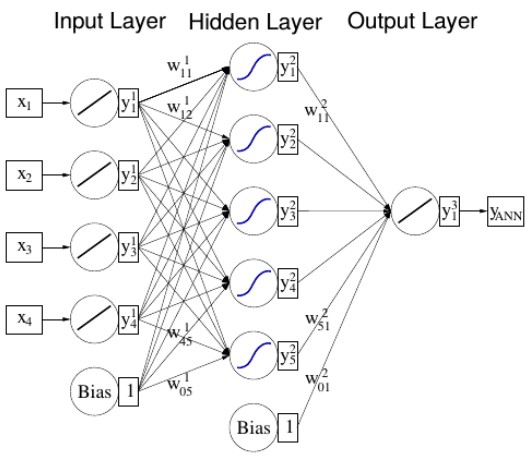
Moderne Methoden der Datenanalyse

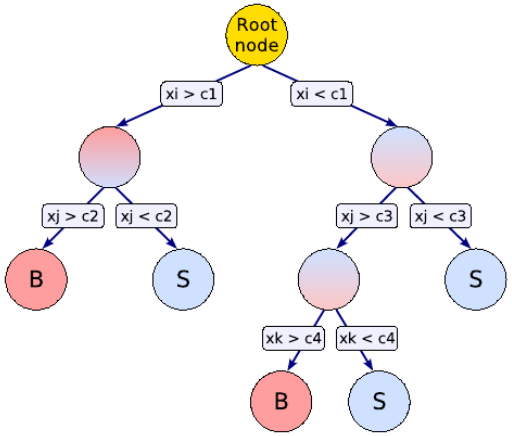
Klassifikation (ANN, BDT)

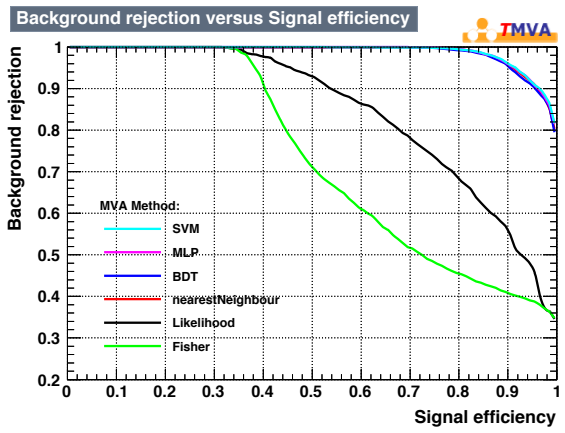
Andreas B.Meyer

Fakultät für Physik
Institut für Experimentelle Kernphysik - IEKP

4. Juli 2017







Vorlesungsprogramm

- Einführung: Überblick und grundlegende Konzepte (1)
- Einführung: Überblick und grundlegende Konzepte (2) — A.MEYER
- Zufallszahlen und Monte-Carlo Methoden

- Hypothesentests
- Parameterschätzung — R.ULRICH
- Parameterschätzung (Goodness-of-fit)
- Optimierungs- und Parametrisierungsmethoden

- Konfidenzintervalle (1)
- Konfidenzintervalle (2)
- Klassifikation (lineare Methoden) — A.MEYER
- Klassifikation (ANN, BDT)
- Klassifikation (SVM and Deep Learning)
- Messen und Entfalten
- Systematische Unsicherheiten

7.

KLASSIFIKATION

Klassifikation (2)

- Einlagiges Perceptron
- Gradientenabstiegsmethode
- Beispiel

7.5
Artificial Neural Networks

- Entscheidungsbäume
- Wald aus Entscheidungsbäumen und Boosting
- AdaBoost
- Beispiele

7.6
Boosted Decision Trees

- Literatur:
  - Hastie, Tibshirani, Friedmann, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*
  - Narsky/Porter: *Statistical Analysis Techniques in Particle Physics,* Wiley
  - I.Goodfellow and Y.Bengio and A.Courville: *Deep Learning Book* http://deeplearningbook.org
  - ROOT-Paket TMVA: *TMVA User Guide* http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf

Erinnerung: Supervised Learning

- Multivariate Analyse: Merkmalvektor folgt d-dimensionaler PDF
- Curse of dimensionality: d-dimensionales Histogramm des (endlich großen Datensatzes) ist für d groß praktisch leer $N/n_b^d \rightarrow 0$, für $d \rightarrow \infty$

- Supervised Learning: Bestimmung (“Training”) des Klassifikators anhand von vorklassifizierten Daten.
  - Beispiele: SPAM-filter, Postleitzahlen, Higgs-data challenge, u.v.m.
  - Auswahl und Vorverarbeitung der Eingangsvariablen zur Reduktion der Dimensionalität des Merkmalvektors.
  - In der Physik können Randbedingungen aus der Theorie verwendet werden
  - Training und Test der Verallgemeinerbarkeit mit statistisch unabhängigem, vorklassifizierten Testdatensatz (vermeide Unter- oder Übertraining)
  - Wahl der Methode je nach Problemstellung: Transparenz vs Mächtigkeit.

- Vorige Vorlesung: Bewährtes lineares und analytisches Verfahren: Fisher-Diskriminante (LDA)
- Heutige Vorlesung: nicht-lineare supervised-learning Verfahren

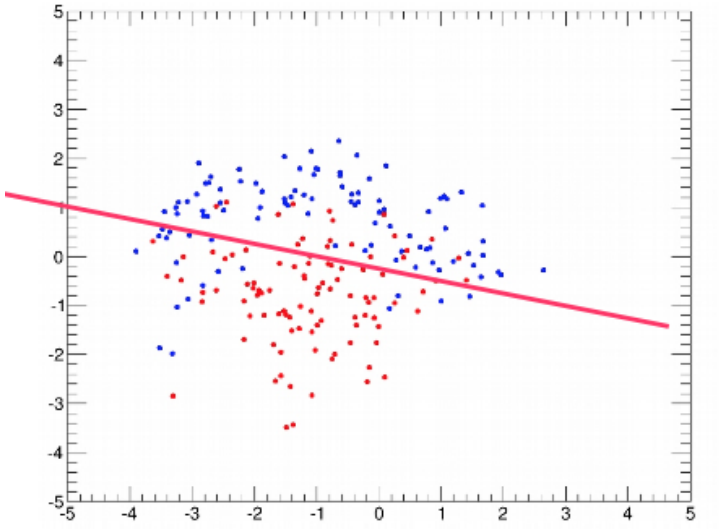
Bestimmung der Fisher Diskriminante

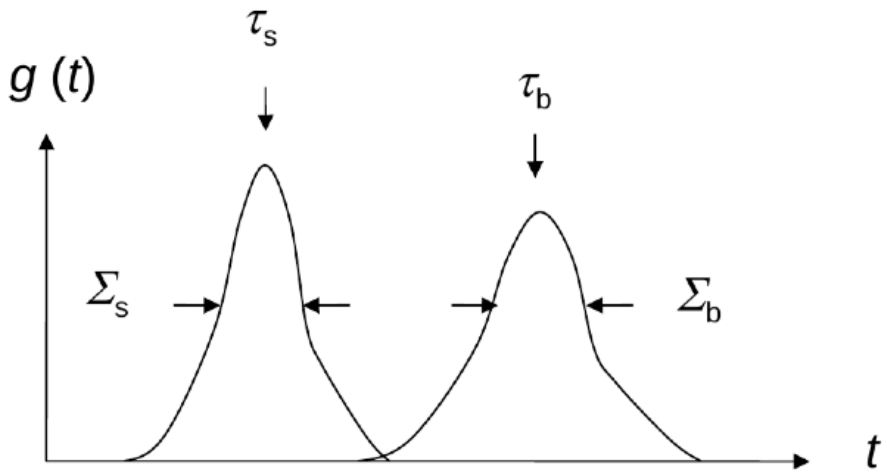
- Maximiere $J(\vec{a}) = \dfrac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$ = $\dfrac{\text{Separation zwischen Kategorien}}{\text{Separation innerhalb Kategorien}}$

- Setzt man $\dfrac{\partial J(\vec{a})}{\partial a_i} = 0$, so erhält man Fishers lineare Diskriminante

$$t(\vec{x}) = \vec{a}^T \vec{x} \quad \text{mit } \vec{a} \propto W^{-1}(\vec{\mu}_s - \vec{\mu}_b)$$

wobei $W = V_s + V_b$



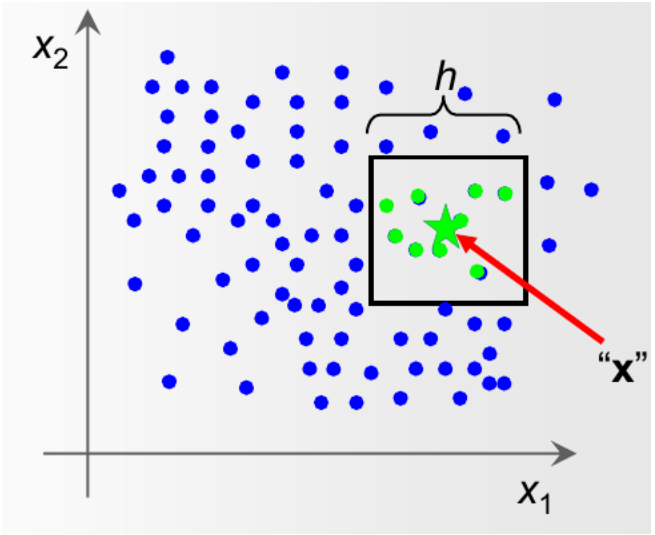


Näherung der Likelihood: k-Nearest Neighbours

- Approximiere Verteilungsdichte im *d*-dimensionalen Raum aus Zahl der Signal- und Untergrundereignisse (Monte Carlo) in einer Umgebung

- PDF(*x*) aus Anteil *k*/*N* von Signal und Untergrundereignissen im Volumen h^d

$$\mathrm{PDF}_{\mathrm{s,b}}(\vec{x}) = \frac{1}{N_{s,b}} \sum_{i=1}^{N} K(\vec{x} - \vec{x}_i)$$



- K ist sog. Kernelfunktion, z.B. Rechteck-Volumen mit euklidischem Abstand

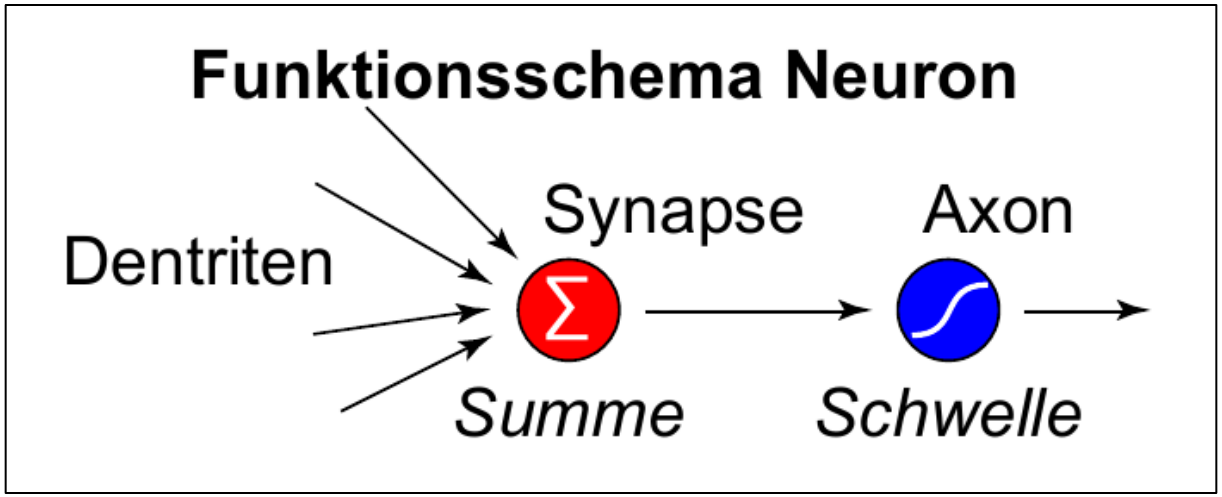
$$K(\vec{x}_i) = \frac{1}{h^d} \prod_{j=1}^{d} \mathrm{R}\left(x_j - (\vec{x}_i)_j - \frac{1}{2}\right) \quad \text{mit } \mathrm{R}(x) = 1 \text{ wenn } x \in [0,1]$$

