h1util IL1MINL2

l1ess1l2(l1esnsub)	level 1 actual subtriggers after level 2 Level 1 actual subtriggers that do not contribute to an L2keep are switched off.	h1util Il1MINL2
l1essebl(l1esnsub)	level 1 subtrigger enable bits	h1util L1MASKS
l1essphy(l1esnsub)	level 1 physics trigger flags	h1util L1MASKS
l1essmon(l1esnsub)	level 1 monitor trigger flags	h1util L1MASKS
l1esravl	if online TLV1 bank exists	h1util UNTLV1
l1esraws(l1esnsub)	level 1 actual subtrigger bits	h1util UNTLV1
l1esacts(l1esnsub)	level 1 raw subtrigger bits	h1util UNTLV1
module: l2es2nt		
l2esnsys	NL2SY : number of level 2 systems 0 : L2NN 1 : L2TT 2 : L2HH	h1util UNTEL2
l2esntes	NL2TEPSY : number of level 2 trigger elements per system	h1util UNTEL2
l2estel(3,16)	level 2 trigger elements	h1util UNTEL2
l2esntrg	NL2SUBTR : number of level 2 subtriggers	h1util L2OVRI
l2esovfc(l2esntrg)	l2 subtrigger override factors	h1util L2OVRI
l2esovbt(l2esntrg)	override enable bits	h1util L2OVRI
l2esprsc(l2esntrg)	l2 subtrigger prescale factors	h1util L2PRSC
l2esstva	validation for valiables l2esurws, l2esvabt, l2esdwns, l2esovrs	h1util UNTLV2
l2esurws(l2esntrg)	unvalidated raw l2 subtriggers	h1util UNTLV2
l2esvabt(l2esntrg)	subtrigger validation bits	h1util UNTLV2
l2esdwns(l2esntrg)	downscaled l2 subtriggers	h1util UNTLV2
l2esovrs(l2esntrg)	overridden l2 subtriggers	h1util UNTLV2
module: gi2nt		
giphase	run phase	h1util TPHASE

module: jrdt2nt		
jrdtvers	VERSION = 1 + LCONFG(setup configuration=1..3)	h1util JRDATA, HEAD
jrdtverv	validation of jrdtvers	h1util JRDATA
jrdtnrun	NRUN : run number	h1util JRDATA, HEAD
jrdtnruv	validation of jrdtnrun	h1util JRDATA
jrdtevnt	NEVENT : event number	h1util JRDATA, HEAD
jrdtevnv	validation of jrdtevnt	h1util JRDATA
jrdtewgt	EWEIGHT : Event weight = Min Prescale factor active in this event	h1util JRDATA, HEAD
jrdtewgv	validation of jrdtewgt	h1util JRDATA
jrdtrtyp	RUNTYPE : Run type = 0 for H1 Data = 1 Monte Carlo = 2 for CERN Test Data = 3 for MC CERN Test Data	h1util JRDATA, HEAD
jrdtrtyv	validation of jrdtrtyp	h1util JRDATA
jrdtbrer	BRERROR : combined processing error code 1bit per branch: 0 if ok, to be filled by level 4	h1util JRDATA, HEAD
jrdrbrev	validation of jrdtbrer	h1util JRDATA
jrdtdtst	DETSTAT : Global detector status, single bits, 0 if ok 0-15 BBL3 alarm bits 16-31 HV bits	h1util JRDATA, HEAD
jrdrdtsv	validation of jrdtdtst	h1util JRDATA
jrdtl1cl	L1CLASS : Level 1 classification, bit coded 0: 1: LAr trigger 2: BEMC trigger 3: Track trigger 4: eTag trigger 5: Muon trigger	h1util JRDATA, HEAD