- Klassifikator ist das (genäherte) Likelihood-Verhältnis $\mathrm{PDF_S}(\vec{x})/\mathrm{PDF_B}(\vec{x})$
  - Kein Trainingsvorgang (→ langsam), auch für nicht-lineare Probleme verwendbar
  - Anwendung erfordert eine große Zahl von Monte-Carlo Ereignissen
  - Verwendung einer Metrik erfordert Normierung der Eingangsvariablen
  - Alternative Metriken: Dreieck, Gauß, Epanechnikov, Mahalanobis-Distanz …

7.5

KÜNSTLICHE NEURONALE NETZE

Künstliche Neuronale Netze



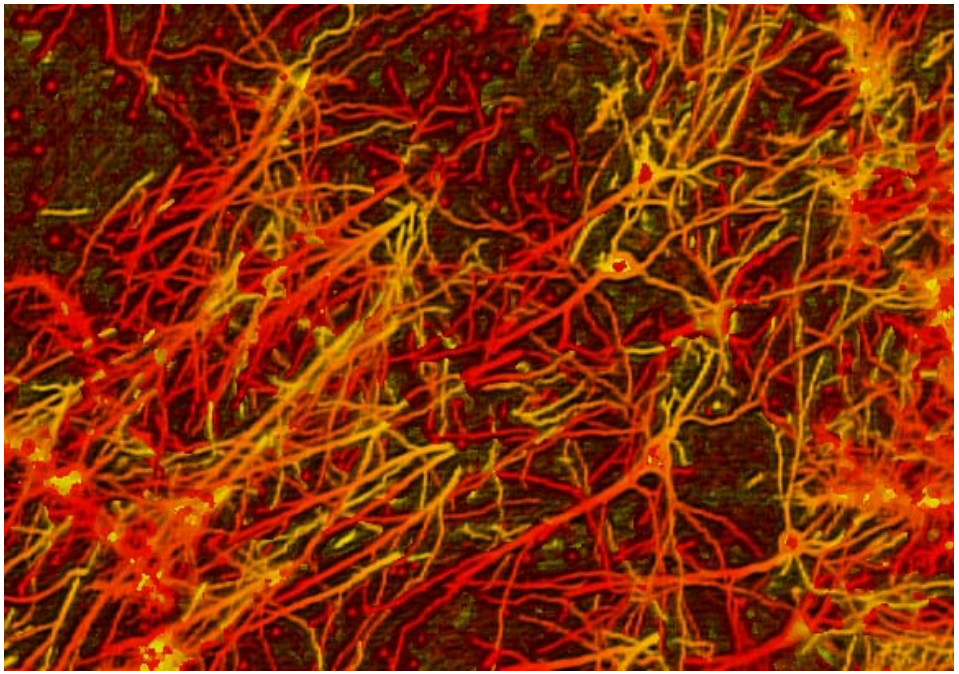
- Diese Vorlesung: Funktionsweise konventioneller Feed-Forward-Netze, die mittels Fehlerrückführung trainiert werden.
- Deep Learning → nächste Vorlesung
- Komplexe, selbstorganisierende Netzwerke nicht behandelt.

Biologische Neuronale Netze

- Menschliches Gehirn:
  - Viele Prozessoren = Neuronen: $O(10^{11})$
  - Einzelner Prozess-Schritt langsam: O(10 ms) ~ 100 Hz
  - Massiv parallel: $O(10^{14})$ Synapsen

- Lernen durch Selbstorganisation - Details menschlicher Lernprozesse sind wenig verstanden

- Tolerant gegenüber unvollständiger oder verrauschter Information

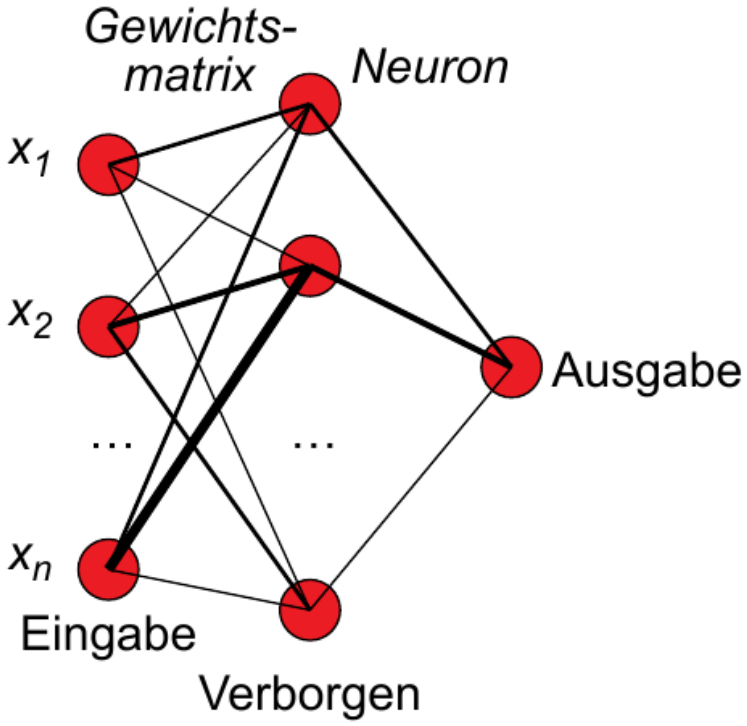


Quelle: www.willamette.edu/~gorr/classes/cs449/brain.html

Künstliche Neuronale Netze (ANN)

- Artificial Neural Network (ANN)
  - Gängige Methode des überwachten Lernens
  - Inspiriert von biologischen neuronalen Netzen
  - Nicht-lineares Verfahren

- Feed-Forward Netz im Computer
  - Typisch: O(10^3) Neuronen (bei Deep Learning bis zu 10^7)
  - Einfache Topologie
  - Schnell (O(ns)) ~ GHz
  - Langsam beim Lernen (Training)



Künstliche Neuronale Netze

- Bisher behandelte Methoden konnten durch analytische Berechnung der Parameter angewandt werden.

- Suche jetzt: allgemeine skalare Funktion *t*(*x*) des *n*-dimensionalen Merkmalvektors *x*, so dass eine Kosten- oder Fehlerfunktion (engl. Loss Function) $\mathbf{Er}(t_{\mathrm{true}}, t(\vec{x}))$ minimal wird

- Allgemeiner Ansatz: $t(\vec{x}) = \sum_i w_i h_i(\vec{x})$ mit i=1,*n* beliebigen Funktionen $h_i(x)$

- Gewichte w_i werden durch Minimierung der Fehlerfunktion bestimmt

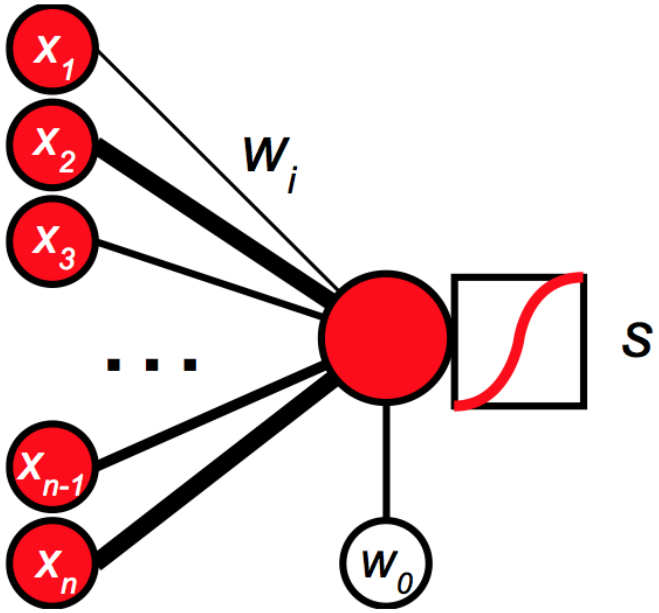
- Bemerkung: Ansatz ähnlich wie LDA (Fisher-Diskriminante), aber:
  - $h_i(x)$ i.a. nicht-linear
  - Keine Obergrenze für die Anzahl der Funktionen

Einlagiges Perzeptron

Rosenblatt, 1958

- One-Layer Perceptron (keine verborgenen Lagen)
- Prüfgröße $t(\vec{x})$: Gewichtete Summe der n Eingabewerte x_i
- Aktivierungsfunktion $A \rightarrow$ Schwelle

$$t(\vec{x}) = A\left(w_0 + \sum_{i=1}^{n} w_i\, x_i\right)$$



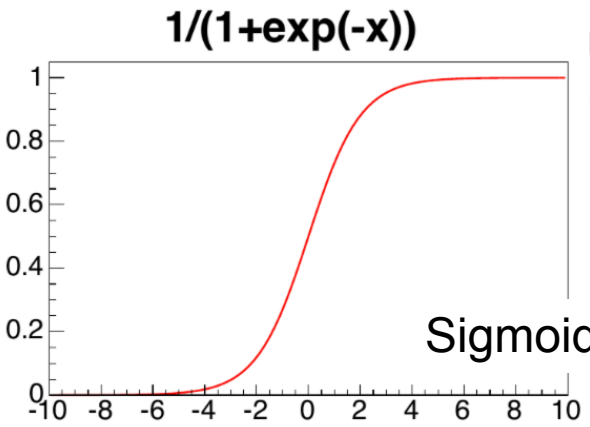
- Zusätzlich oft: Bias $w_0 \rightarrow$ Verschiebung
- Supervised Learning: Bestimmung der Gewichte w_i für optimale Trennung
- A monoton: Einlagiges Perzeption nur für linear separierbare Probleme

Aktivierungsfunktion

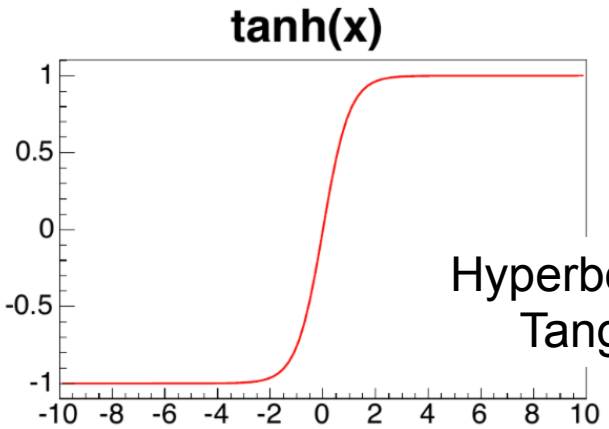
- Monotone Abbildung
- Bedingung zur Optimierung (Training)
  A sollte differenzierbar sein

- Konventionelle ANN meist: R → [-1,1]
  - Logistische Funktion
    = Sigmoidfunktion
    = Fermi-Funktion
  - Hyperbolischer Tangens

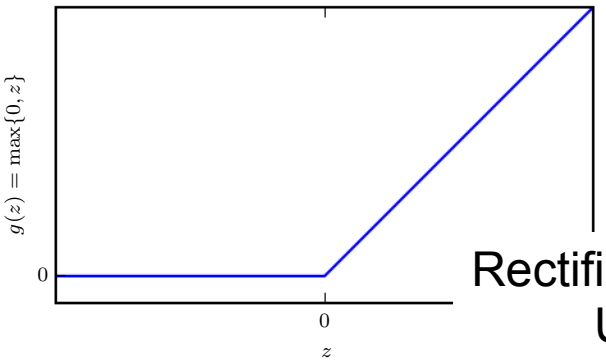
- Heute auch häufig (e.g. DL):
  - Rectified Linear Unit (ReLU)
    g(z) = max(0,z)



Sigmoid-Funktion



Hyperbolischer Tangens

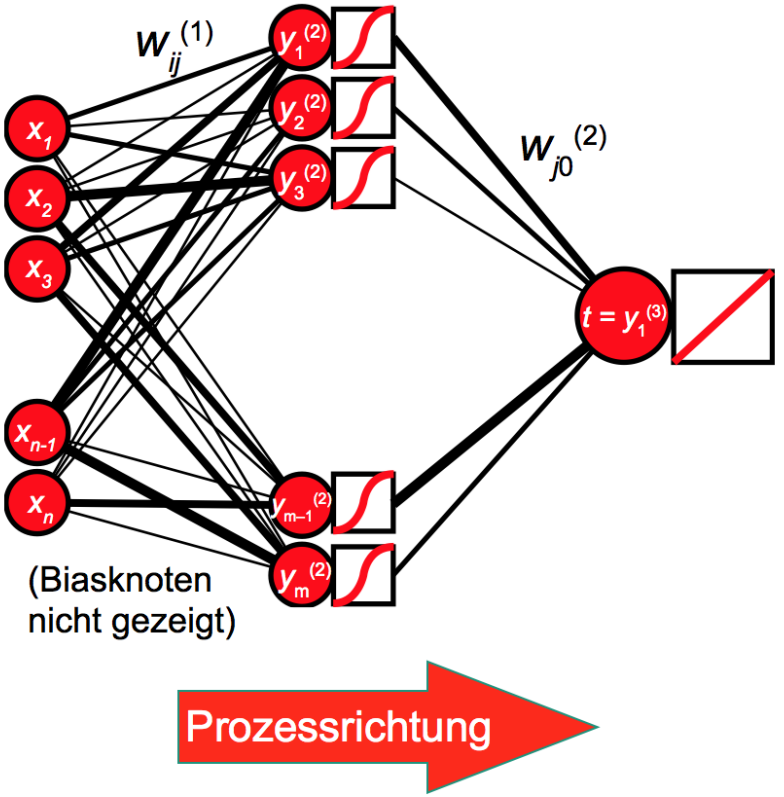


Rectified Linear Unit

Mehrlagiges Perzeptron

- Multi-Layer Perceptron (MLP)
- Verallgemeinerung: Eine (oder mehrere) verborgene Lagen

- Feed-forward Network
  - Jede Lage wird nur aus vorheriger Lage gespeist
  - Wichtigster Fall: eine einzelne verdeckte Lage (engl: „hidden layer“) mit *m* Knoten (oft: *m*>*n*)

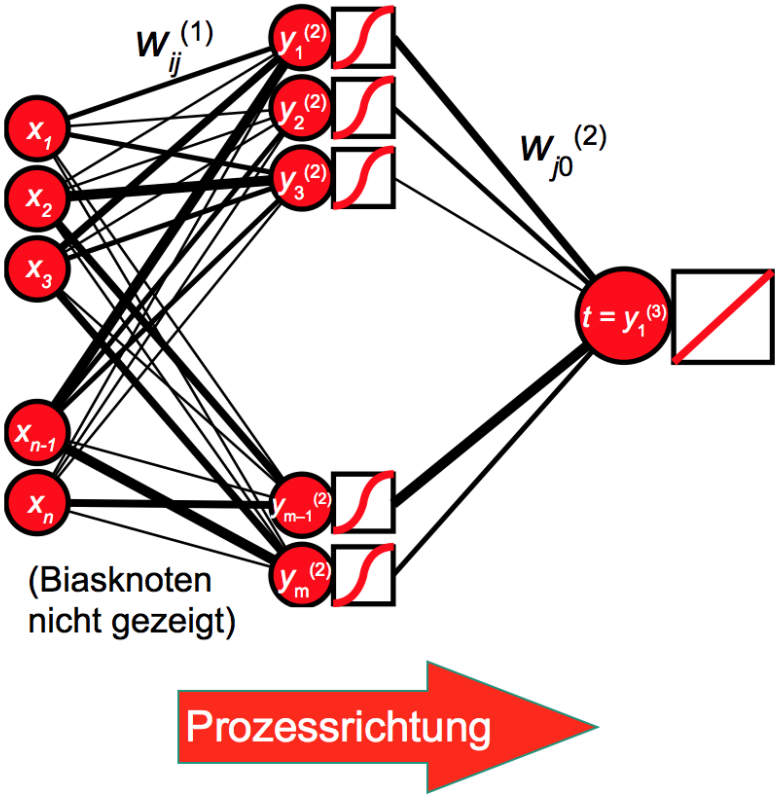


- Nicht-lineare Prüfgröße:

$$t(\vec{x}) = A^{(2)}\left(\sum_j^m w_j^{(2)} \cdot A^{(1)}\left(\sum_{i=0}^n w_{ij}^{(1)}\, x_j\right)\right)$$

Mehrlagiges Perzeptron

- Netzwerk mit *n* Eingabeknoten, *m*>*n* verborgenen Knoten und einem Ausgabeknoten → Gewichtsmatrix *W* mit
  - *m*+1 Gewichten $w_i^{(2)}$
  - *m**(*n*+1) Gewichten $w_{ij}^{(1)}$

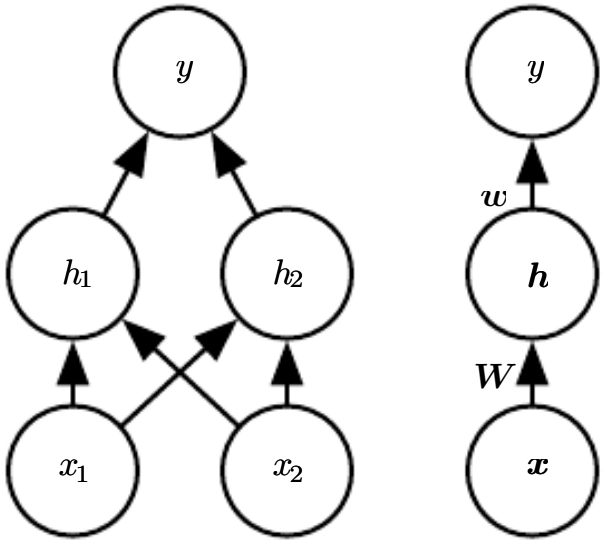


- Mit geeigneten Aktivierungs-funktionen kann mehrlagiges Perzeptron beliebige kontinuierliche Funktionen auf Unterräumen von R^n annähern.

- Training: Bestimmung der Gewichte durch Anpassung → Minimierung der Fehler- oder Kostenfunktion
- Bemerkung: sehr viele Parameter, Vorverarbeitung des Merkmalvektors und genaue Überwachung des Trainingsprozesses

Beispiel: XOR Funktion

- Netz mit einer einzelnen verborgenen Lage: $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$



$$y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$$

$$h_i = g(\boldsymbol{x}^\top \boldsymbol{W}_{:,i} + c_i)$$

- x,y: Ein- und Ausgabewerte
- w,W: Gewichte
- g: Aktivierungsfunktion (ReLU): g(z)=max(0,z)
- b,c: Bias

- Insgesamt: $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$

http://www.deeplearningbook.org/ Seite 171 ff

Beispiel: XOR Funktion

- Feedforward Netz: $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$
- Lösung:

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \qquad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \qquad b = 0.$$

- Beweis:

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} \qquad + \mathrm{c} \rightarrow \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$\mathrm{ReLU} \rightarrow \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \qquad \times \omega^\mathsf{T} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

http://www.deeplearningbook.org/ Seite 171 ff

Training: Minimierung der Fehlerfunktion

- Suche Gewichtsmatrix *W*, so dass Fehlerfunktion minimal:
  - Gängigste Methode: Fehler-Rückführung („error back-propagation“)
  - Iteratives Verfahren zur Bestimmung der optimalen Gewichte W
  - Startpunkt: wähle zufällige oder geschätzte Startwerte für W

- Fehlerfunktion (auch Kosten- oder Verlustfunktion, engl. Loss Function): Maß für Übereinstimmung des Klassifikators mit Erwartung

$$\mathbf{Er}(t_{\text{true}}, t(\vec{x}))$$

- Häufig verwendet: linearer („mean average distance“ (MAD)) oder quadratischer Abstand („mean squared error“ (MSE)) gemittelt über Ereignisse des Trainings-Datensatzes.

$$\text{MSE: } \mathbf{Er}(\vec{x}|W) = \sum_{a=1}^{N} \mathbf{Er}(\vec{x}_a|W) = \frac{1}{2}\sum_{a=1}^{N}(t_{\text{true}} - t(\vec{x}_a|W))^2$$

Gradientenabstiegsverfahren

- Verwende Ereignisse mit bekannter Klassifikation (d.h. t_{true} = 0 oder 1)
- Iterative Minimierung der Fehlerfunktion und Aktualisierung der Gewichte
- Gradientenabstiegsverfahren: möglich, wenn Aktivierungsfunktion differenzierbar. Beispiel hier: sigmoide Funktion

$$A(x) = \frac{1}{1+\exp(-x)} \rightarrow \frac{dA(x)}{dx} = \frac{\exp(-x)}{(1+\exp(-x))^2} = A(x)(1-A(x))$$

- In jeder Iteration („Lernzyklus“) werden die Gewichte W in Richtung des steilsten Abstiegs (Gradient) der Fehlerfunktion korrigiert („Fehler-Rückführung“)

$$W^{(n+1)} = W^{(n)} - \eta \nabla_W \mathbf{Er}(t_{\text{true}}, t(\vec{x})|W)$$

- η ist dabei die sog. „Lernrate“: Schrittweite in Richtung Gradient

Fehlerrückführung

- Ausgangspunkt: *n* Input-Variablen, *m* verborgene Knoten, *N* Ereignisse

$$t(\vec{x}) = A^{(2)} \sum_j^m w_j^{(2)} y_j^{(2)}(\vec{x}) \quad \text{mit} \quad y_j^{(2)}(\vec{x}) = A^{(1)} \sum_i^n w_{ij}^{(1)} x_i$$

- Fehlerfunktion (MSE):

$$\mathbf{Er_a} = \frac{1}{2}(t_{\text{true}} - t(\vec{x}_a))^2$$

- Änderung der Gewichte der verborgenen Lage (2):

$$\Delta w_j^{(2)} = -\eta \sum_{a=1}^N \frac{\partial E_a}{\partial w_j^{(2)}} = -\eta \sum_{a=1}^N \frac{\partial E_a}{\partial t(\vec{x}_a)} \frac{\partial t(\vec{x}_a)}{\partial w_j^{(2)}} = -\eta \sum_{a=1}^N (t_{true} - t(\vec{x}_a)) y_j^{(2)}(\vec{x})$$

- Änderung der Gewichte der Eingabe-Lage (1):

$$\Delta w_{ij}^{(1)} = -\eta \sum_{a=1}^N \frac{\partial E_a}{\partial y_j^{(2)}} \frac{\partial y_j^{(2)}}{\partial t(\vec{x}_a)} \frac{\partial t(\vec{x}_a)}{\partial w_{ij}^{(1)}} = -\eta \sum_{a=1}^N (t_{true} - t(\vec{x}_a)) \cdot y_j^{(2)}(\vec{x}_a)(1 - y_j^{(2)}) \cdot x_{i,a}$$

Lernzyklen

- Richtige Wahl der Lernrate η (Schrittweite) ist wichtig
  - Zu groß: Algorithmus oszilliert um Minimum
  - Zu klein: Langsame Konvergenz
- Optimales η bestimmbar aus negativ Inversem der Hesse-Matrix

$$\eta = -\left(\frac{\partial^2 \mathbf{Er}}{\partial W_i \partial W_j}\right)^{-1}$$

  - Ausrechnen durch Taylor-Entwicklung von Er(W) um Minimum Er(W_0) und Vergleich mit

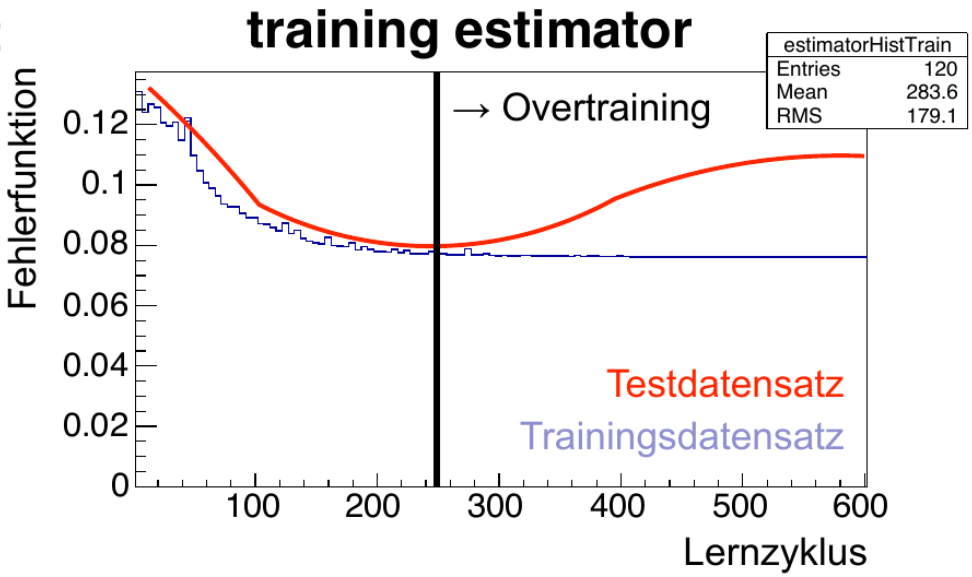
$$\Delta W = W^{(n+1)} - W^{(n)} = -\eta \nabla_W Er$$

- Es gibt im hochdimensionalen Parameter-Raum viele Gebiete mit kleinem Gradienten (vor allem Sattelpunkte)
  - Langsames Anschalten der Schwellenfunktion (Temperatur)
  - Regularisierung → Penalty in der Verlustfunktion gegen hohe Gewichte:

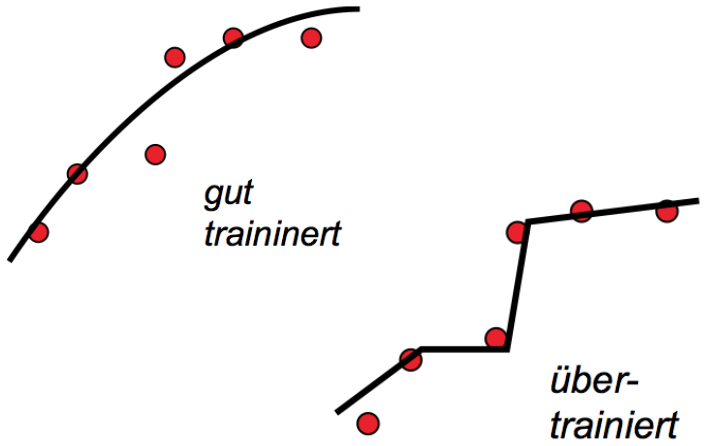
$$M = \mathrm{Er} + \beta \sum_{ij} w_{ij}^2$$

ANN - Lernkontrolle

- Berechne nach jedem Lernzyklus:
  - Fehlerfunktion: Er($x|W$)
  - Effizienz und Reinheit (ROC)



- Generalisierungsfehler "Über-Training":
  - ANN folgt spezifischen Eigenschaften des Trainings-Datensatzes (z.B. statistische Fluktuationen)
  - Kontrolle: Aufteilung der Daten in Training- und Test-Datensatz
  - Auswertung von Fehlerfunktion und Effizienz des Test-Datensatzes



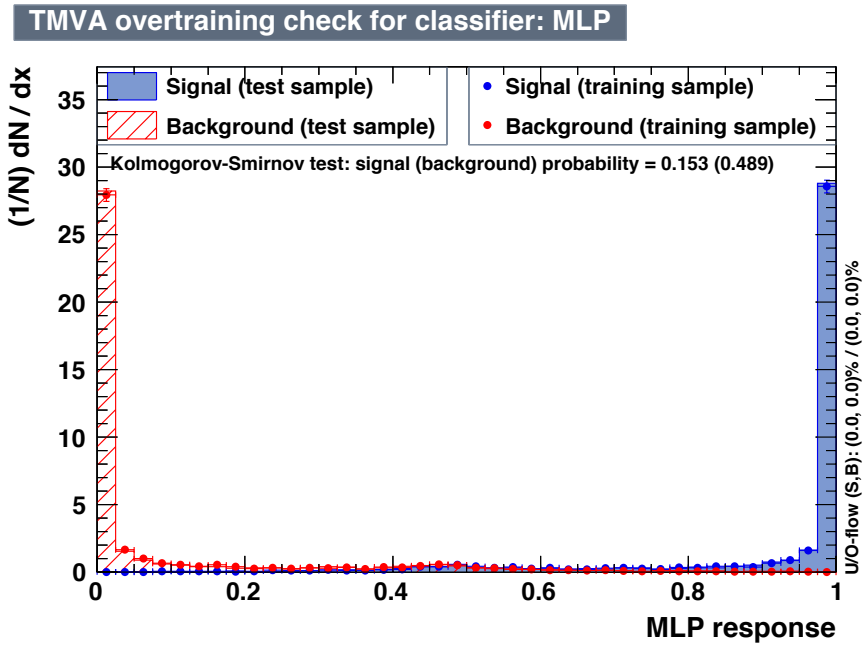
ANN Training

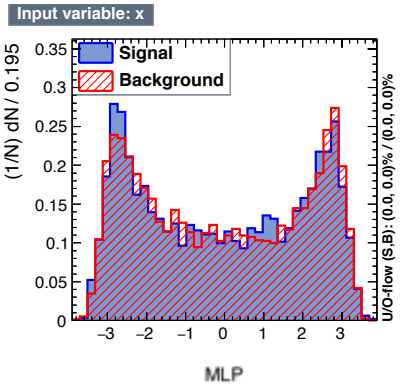
	Batch-Lernen (auch: Bulk-Lernen)	Stochastisches Lernen (auch: Online-Lernen)
Verfahren	Gewichtsänderung nach vollständiger Iteration über alle Trainingsereignisse → globaler Gradient	laufende Anpassung der Gewichte nach einem oder wenigen (<200) Ereignissen → lokaler Gradient
Geschwindigkeit	langsam bei großen Trainingsdatensätzen	schneller als Batch-Lernen (besonders bei Training mit teilweise redundanten Ereignissen)
Bemerkung	günstig für theoretische Betrachtungen zu Konvergenz	in Teilchenphysik (fast) immer bessere Wahl (z. B. verwendet in TMVA)

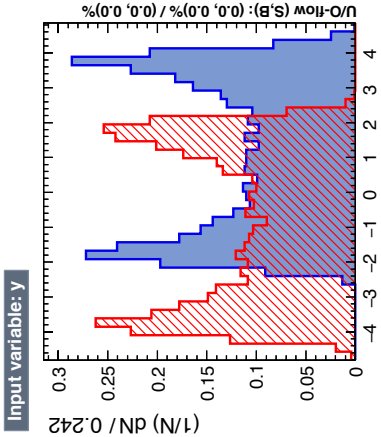
- ANN profitieren stark von Präprozessierung der Merkmale (Normierung und Dekorrelation)

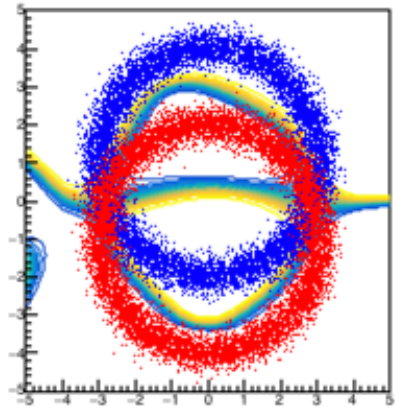
Neuronales Netz in TMVA

- MLP mit zwei Eingangsvariablen, 8 verborgenen Knoten, sigmoide Aktivierungsfunktion, 600 Lernzyklen (“Epochen”), 10000 Ereignisse
- Training: 23 Sekunden, Ausführung: 0.04 Sekunden
- Sehr gute Trennung





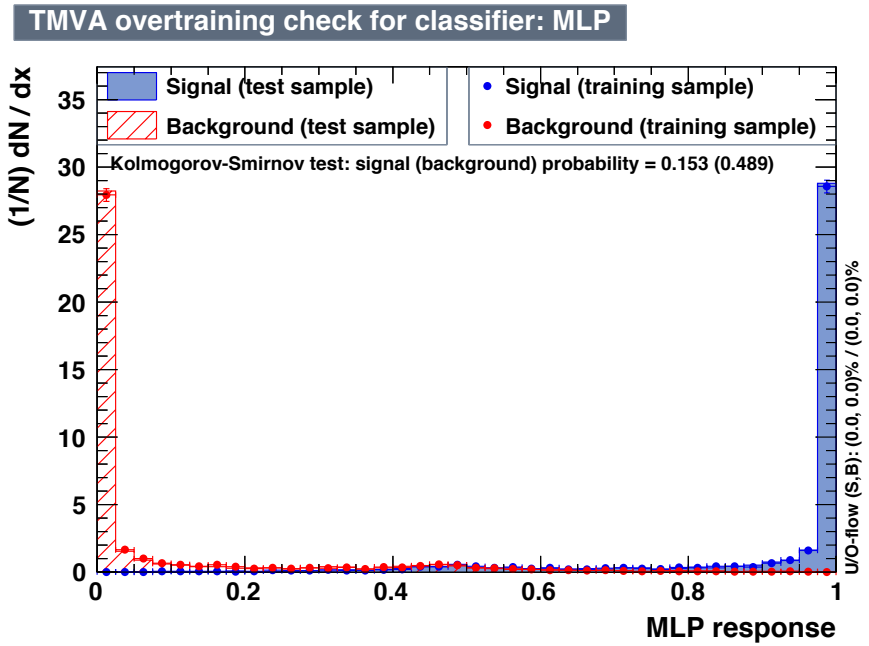


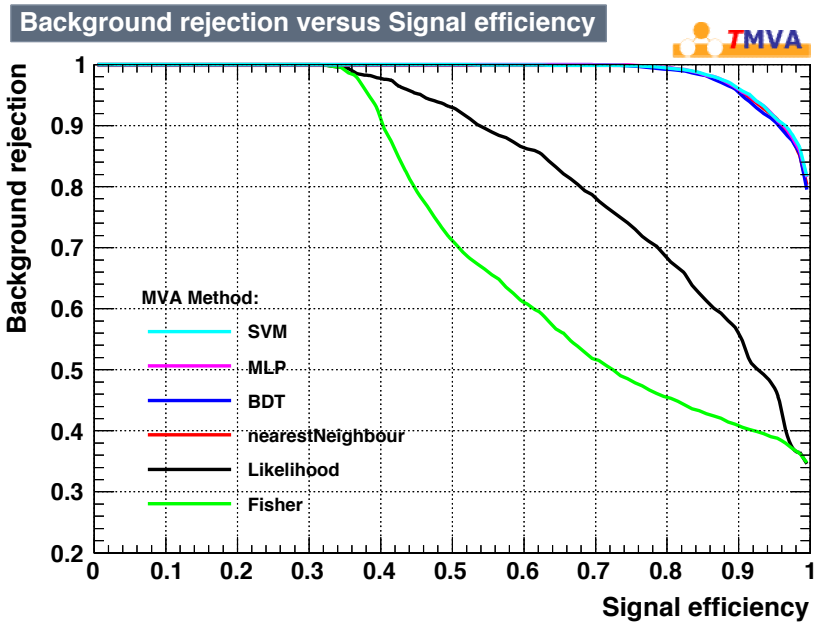


Beispiel: “python TrainEvaluateResponse_v11.py Circle”

Neuronales Netz in TMVA

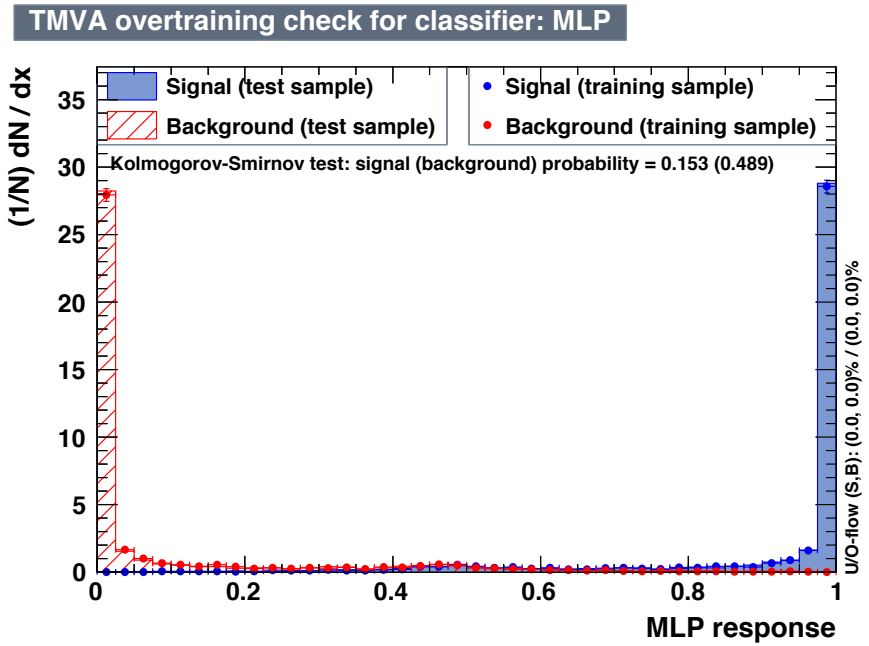
- MLP mit zwei Eingangsvariablen, 8 verborgenen Knoten, sigmoide Aktivierungsfunktion, 600 Lernzyklen (“Epochen”), 10000 Ereignisse
- Training: 23 Sekunden, Ausführung: 0.04 Sekunden
- Sehr gute Trennung