jrdtl1cv	validation of jrdtl1cl	h1util JRDATA
jrdtl2cl	L2CLASS : Level 2 classification	h1util JRDATA, HEAD
jrdtl2cv	validation of jrdtl2cl	h1util JRDATA
jrdtl3cl	L3CLASS : Level 3 classification	h1util JRDATA, HEAD
jrdtl3cv	validation of jrdtl3cl	h1util JRDATA
jrdt4cl	L4CLASS : Level 4 classification	h1util JRDATA, HEAD
jrdt4cv	validation of jrdt4cl	h1util JRDATA
jrdtl5cl	L5CLASS : Level 5 classification	h1util JRDATA, HEAD
jrdtl5cv	validation of jrdtl5cl	h1util JRDATA
jrdtnebu	NEBUNCH : Npbunch*2**16 + Nebunch (0..219) (0..219)	h1util JRDATA, HEAR
jrdtnebv	validation of jrdtnebu	h1util JRDATA
module: trk2nt		
trkldtnv	number of rows of DTRA bank -> guess for number of vertex fitted tracks DTRA : DST vertex fitted tracks all vertex fitted tracks including multipe	fit hypotheses and \V0\ tracks ...
trkldtnv	number of rows of DTNV bank -> guess for number of non vertex fitted tracks	DTNV : DST Non-vertex fitted tracks, all tracks included
trklcjk4	number of rows of CJK4 bank -> guess for number of tracks reconstructed by filter farm	CJK4 : CJC reconstruction from filter farm
module: htk2nt	information of hits belonging to (non) vertex fitted tracks	DTNV, DTRA, DVER

htknlen	number of rows in DTNV bank 0 : Bank does not exist or is empty.	DTNV : DST Non-vertex fitted tracks, all tracks included
htknval	number of tracks with zero in PLA und RAD bitfields Only in this case the bitfields for BPC, CJ1, CJ2, CIZ, COZ can be interpreted properly.	DTNV
htknsbpc	sum of BPC hits for DTNV tracks	DTNV
htkn0bpc	number of tracks with no hits in BPC	DTNV
htknscj1	sum of CJ1 hits for DTNV tracks	DTNV
htkn0cj1	number of tracks with no hits in CJ1	DTNV
htknscj2	sum of CJ2 hits for DTNV tracks	DTNV
htkn0cj2	number of tracks with no hits in CJ2	DTNV
htknsciz	sum of CIZ hits for DTNV tracks	DTNV
htkn0ciz	number of tracks with no hits in CIZ	DTNV
htknscoz	sum of COZ hits for DTNV tracks	DTNV
htkn0coz	number of tracks with no hits in COZ	DTNV
htknspla	sum of PLA hits for DTNV tracks	DTNV
htkn0pla	number of tracks with no hits in PLA	DTNV
htknsrad	sum of RAD hits for DTNV tracks	DTNV
htkn0rad	number of tracks with no hits in RAD	DTNV
htknscvl	= 6 : length of the arrays htknsc1v(htknscvl) htknsc2v(htknscvl)	htk2nt

htknsc1v(htknscvl)	sum of CJ1 hits for DTNV tracks with |z at DCA| <= index * 10 cm	DTNV
htknsc1v(htknscvl)	sum of CJ2 hits for DTNV tracks with |z at DCA| <= index * 10 cm	DTNV
htkntrkv(htknscvl)	number of DTNV tracks with |z at DCA| <= index * 10 cm	DTNV
htkvlen	number rows in DTRA bank 0 : Bank does not exist or is empty.	DTRA : DST vertex fitted tracks
htkvcnt(8)	number of DTRA tracks belonging to a vertex type given by the index index : vertex type 1+o : default run vertex 2+o : primary (z vertex from event) 3+o : decay 4+o : secondary interaction 5+o : alternative primary hypothesis 6+o : DVER vertex type not in 0,1,2,4 7+o : Link to vertex in DVER is 0. 8+o : Bank DVER not found. Tho offset o=0 for FORTRAN/HBOOK and o=-1 for C++/ROOT. index 1,2,3,4,5: DVER vertex type + 1 index 6,7,8: if something goes wrong.	DTRA, DVER
htkvsbpc(8)	sum of BPC hits for DTRA tracks belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER

htkv0bpc(8)	number of tracks with no hits in BPC belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkvscj1(8)	sum of CJ1 hits for DTRA tracks belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkv0cj1(8)	number of tracks with no hits in CJ1 belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkvvcj2(8)	number of DTRA tracks with zero in PLA and RAD Only in this case the bitfield for CJ2 can be interpreted properly.	DTRA
htkvscj2(8)	sum of CJ2 hits for DTRA tracks with PLA=RAD=0 belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkv0cj2(8)	number of tracks with no hits in CJ2 with PLA=RAD=0 belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER

htkvsciz(8)	sum of CIZ hits for DTRA tracks belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkv0ciz(8)	number of tracks with no hits in CIZ belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkvscoz(8)	sum of COZ hits for DTRA tracks belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkv0coz(8)	number of tracks with no hits in COZ belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkvspla(8)	sum of PLA hits for DTRA tracks belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkv0pla(8)	number of tracks with no hits in PLA belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkvsrad(8)	sum of RAD hits for DTRA tracks belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER

htkv0rad(8)	number of tracks with no hits in RAD belonging to a vertex type given by the index See htkvcnt(8) for explanation of the index.	DTRA, DVER
htkvscvl	= 6 : length of the arrays htkvsc1v(htkvscvl) htkvsc2v(htkvscvl)	DTRA
htkvsc1v(htkvscvl)	sum of CJ1 hits for DTRA tracks with |z at DCA| <= index * 10 cm	DTRA
htkvsc1v(htkvscvl)	sum of CJ2 hits for DTNV tracks with |z at DCA| <= index * 10 cm	DTRA
htkvtrkv(htkvscvl)	number of DTNV tracks with |z at DCA| <= index * 10 cm	DTRA

A.2 `l2sel`

Das Programm `l2sel` arbeitet auf ROOT-`TTree`s, welche aus von `ntfill` erzeugten `Ntuple`s gebildet werden und erfüllt folgende Aufgaben:

1. Analyse der Daten für die Gewinnung eines Kriteriums für *big events*
2. Auswahl für das Training geeigneter L2-Größen anhand von Korrelationen bzw. Abweichungen von Mittelwerten und Standardabweichungen für Daten verschiedener Klassen
3. Aufbereitung der Daten in ein für das Training geeignetes Format

Im Hauptprogramm von `l2sel` werden für die „Branchs“ des `TTrees` („Variablen des `Ntuples`“) globale Variablen deklariert, welche in einem zentralen *eventloop* gefüllt werden. In diesem werden für jedes Ereignis mehrere Module aufgerufen, welche Teile obiger Aufgaben erfüllen.

Um für alle 3 Punkte Schnitte auf einheitliche Weise behandeln zu können, werden Schnitte als C-Funktionen implementiert, welche ihre Entscheidung aufgrund der globalen „Variablen des `Ntuples`“ fällen. Z. B.

```
Int_t isBigEvent(void)
   {
      return ...;
   }
```

Obwohl ROOT es ermöglicht, bei einer Schleife über einen `TTree` nur ausgewählte Zweige (`TBranch`) einzulesen, wodurch erhebliche Zeitersparnis durch die entsprechend ausgelegte Datenstruktur `TTree` entsteht, wurde davon kein Gebrauch gemacht, da in den vielen *cut*-Funktionen die Verwendung der „Variablen des `Ntuples`" verborgen ist. Daher werden für jedes Ereignis alle Variablen eingelesen.

Für Punkt 1 der Aufgaben werden zahlreiche Histogramme unterschiedlicher Größen (Variablen des `Ntuples` und berechnete Größen wie z.B. `isL2Keep()`) benötigt. Dabei treten folgen Anforderungen auf:

- *cuts* als C-Funktion
- Speicherung des fertigen Histogramms in einem ROOT-File
- große Anzahl verschiedener *cuts*
- große Anzahl verschiedener Größen
- Zu histogrammierende Größen liegen als Variable, Array (z. B. L1ST) oder Erbgebnis einer Funktion (z. B. `isL2Keep()`, ...) mit unterschiedlichen Datentypen Int_t, UInt_t, Float_t vor.
- eindimensionale, zweidimensionale Histogramme
- Textausgabe von L2-Größen für die Erzeugung von Traininsgdatensätzen.