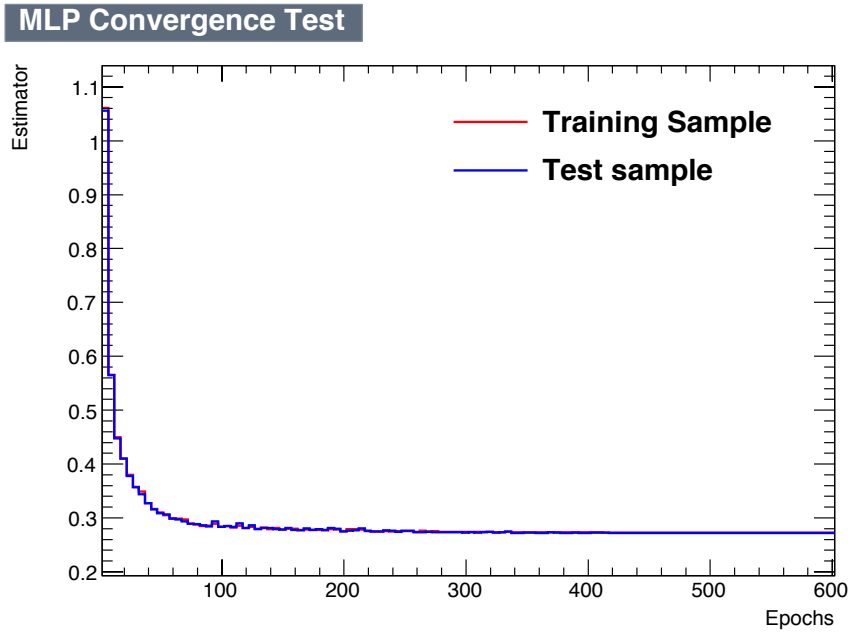




Neuronales Netz in TMVA

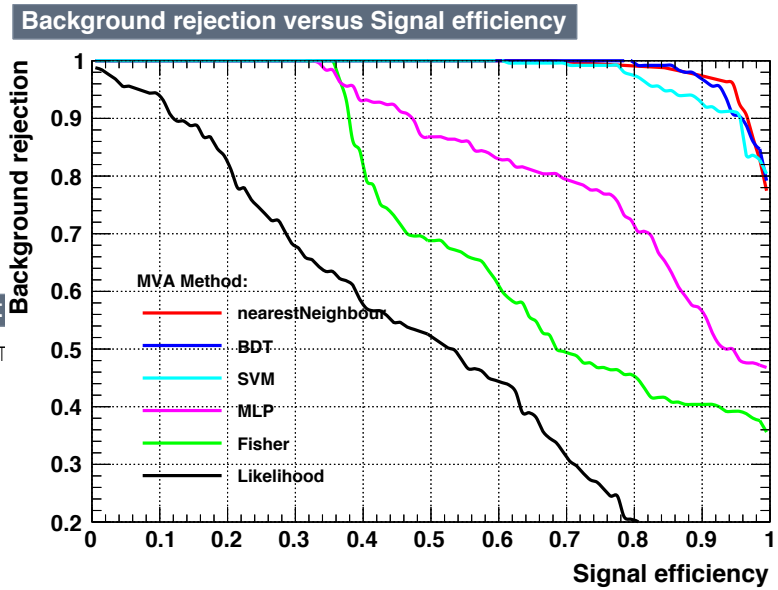
- MLP mit zwei Eingangsvariablen, 8 verborgenen Knoten, sigmoide Aktivierungsfunktion, 600 Lernzyklen (“Epochen”), 10000 Ereignisse
- Training: 23 Sekunden, Ausführung: 0.04 Sekunden
- Sehr gute Trennung
- Kein Anzeichen von Übertraining

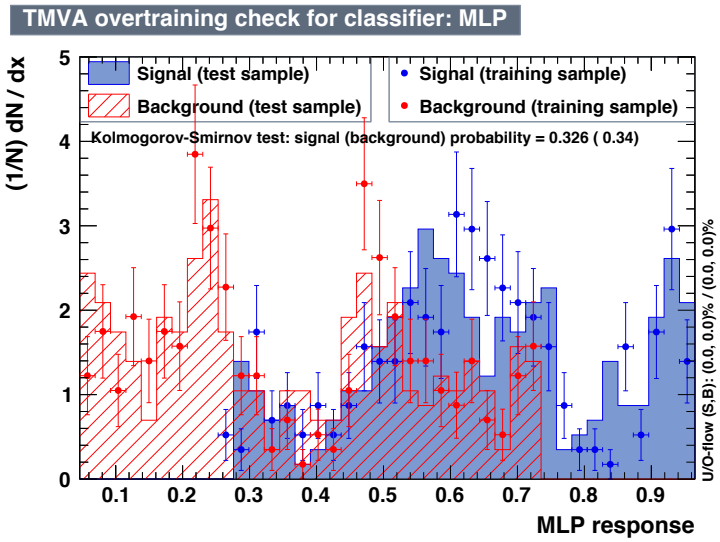


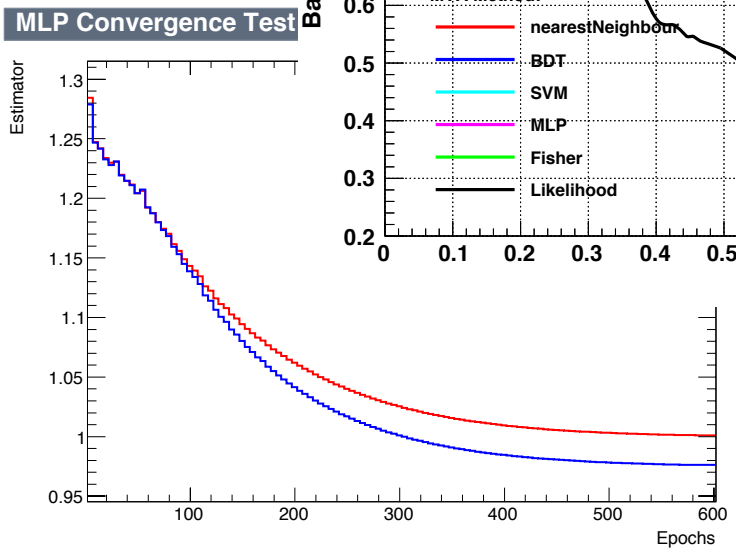


Neuronales Netz in TMVA

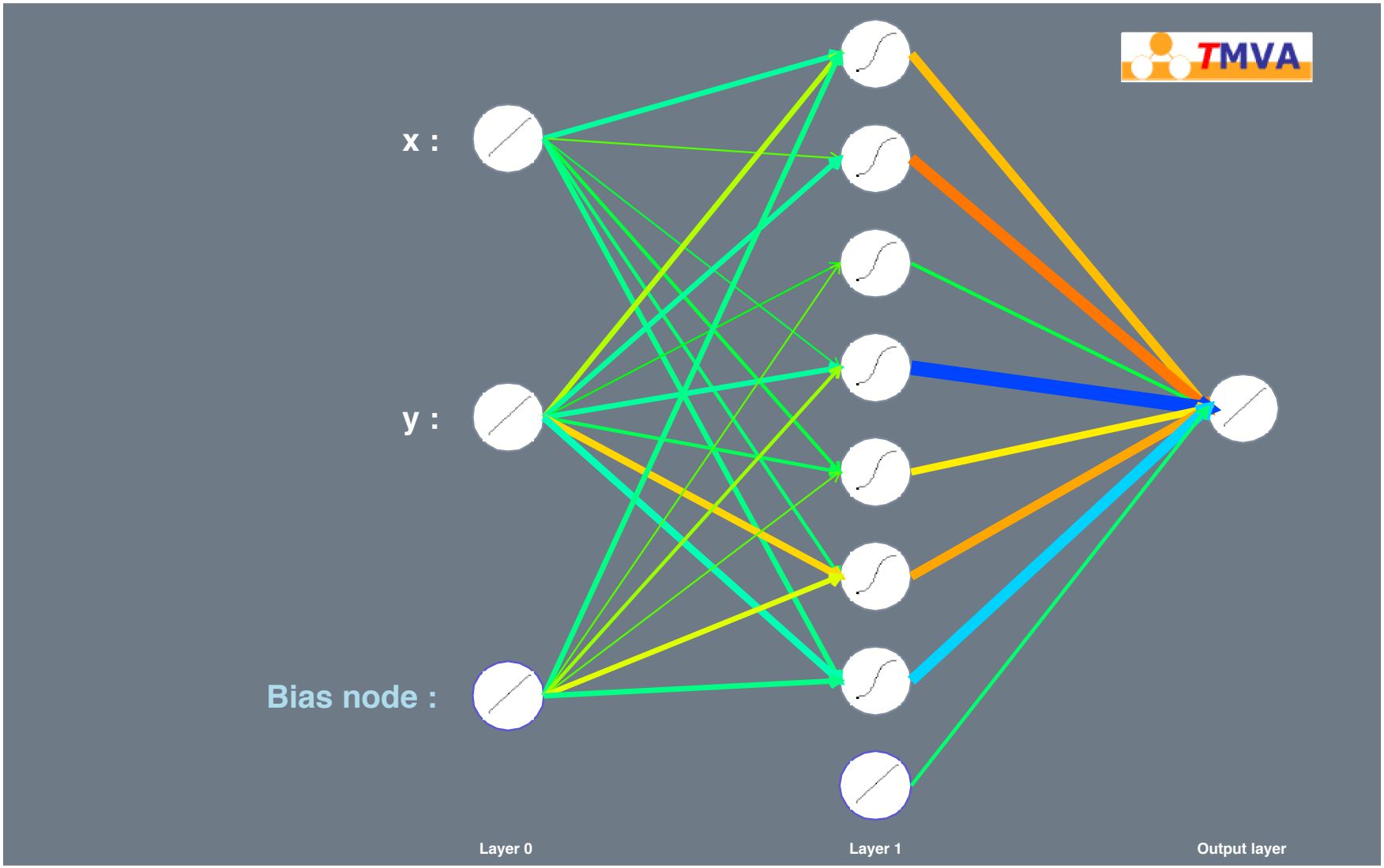
- MLP mit zwei Eingangsvariablen, 8 verborgenen Knoten, sigmoide Aktivierungsfunktion, 600 Lernzyklen (“Epochen”), 500 Ereignisse
- Training: 1.5 Sekunden, Ausführung: 0.04 Sekunden
- Schlechte Trennung, Übertraining
- Bessere Ergebnisse: BDT, SVM, kNN







Neuronales Netz in TMVA



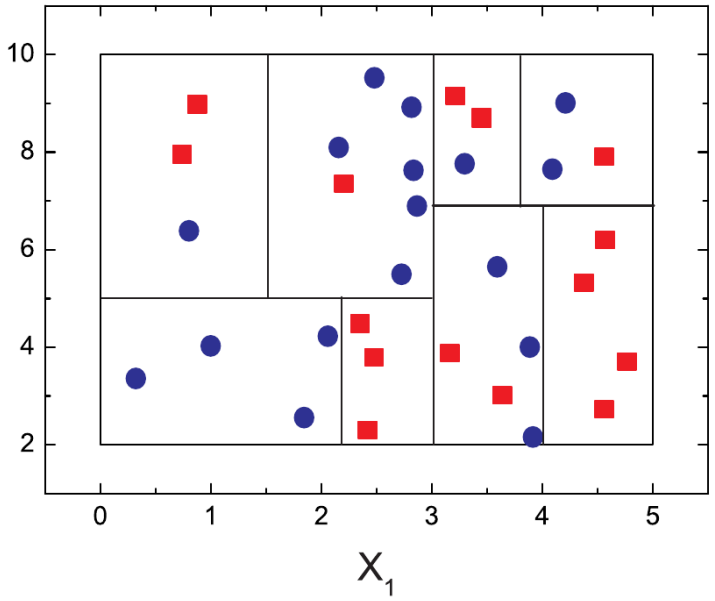
7.6

VERSTÄRKTE ENTSCHEIDUNGSBÄUME

Entscheidungsbäume

- Eingangsdaten: Merkmalvektor x
- Ausgabe: Klassifikation S und B
- Sequentielle Schnitte („<“ oder „>“) in einzelnen Variablen:

  Merkmalraum zerlegt in (Hyper-)Rechtecke: Konstruktion festgelegt durch Sequenz von Entscheidungen