Um die Anforderungen einheitlich durch leicht wartbaren Programmcode erfüllen zu können, wurde folgende Klassenstruktur entwickelt, bei der darauf geachtet wurde, Funktionalität so weit wie möglich in Basisklassen zu implementieren:

```
LJFillHistoUIntArrayCut
   LJFillHistoUIntArrayProdCut
LJLoopAction
   LJLoopActionNamed
      LJLoopActionCut
         LJLoopCalcCorrelationBase
            LJLoopCalcCorrelationFunctionFunction
            LJLoopCalcCorrelationL2QuantityFunction
```

```
            LJLoopCalcCorrelationVarFunction
            LJLoopCalcCorrelationVarVar
         LJLoopFillHBase
            LJLoopFillH1Base
               LJLoopFillH1
               LJLoopFillH1Array
                  LJLoopFillH1ArrayProd
                  LJLoopFillH1ArrayProdThreshold
               LJLoopFillH1Function
               LJLoopFillZVertexHistogram
               LJLoopFillZVertexHistogramShifted
            LJLoopFillH2Base
               LJLoopFillH2
               LJLoopFillH2L2Quantities
         LJLoopTextOutputBase
            LJLoopTextOuputTData
LJLoopActionCuts
   LJLoopFillH1ArrayCuts
   LJLoopFillH1ArrayProdCuts
   LJLoopFillH1Cuts
   LJLoopFillH1FunctionCuts
   LJLoopFillZVertexHistogramCuts
   LJLoopFillZVertexHistogramShiftedCuts
LJLoopCalcCorrelationL2QuantitiesFunction
LJLoopFillH2L2QuantitiesAll
LJZVertexHistogram
   LJZVertexHistogramShifted
```

Die Klasse `LJLoopAction` stellt mit den Funktionen `action(int)` bzw. `operator()(int)` die gleiche Schnittstelle für aus dem *event loop* zu rufende Module zur Verfügung. Als Argument wird nur die Nummer des bearbeiteten Ereignisses übergeben. In Abhängigkeit von dieser werden die virtuellen Funktionen `init()`, `fill()` oder `final()` aufgerufen.

Die Verwaltung von Name, Titel und Verzeichnisname für die Speicherung der berechneten Daten ist Aufgabe von `LJLoopActionNamed`.

Die Behandlung von Schnitten über den selektierten Aufruf der Elementfunktion `fillStep()` erfolgt in `LJLoopActionCut`.

Da zur Gewinnung des Kriteriums für *big events* eine große Anzahl von verschiedenen Schnitten getestet worden ist, wurde ein Mechanismus implementiert, der auf einem `Array` von Schnitten arbeitet. Die von `LJLoopActionCuts` abgeleiteten Klassen führen ihre Aktionen für jeden dieser Schnitte aus.

Ein weiterer, später entdecktern Ansatz, die boolschen Schnittinformationen als zusätzliche „Zweige" (`TBranch`) nachträlich hinzuzufügen hat foldgende Vor- bzw. Nachteile: Da die Ausgangsdaten in Form mehrere `TTrees` in verschiedenen Dateien vorliegen, die mittels `TChain` zusammenfaßt wurden, müssen alle Daten der `TTrees` in einen neuen `TTree` in eine Datei kopierte werden, bevor die zusätzlichen „Zweige" hinzugefügt werden können, da diese nachträgliche Ergänzung nur bei `TTrees` möglich ist, die sich in einer Datei befinden. Dadurch wird zusätlicher Speicherplatz benötigt. Von Vorteil ist hingegen, daß das Vorliegen der boolschen Schnittinformationen als eigene „Zweige" ein einfaches interaktives Arbeiten ermöglicht. Dieser Ansatz entbehrt jedoch noch der einheitlichen Speicherung von z. B. erzeugten Histogrammen, die der zuerst vorgestellte Ansatz bietet.

A.3 Namenskonvention für C++

Für die im Rahmen dieser Arbeit entwickelte C++TL BEXX besteht folgende, durch Smalltalk beeinflußte, Namenskonvention:

Klassennamen	beginnen mit `LJ`; folgende Wortteile beginnen mit Großbuchstaben, ansonsten Kleinbuchstaben
Variablennamen	beginnen mit Kleinbuchstaben; folgende Wortteile beginnen mit Großbuchstaben, ansonsten Kleinbuchstaben
Macros	Großbuchstaben
Templateargumente	beginnen mit `T_`; Großbuchstaben
private Datenelemente	wie Variablennamen; enden mit `_`; Möglichkeit gleichnamiger Zugriffsfunktion
Funktionsnamen	wie Variablennamen

[12]

[12] In verschiedenen C++-Klassenbibliotheken, wie ROOT oder Bibliotheken zu Borland C++, besteht die Konvention, Klassennamen mit T beginnen zu lassen. Damit solche Bibliotheken, insbesondere ROOT, zusätzlich genutzt werden können ist auf „`LJ`" ausgewichen worden. Bsp.: `TArray` $\Longleftrightarrow$ `LJArray`

A.4 BEXX, C++TL

Für die flexible und übersichtliche Implementation von unüberwachten Trainingsalgorithmen wurde die C++-*template library* BEXX[13] entwickelt. Die zentralen Anforderungen sind:

- einheitliche Behandlung von Ein- und Ausgabe, die einfaches Laden und Speichern von z. B. Netztopologien und Trainings-/Testdaten ermöglicht. Umwandlung von als ASCII-Dateien vorliegenden Trainings/Testdaten in dieses spezielle Format kann durch das `perl`-Skript `asc2bexx` erfolgen.
- Bereitstellung von *container*-Klassen, wie z. B. `LJArray`, `LJSet`, `LJGCIDDictionary`
- Implementation gleicher Funktionalitäten der verschiedenen Trainingsalgorithmen, wie z. B. die Behandlung von Knoten und Verbindungen, die Suche nach dem *winner*-Knoten, in Basisklassen, um mehrfachen, gleichen Programmcode zu vermeiden

Den ersten beiden Punkten könnte entgegnet werden, das die C++-*standard template library* (C++STL) bzw. die Klassen der ROOT-*frameworks* dies Aufgabe erfüllen könnten. Da zumindest zu Beginn dieser Arbeit die C++STL nicht zur standard Installation unter AIX 3.2 am MPI für Physik gehörte und noch keine Vertrautheit mit ROOT bestand, wurde dieser eigenständige Ansatz gewählt. Dadurch ist eine hohe Portabilität gewährleistet (siehe Tabelle 10), trotz Unterschieden bei der C++ interpretation durch die verschiedenen Compiler (insbesonder Behandlung von *templates*). Auch der Lerneffekt ist nicht zu unterschätzen.

Betriebssystem	Compiler
AIX 4.2.1	xlC
Linux 2.0.31	g++ 2.7.2.1
IRIX 6.2	g++ 2.7.2.1

Tabelle 10: Aufstellung der Betriebssysteme und Compiler unter denen die Traininigsalgorithmen der BEXX-*library* und das Datenselektionsprogramm `l2sel` erfolgreich getestet betrieben wurden

Eine detaillierte Beschreibung jeder Klasse der BEXX-*library* würde zu weit führen. Dazu sei hier leider nur auf den kommentierten Quellcode verwiesen und ein Übersicht über die Klassenstruktur gegeben.

[13] „background encapsulator ++“

```
LJArrayConstIterator
LJArrayIterator
LJBexxBaseTrainParams
   LJGCSBaseTrainParams
      LJGCSWithActFctNodesTrainParams
   LJGCSBaseTrainParamsTimeDependent
      LJGCSWithActFctNodesTrainParamsTimeDependent
   LJGNGBaseTrainParams
      LJGNGSimpleTrainParams
      LJGNGWithActFctNodesTrainParams
   LJGNGBaseTrainParamsTimeDependent
      LJGNGWithActFctNodesTrainParamsTimeDependent
   LJKohonenBaseTrainParams
   LJKohonenBaseTrainParamsTimeDependent
   LJSAFBaseTrainParams
   LJSAFBaseTrainParamsTimeDependent
LJDictionaryConstIterator
LJDictionaryIterator
LJGCIDDictionaryConstIterator
LJGCIDDictionaryIterator
LJKohonen2TrainParams
LJObject
   LJArray
      LJArrayEndContinued
         LJArrayEndContinuedInterpol
      LJMVector
      LJString
   LJArray2
   LJCounter
   LJDictionary
      LJIDDictionary
      LJSteerCards
   LJFunction
      LJActFctSphGauss
      LJActivationFunction
         LJActFctEllGauss
      LJFunctionGauss
   LJGCIDDictionary
   LJKohonen1
   LJNeuralNetwork
      LJGCS
```

```
    LJNNWithNodes
      LJKohonen2
      LJNNWithNodesConnections
        LJGCSBase
          LJGCSWithActFctNodes
        LJGNGBase
          LJGNGSimple
          LJGNGWithActFctNodes
        LJKohonenBase
          LJKohonenWithActFctNodes
      LJSAFBase
        LJSAFWithActFctNodes
      LJTTDBase
  LJObjectWithID
    LJGCSSimplex
    LJGNGEdge
    LJNode
      LJGNGNode
      LJNodeWithActFct
        LJNodeWithActFctCounter
      LJUSLSimpleNode
  LJPair
  LJSet
    LJIDSet
  LJSingle
  LJTDatum
LJSetConstIterator
  LJIDSetConstIterator
LJSetIterator
  LJIDSetIterator
LJTTDBaseTrainParams
TDY
```