Quelle: Bohm/Zech

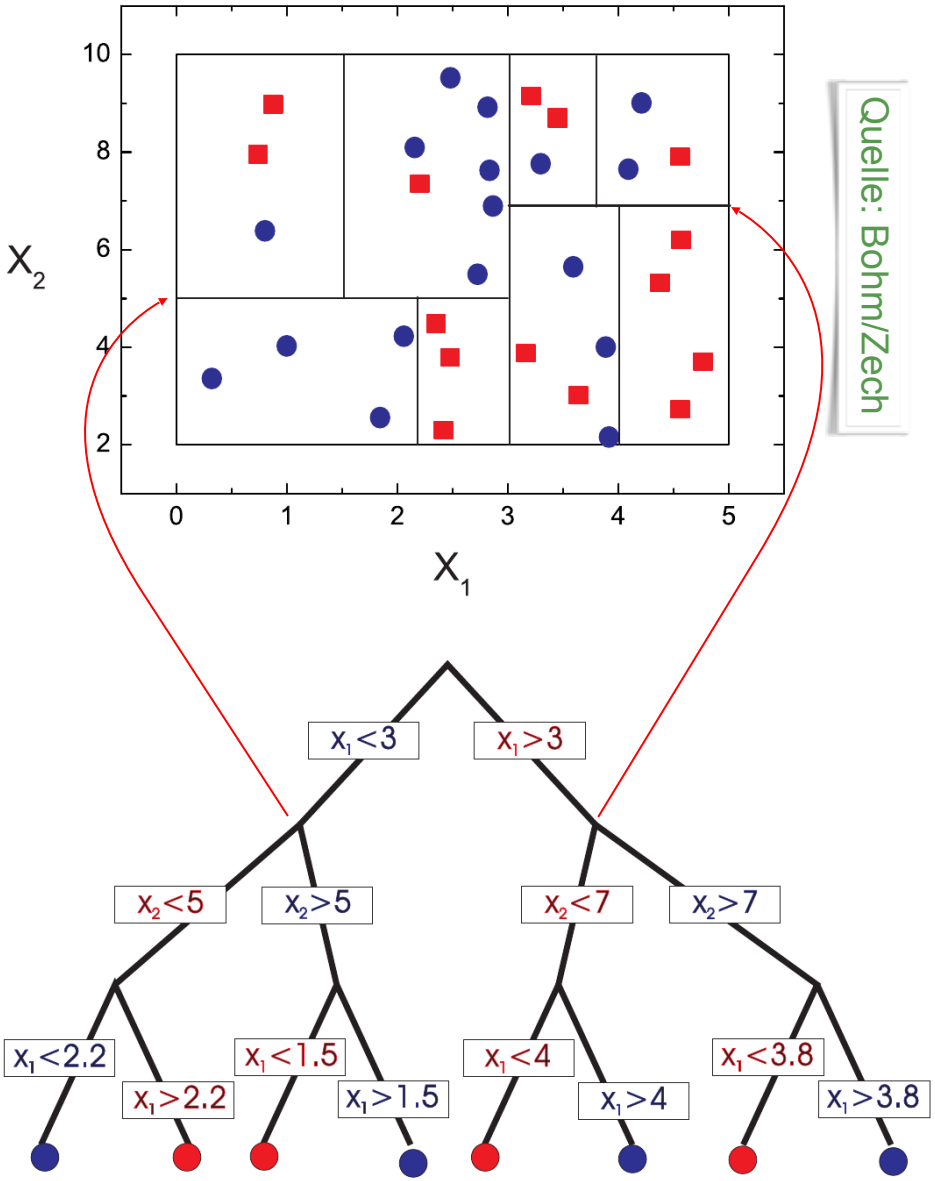
Entscheidungsbäume

- Eingangsdaten: Merkmalvektor $\vec{x}$
- Ausgabe: Klassifikation S und B
- Sequentielle Schnitte („<“ oder „>“) in einzelnen Variablen:

  Merkmalraum zerlegt in (Hyper-)Rechtecke: Konstruktion festgelegt durch Sequenz von Entscheidungen

- Entscheidungsbaum:
  - Einzelne Entscheidungen bilden Äste, die Endknoten heissen Blätter

- Abbruchregel beendet den Prozess



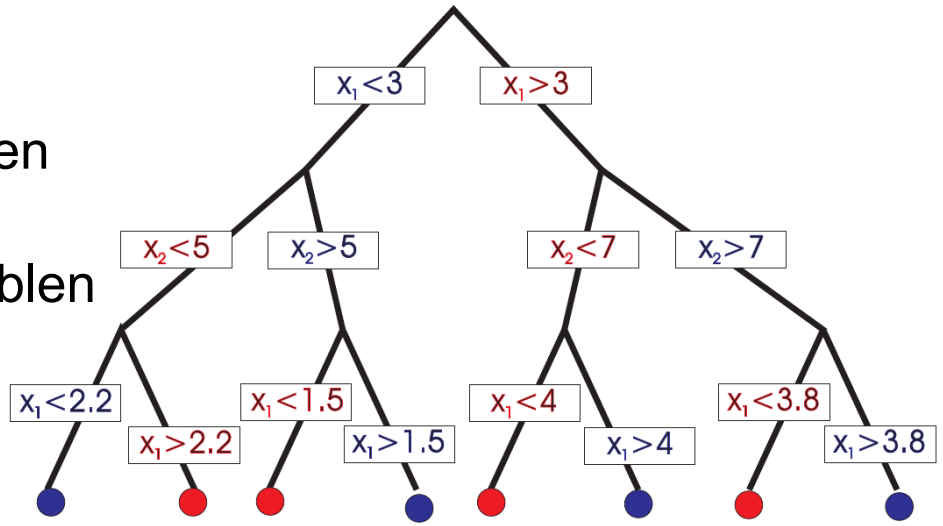
Quelle: Bohm/Zech

Aufwachsen des Entscheidungsbaums

- Suche signifikantestes Merkmal x_i des Trainingsdatensatzes
- Teile Ereignisse in zwei Äste (an einer “Entscheidungsgrenze”)
- Weitere Aufteilung der resultierenden Untermengen: verwende diejenige Variable zur Trennung, die die beste weitere Separation anhand des gleichen Teilungskriteriums liefert („greedy algorithm“).

- Fortsetzung bis Abbruchkriterium

- Bemerkungen:
  - Merkmale können mehrfach oder gar nicht verwendet werden
  - Trennung ist unabhängig von Skala → Normierung der Variablen und Behandlung von Outliern nicht notwendig



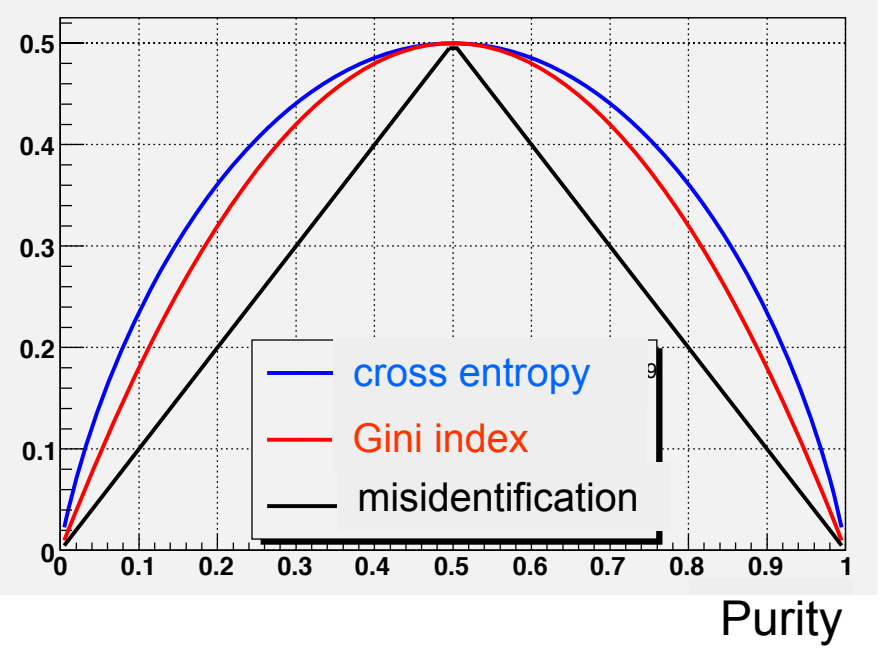
Festlegen der Entscheidungsgrenze

- Teilung der Elternzelle *E* in zwei Zellen *s* und *b.*
- Übliche Kriterien:
  - Fehlidentifikation: $F = 1 - \max(p, 1-p)$
  - Gini-Index: $G = 2p(1-p)$
  - Cross Entropy $S = -(p \ln(p) + (1-p)\ln(1-p))$

Reinheit:

$$p = \frac{n_s}{n_s + n_b}$$

- Bester Schnitt, wenn Gewinn durch Teilung maximal für beide Samples, Signal und Untergrund
- Reinheitszuwachs:
  $D = n_E G_E - n_s G_s - n_b G_b$
- Abbruch, wenn $D < D_{crit}$



H. Voss, Root-Workshop 2013

Abbruchkriterien

- Ziel: Äste mit möglichst hoher Reinheit
- Bäume mit wenigen Ästen sind robuster (da Statistik in allen Blättern groß). Beste Resultate für kleine Entscheidungsbäume (< 10 Blätter)

- Typische Abbruchkriterien:
  - Reinheitszuwachs ist unterhalb einer Schwelle, z.B. $D_{crit} > D = G_E - G_s - G_b$ (siehe vorige Seite)
  - Maximale Tiefe, maximale Anzahl von Blättern, minimale Anzahl der Ereignisse u.ä.
- Wahl der Tiefe ist i.a. wichtiger als exakte Wahl des Reinheitskriteriums

- Pruning: Nachträgliches Entfernen der Äste mit geringer Trennschärfe mithilfe einer Verlustfunktion (auf einem Test-Datensatz).
  TMVA Manual: Pruning nicht empfohlen, da niedriger liegende Äste ggf. bessere Trennung liefern könnten.

Wald aus Entscheidungsbäumen

- Generalisierungsfähigkeit eines einzelnen Baumes ist i.a. gering: „weak learner“

- Ein ganzer Wald (≥1000 Bäume) ist aber sehr mächtig
  - Robuste und scharfe Lösung durch Mehrheitsentscheidung vieler Bäume mit jeweils geringer Trennschärfe. Ensemble-Methode

- Verbesserung der Klassifizierung:
  - Random Forest: Jeder Baum wird anhand von zufällig ausgewählten Untermengen von Variablen entwickelt.
  - Stochastische Teilmengen („Bagging“): Teilmengen des Test-Datensatzes werden verwendet, um einzelne Entscheidungsbäume zu generieren.
  - Verstärkung der Merkmalvektoren („Boosting“): Fehlklassifizierte Ereignisse werden mit Gewichten versehen und in den so trainierten Bäumen stärker berücksichtigt

AdaBoost

Freund, Schapire 1997

- Adaptive Verstärkung (engl. Adaptive Boosting, AdaBoost)
- Berücksichtigung falscher Zuordnungen mit höherem Gewicht beim Aufwachsen des nächsten „weak learner“ Baums.
- Effekt: Höhere Wahrscheinlichkeit der korrekten Zuordnung, da Variablen mit relevanter Information stärker berücksichtigt werden.
- Gewicht α_i für fehlklassifizierte Ereignisse, berechnet aus Fehlklassifikationsrate err_{i-1} des vorigen Baums

$$\alpha = \frac{1 - \mathrm{err}}{\mathrm{err}}$$

- Boosted Klassifikation:

$$y_{\mathrm{Boost}}(\mathbf{x}) = \frac{1}{N_{\mathrm{collection}}} \cdot \sum_{i}^{N_{\mathrm{collection}}} \ln(\alpha_i) \cdot h_i(\mathbf{x})$$

mit $h_i(x)$: 1(-1) für Signal(Untergrund) und $N_{collection}$: Anzahl der Bäume

- Abschwächung des Boosts: Langsameres Lernen: $\alpha \rightarrow \alpha^{\beta}$

AdaBoost und GradientBoost

- Sequenz von “weak learner”-Klassifikatoren, die gemeinsam (im Ensemble) eine gute Trennung liefern.
- „Stochastic Gradient Boosting“: Zusätzliche Randomisierung durch „bagging“ Technik: beschleunigt Lernzyklen bei gleichzeitiger Abschwächung des einzelnen „weak learners“

- Man kann zeigen, dass Boosting in AdaBoost äquivalent ist zur Minimierung einer Verlustfunktion:

$$L(F, y) = \exp(-F(x) \cdot y)$$

  wobei *F*(*x*) das Zwischenergebnis der Klassifikation; und *y*: wahrer Wert (0,1)
- GradientBoost verwendet:

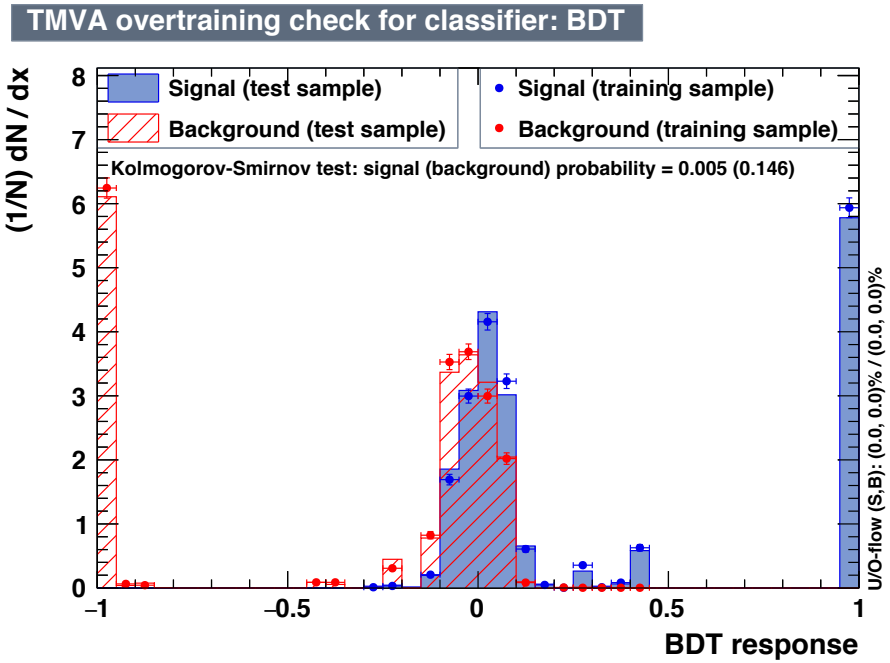
$$L(F, y) = \ln(1 + \exp(-2F(x) \cdot y))$$

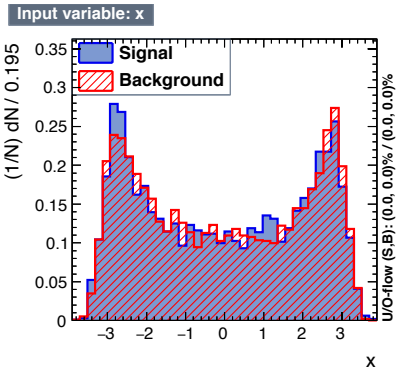
  - Vorteil: robusteres Verhalten bei Outliers und Rauschen
  - Nachteil: keine einfache analytische Lösung
- Bestimme Gradient der Verlustfunktion und bestimme neue Gewichte durch Regression.

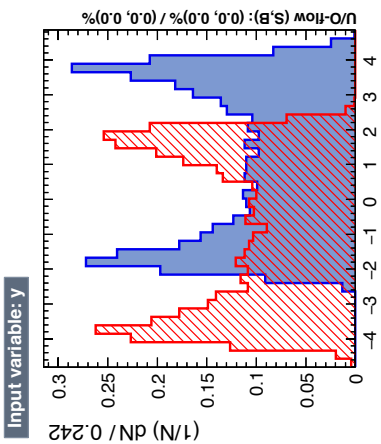
Boosted Decision Tree in TMVA

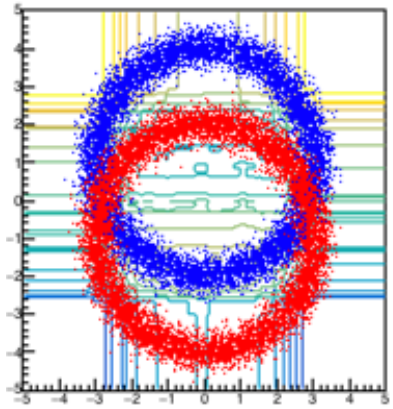
TMVA default: 5%

- AdaBoost, NTrees=500, MinNodeSize=0.05, AdaBoostBeta=0.5, MaxDepth=3, nCuts = 20, SeparationType=GiniIndex, 10000 Ereignisse
- Training: 2 Sekunden, Testing: 0.5 Sekunden
- Schlechte Trennung und p(KS)=0







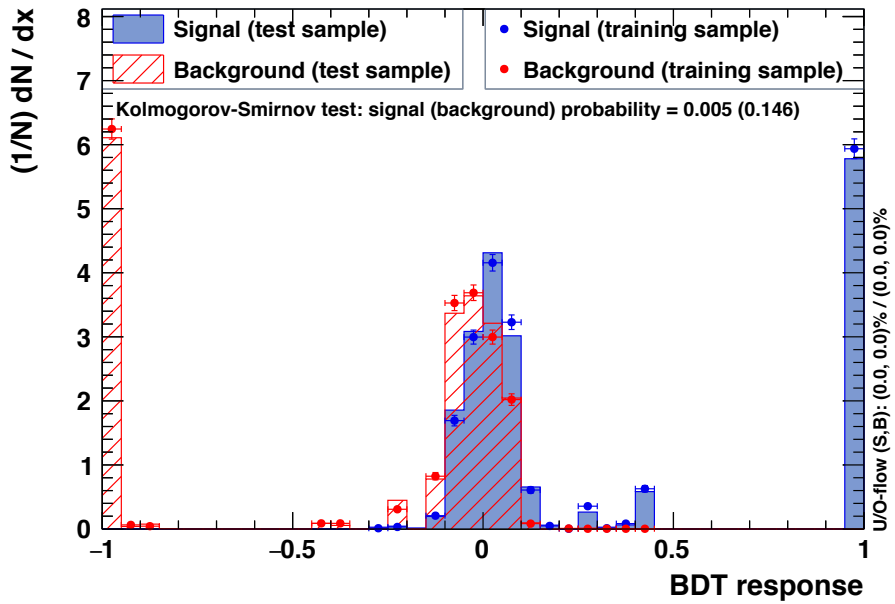


Boosted Decision Tree in TMVA

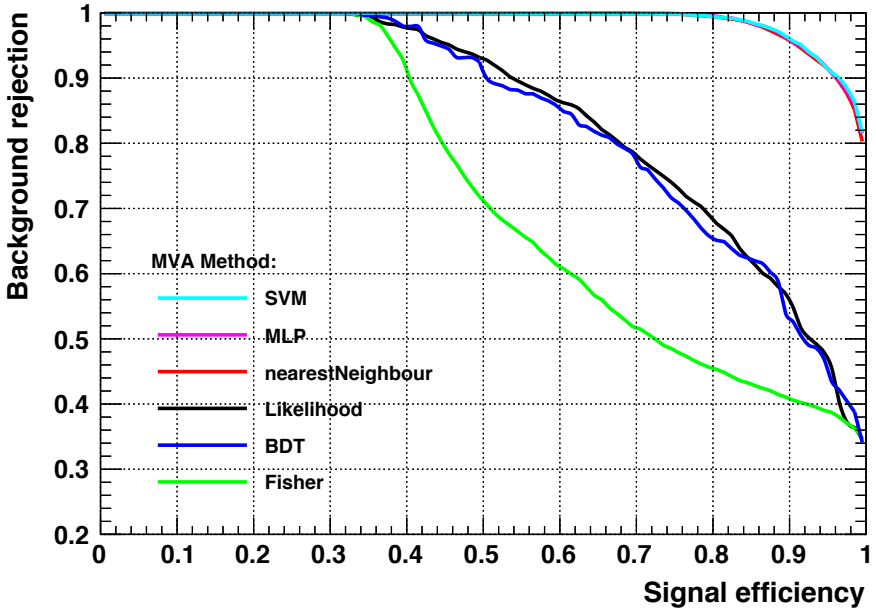
TMVA default: 5%

- AdaBoost, NTrees=500, MinNodeSize=0.05, AdaBoostBeta=0.5, MaxDepth=3, nCuts = 20, SeparationType=GiniIndex, 10000 Ereignisse
- Training: 2 Sekunden, Testing: 0.5 Sekunden
- Schlechte Trennung und p(KS)=0

TMVA overtraining check for classifier: BDT



Background rejection versus Signal efficiency

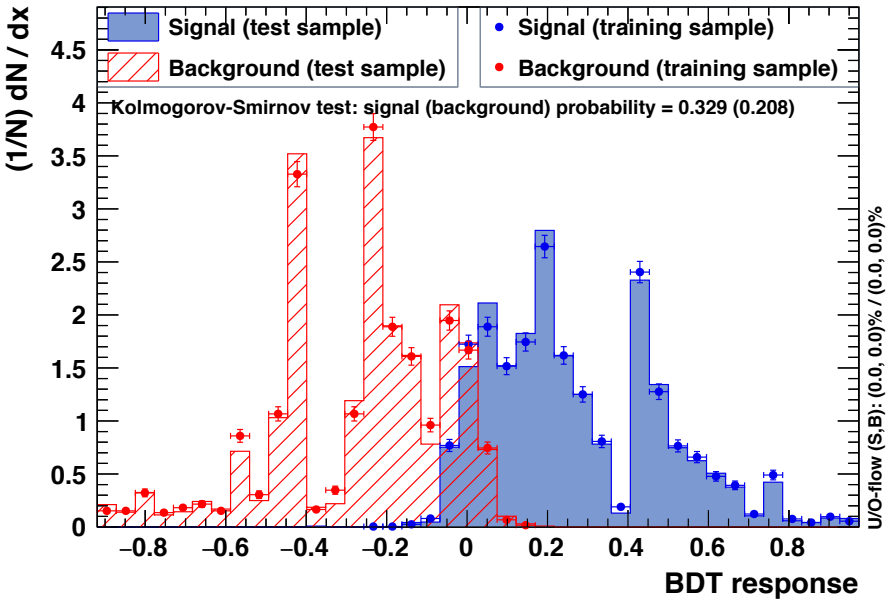


Boosted Decision Tree in TMVA

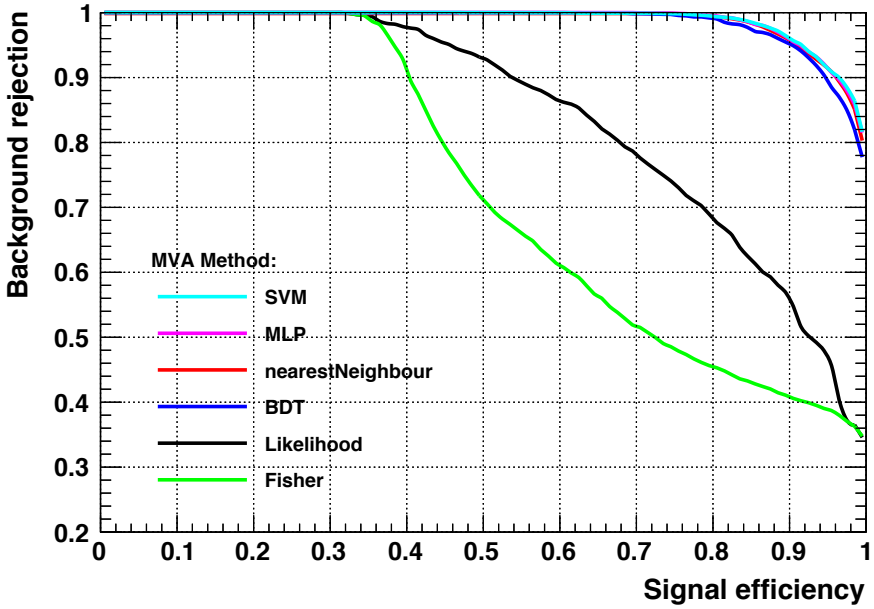
50%

- AdaBoost, NTrees=500, MinNodeSize=0.5, AdaBoostBeta=0.5, MaxDepth=3, nCuts = 20, SeparationType=GiniIndex, 10000 Ereignisse
- Training: 2 Sekunden, Testing: 0.5 Sekunden
- Gute Trennung

TMVA overtraining check for classifier: BDT



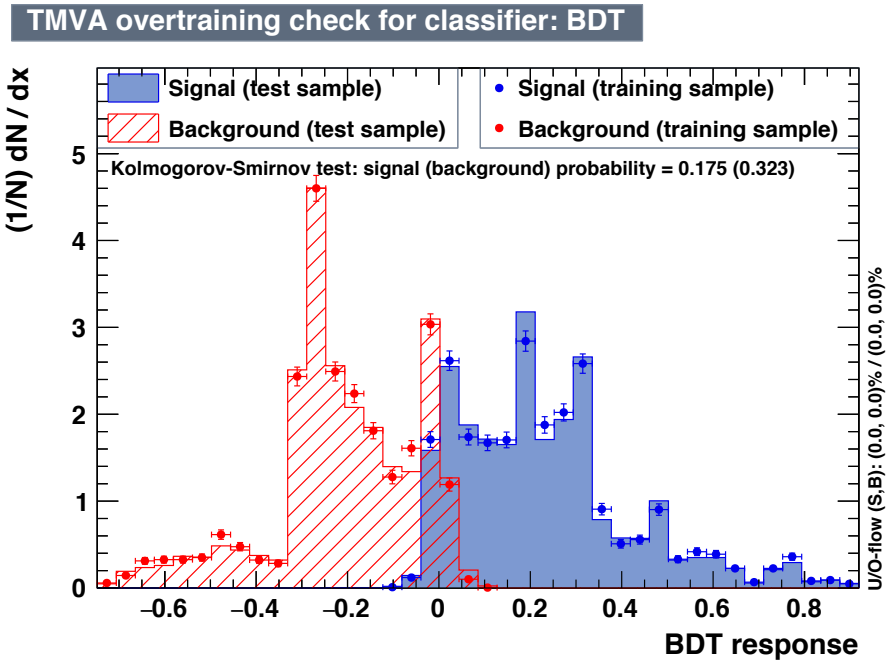
Background rejection versus Signal efficiency