Bemerkenswert ist die Klasse `LJGCIDDictionary`, welche die Funktionalität eines *dictionaries* mit schnellem Zugriff und Speicherverwaltung (*garbage collection*) zu Verfügung stellt. Mit dieser Klasse werden die Knoten und Verbinung der Trainingsalgorithmen verwaltet. Der schnelle Zugriff bei der Klasse `LJGCIDDictionary` hat somit entscheidenden Einfluß auf die Trainingszeiten.

Eine Anwendung der Klasse `LJSteerCards` wird im nächsten Abschnitt dargestellt

A.5 *steering cards*

Die Parameterübergabe an die meisten der oben beschriebenen Programme, wie z. B. die Lernparameter bei den Trainingsalgorithmen, die Dateinamen für Trainings-/Testdaten, erfolgt über Steuerkarten (*steering cards*) über STDIN. Eine Steuerkarte wird durch eine mit dem Kartenbezeichner beginnende Zeile und einer Folge von jeweils durch *white space* (außer *new line*) getrennte Datenelemente gebildet. Um die *steering cards* für mehrere Programme, z. B. für ein Netztraining und die anschließende Konvertierung der statistischen Daten, in einer Datei zusammenfassen zu können, sind die Kartenbezeichner eine Verbindung von Programmname und Kartenname durch „`::`".

Das Einlesen und der Zugriff auf die Daten der Steuerkarten ist Aufgabe der Klasse `LJSteerCards`. *default*-Werte für fehlende Steuerkarten könne angegeben werden. Für Überwachungs und Buchführungszwecke werden die angeforderten Werte von Steuerkarten ausgegeben.

Dadurch ist es z. B. möglich, den Satz von möglichen Steuerkarten mit den *default*-Werten für die Programme, wie z. B. `saf`, `gng`, `gcs`, `kohonen`, `stat-root`, `bexx-2d-root`, die diese Funktionalität benuzten, durch Angabe einer leeren Steuerdatei zu erhalten.

Tabellenverzeichnis

Abbildungsverzeichnis

Literatur

[1] I. Abt et al. The tracking calorimeter and muon detectors of the H1 experiment at HERA. *Nuclear Instruments and Methods in Physics Research A*, 386:348, 1996.

[2] I. Abt et al. The H1 detector at HERA. *Nuclear Instruments and Methods in Physics Research A*, 386:310–347, 1997.

[3] Clifford W. Aschley. *The Aschley Book of Knots*. Doubleday, New York, London, Toronto, Sydney, Auckland, 1944.

[4] Hanspeter Beck. Principles and Operation of the z-Vertex Trigger. H1-internal-Note H1-05/96-479, DESY, University of Zürich, Switzerland, 1996.

[5] Roland Bernet. *Production of $D^{\star\pm}$ Mesons measuered with the H1 Detector at HERA*. PhD thesis, A dissertation submitted to the Swiss Federal Institute of Technology Zurich for the degree of Doctor of Natural Sciences, Zurich, 1995.

[6] Volker Blobel. The BOS System, Dynamic memory management, Second updated printing, FORTRAN 77 Version. Technical report, II. Institut für Experimentalphysik, Universität Hamburg, December 1987.

[7] Thomas Boor, J. Hutter, and A. Pribas. *vi-Referenzhandbuch: Das Lehr- und Nachschlagewerk zum UNIX-Standardeditor*. Prentice Hall, München, London, Mexiko, New York, Singapur, Sydney, Toronto, erster korrigierter Nachdruck der ersten edition, 1996.

[8] Borland GmbH, Starnberg. *Borland C++ 3.0, Programmierhandbuch*, zweite verändete edition, 1992.

[9] Braune and Faessler. Physik und Technologie der Teilchendetektoren am LHC. In *Vorlesung LMU München*, Wintersemester 1995/96.

[10] René Brun, Fons Rademakers, Nenad Buncic, and Valery Fine. *The Root System Home Page*. http://root.cern.ch.

[11] Tancredi Carli, 22. Mai 1998. persönliche Mitteilung.

[12] M. Charlet. *The H1 central trigger form an offline-software writer's point of view; useful informtaion an how to access it*. H1, March 4 1997.

[13] *Data format of the Event vector*. http://wwwh1.mppmu.mpg.de/projects/neuro/-doc/eventvector/index.html.

[14] R. Eichler, J. Riedlberger, T. Wolff, S. Egli, S. Eichenberger, C. Meyer, K. Müller, P. Robmann, U. Straumann, and T. Truöl. Technical Proposal for a Driftchamber $r\varphi$-Trigger. H1-internal-Note H1-01/91-161, DESY, 7. May 1990.

[15] Bernd Fritzke. Growing Cell Structures - a Self-organizing Network for Unsopervised and Supervised Learning. In *International Computer Science Institute, Berkeley*, 1993.

[16] Frank Gaede. *Exklusive Produktion von ϕ-Mesonen in ep-Streuung am H1-Experiment bei HERA*. PhD thesis, Mathematisch-Naturwissenschaftliche Fakultät der Christian-Albrechts-Universität zu Kiel, Kiel, 1997.

[17] H1 SPACAL Group. The H1 Lead/Scintillating-Fibre Calorimeter. Technical Report DESY 96-171, DESY, August 1996.

[18] Dirk Hoffmann. Erste Erfahrungen mit den neuen Zwischenstufen (Level2) des H1-Triggers. In *Herbstschule für Hochenergiephysik, Maria Laach*, 1996.

[19] Homer. *Ilias*. Artemis Verlag, München und Zürich, achte edition, 1983.

[20] Ludger Janauschek. Kurzvortrag: Studium selbstorganisierender Trainingsalgorithmen im Hinblick auf Unterdrückung spezieller Untergrundereignisse am H1-Experiment durch einen neuronalen Trigger auf Level 2. In *Frühjahrstagung der Deutschen Physikalischen Gesellschaft in Freiburg*, 1998.

[21] Ludger Janauschek. *L4-Histogramme für L2*. Fortgeschrittenen Praktikum II, LMU München, DESY-Sommerstudentenprogramm, 1996.

[22] Nicolai Josuttis. *Die C++-Standardbibliothek; eine detaillierte Einführung in die vollständige ANSI/ISO-Schnittstelle*. Addison-Wesley-Longman, Bonn, erster korrigierter Nachdruck der ersten edition, 1996.

[23] Christian Kiesling. Anwendung neuronaler Netze in der Kern- und Teilchenphysik. In *Vorlesung LMU München*, Sommersemester 1996.

[24] Immo Kießling and Martin Lowes. *Programmierung mit FORTRAN 77*. Teubner, Stuttgart, vierte überarbeitete und erweiterte edition, 1987.

[25] Konrad Kleinknecht. *Detekoren für Teilchenstrahlung*. Teubner Studienbücher Physik, Stuttgart, dritte, durchgesehene und erweiterte edition, 1992.

[26] J. H. Köhne et al. Realization of a second level neural network trigger for the H1 experiment at HERA. *Nuclear Instruments and Methods in Physics Research A*, 389:128, 1997.

[27] Helmut Kopka. *LaTeX, Einführung*, volume 1, Einführung. Addison-Wesley, Bonn; Paris; Reading, Mass. [u. a.], 1994.

[28] Marcel Kunce and Johannes Steffens. Growing Cell Structure and Neural Gas; Incremental Neural Networks. In *To be published in the Proceedings of the 4th AIHEP Workshop, Pisa, World Scientific, 1995*, 1995.

[29] Vincent Lemaitre, 19. Mai 1998. persönliche Mitteilung.

[30] William R. Leo. *Techniques for Nuclear and Particle Physics Experiments.* Springer-Verlag, Berlin, ..., zweite, überarbeitete edition, 1994.

[31] Jürgen Möck. Einsatz neuronaler Netze als intelligente Trigger im H1-Experiment. Master's thesis, Fakultät für Physik der Technischen Universität München, München, März 1994.

[32] Jürgen Möck. *Untersuchung diffraktiver J/ψ-Ereignisse im H1-Experiment bei HERA und Entwicklung neuronaler Triggeralgorithmen.* PhD thesis, Fakultät für Physik der Technischen Universität München, München, September 1997.

[33] T. Nicholls, M. Charlet, J. Coughlan, E. Elsen, D. Hoffmann, H. Krehbiel, H.-C. Schultz Coulon, J. Schütt, and F. Sefkow. Concept, Design and Performance of the Second Level Triggers of the H1 Detector. In *IEEE 1997 Nuclear Science Symposium, Albuquerque, New Mexico, November 1997. Submitted to IEEE Transactions in Nuclear Science*, November 1997.

[34] J. Riedlberger. The H1 Trigger with Emphasis on Tracking Triggers. *Nuclear Physics B (Proc. Suppl.)*, 44:423–329, 1995.

[35] Raúl Rojas. *Theorie der neuronalen Netze; eine systematische Einführung.* Springer, Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona, Budapest, 1993.

[36] Robert Sedgewick. *Algorithmen.* Addison-Wesley, Bonn; Paris; Reading, Mass. [u. a.], dritte, unveränderte edition, 1994.

[37] Johannes Steffens and Marcel Kunze. Implementation of the Supervised Growing Cell Structure on the CNAPS Neurocomputer. In *Paper to be presented at ICANN'95, Paris*, 1995.

[38] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, Bonn; Paris; Reading, Mass. [u. a.], dritte edition, 1997.

[39] Steffen Udluft. *bpexp2 online manual.* http://wwwh1.mppmu.mpg.de/projects/neuro/doc/-bpexp2/manual/manual.html.

[40] Steffen Udluft. *Training und Einsatz neuronaler Netze auf dem CNAPS-Parallelcomputer.* Fortgeschrittenen Praktikum, LMU München, 1994.

[41] Steffen Udluft. Untersuchungen zu Neuronalen Netzen als Vertextrigger im H1-Experiment bei HERA. Master's thesis, Fakultät für Physik der Ludwig-Maximilans-Universität München, München, Mai 1996.

[42] Steffen Udluft, 13. Januar 1998. persönliche Mitteilung.

[43] verschiedene. Verschiedene Artikel zum Thema neuronaler Netze. *Scientific American*, September 1992.

[44] Thomas Wolff. *Entwicklung, Bau und erste Ergebnisse eines totzeitfreien Spurfinders als Trigger für das H1-Experiment am HERA Speicherring, Abhandlung zur Erlangung des Titels Doktor der Naturwissenschaften.* PhD thesis, Insitut für Teilchenphysik der Eidgenössischen Technischen Hochschule Zürich, Zürich, November 1993. DissNr. 10408 und ETHZ-IPP Internal Report 94-2.

[45] *Central Trigger Level 1 (CTL1) logic.* http://www-h1.desy.de/itrigger/TrigSetup/tdl.subtriggers.

[46] *Central Trigger Level 2 (CTL2) logic.* http://www-h1.desy.de/itrigger/TrigSetup/tdl.l2.

[47] *H1 Default prescale factors.* http://www-h1.desy.de/icgi-trigger/l1presc.pl.

[48] *L1 trigger elements.* http://www-h1.desy.de/itrigger/L1Documentation/-teldoc96.txt.

[49] *L2 trigger elements.* http://www-h1.desy.de/itrigger/L2Trigger/l2tel.html.

2 Danksagung

Für die Ermöglichung meines Physikstudiums danke ich meinen Eltern und meiner Schwester Barbara für gutes Zureden und fachlichen Rat.

Mein besonderer Dank gilt Herr Prof. Dr. Christian Kiesling für die Anregungung zu dieser Arbeit und die gute Betreuung. Ferner bin ich ihm und dem Max-Planck-Institut für Physik für die guten Arbeitsbedingungen und die Möglichkeit, an Konferrenzen und Seminaren teilzunehmen, dankbar.

Für hilfreiche Diskussionen und technische Ratschläge bedanke ich mich bei: René Brun, Tancredi Carli, Vladimir Chekelian, Frank Gaede, Dirk Hoffmann, Joseph Huber, Uwe Leuphold, Günter Manhart, Jürgen Möck, Fons Rademakers, Marlene Schaber, Robert Sütterlin, Steffen Udluft.

Hiermit versichere ich, die vorliegende Arbeit unter Angabe aller wesentlichen Quellen und Hilfsmittel selbständig verfaßt zu haben.

München, 28. Juli 1998

Ludger Janauschek