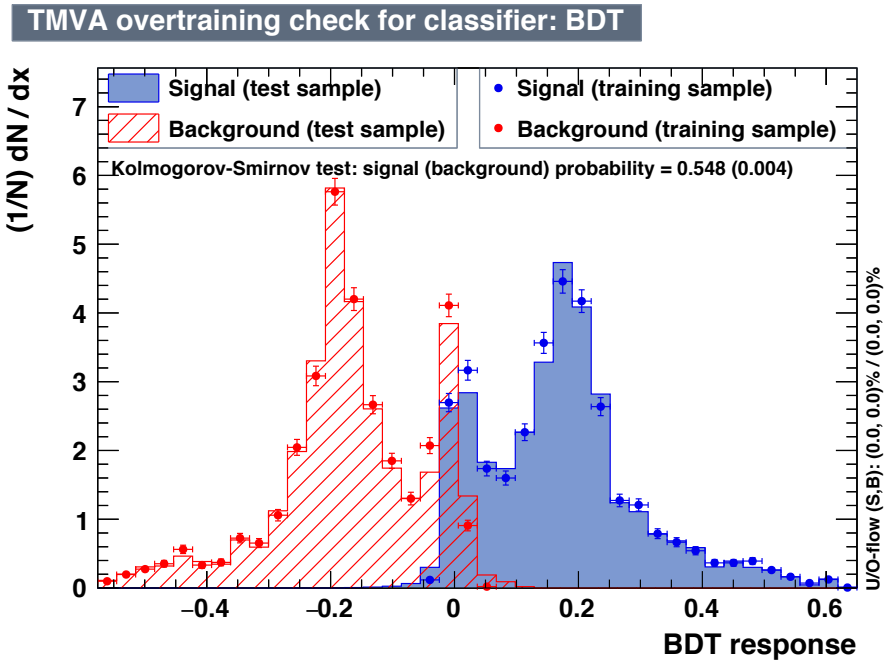
Boosted Decision Tree in TMVA

- AdaBoost, NTrees=5000, MinNodeSize=s.u. , AdaBoostBeta=0.5, MaxDepth=3, nCuts = 20, SeparationType=GiniIndex, 10000 Ereignisse
- Training: 23 Sekunden, Testing 5-6 Sekunden
- Gute Trennung

TMVA default: 5%



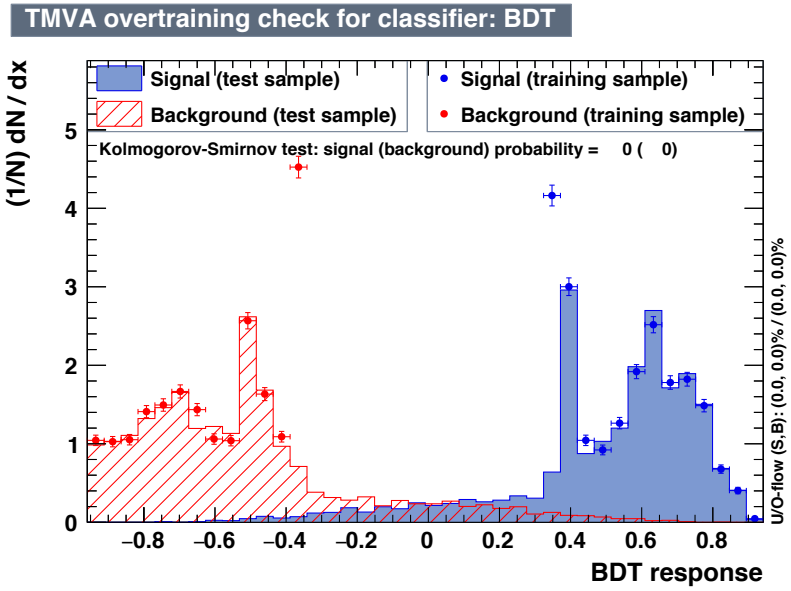
TMVA default: 50%

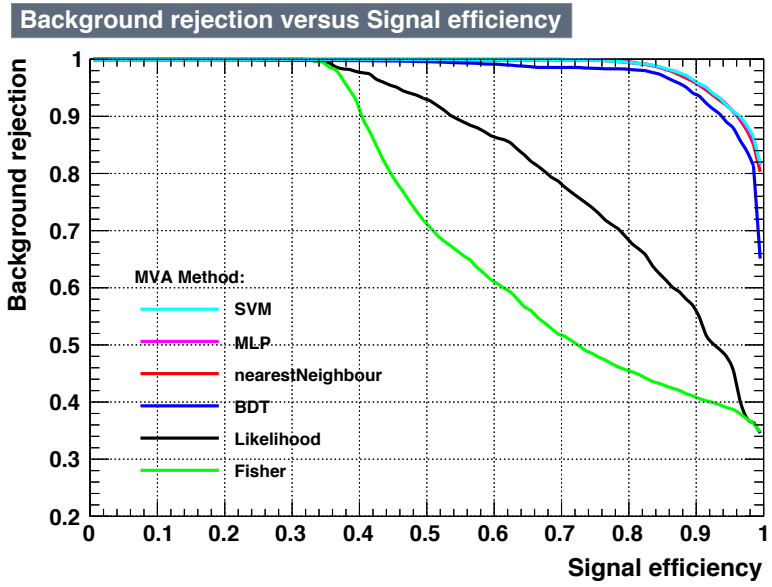


Beispiel: “python TrainEvaluateResponse_v11.py Circle”

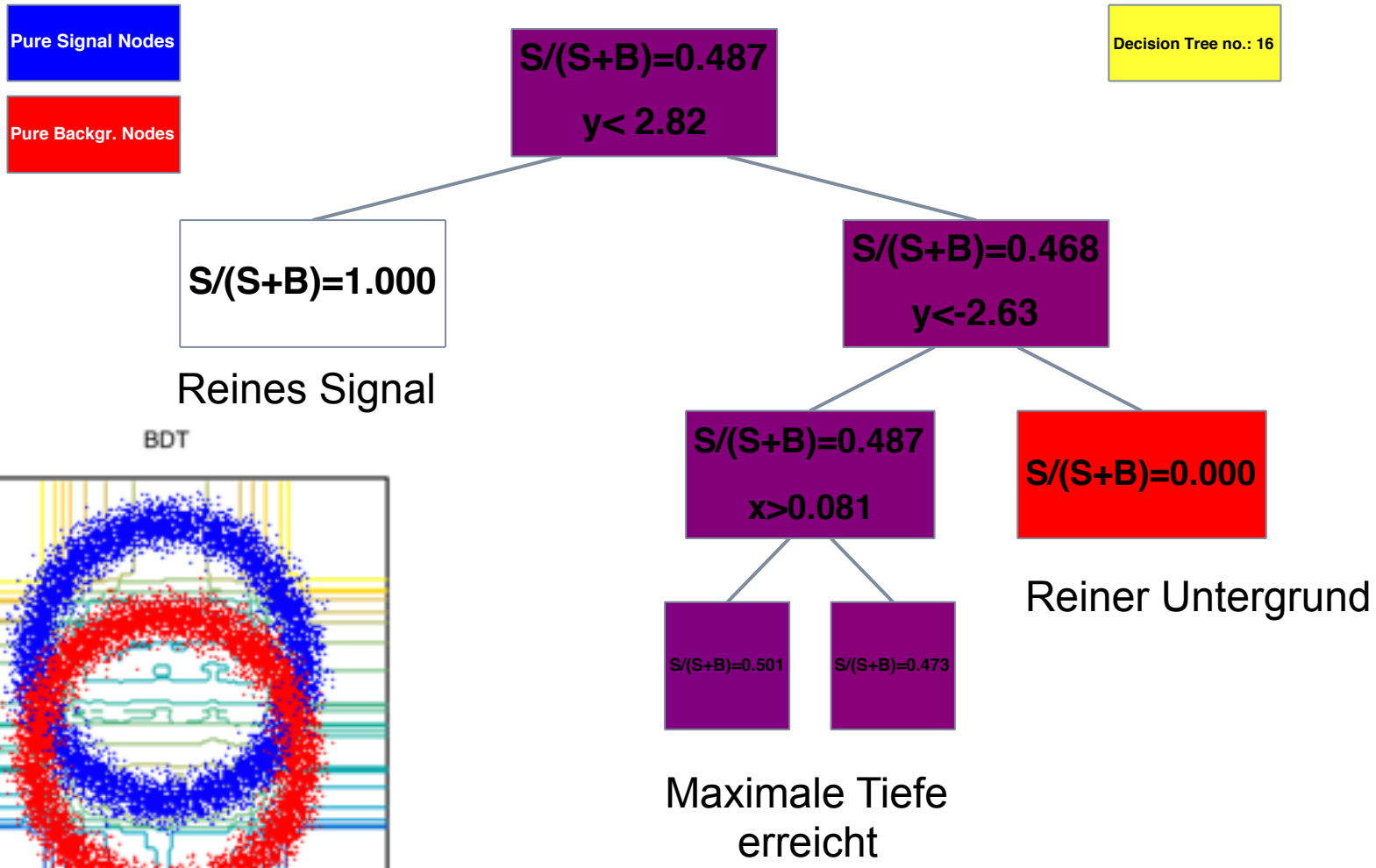
Boosted Decision Tree in TMVA

- AdaBoost, NTrees=5000, MinNodeSize=0.05 , AdaBoostBeta=0.5, MaxDepth=30, nCuts = 20, SeparationType=GiniIndex, 10000 Ereignisse
- Training: 230 Sekunden, Testing 90 Sekunden
- Übertraining! Nicht-optimale Trennung





Entscheidungsbäume

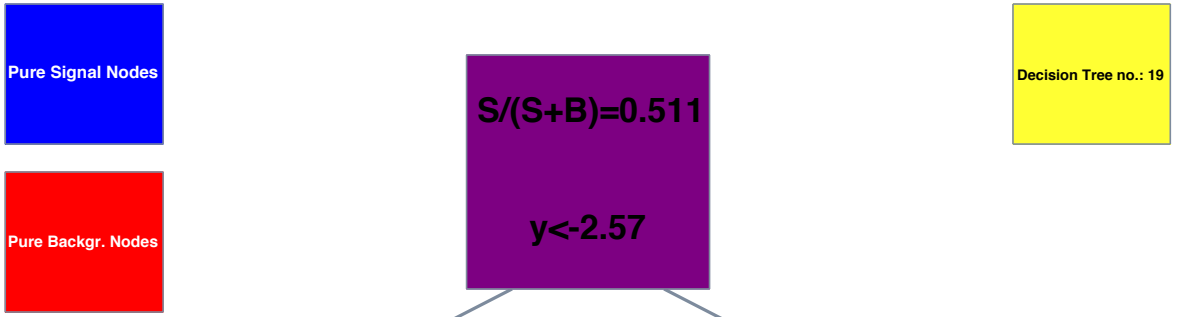


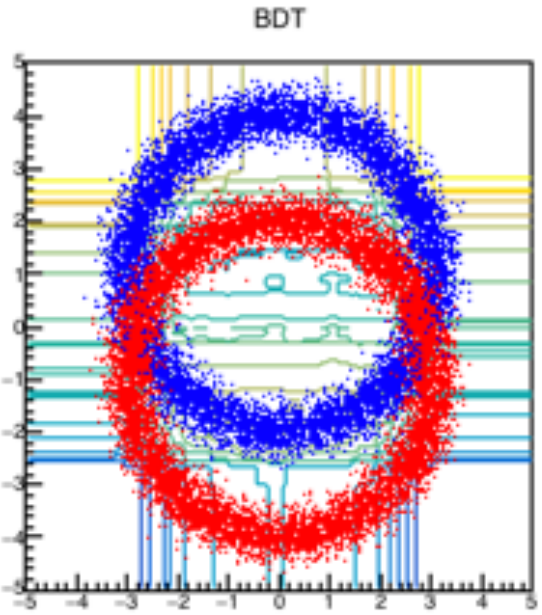
Reines Signal

Reiner Untergrund

Maximale Tiefe erreicht

Entscheidungsbäume

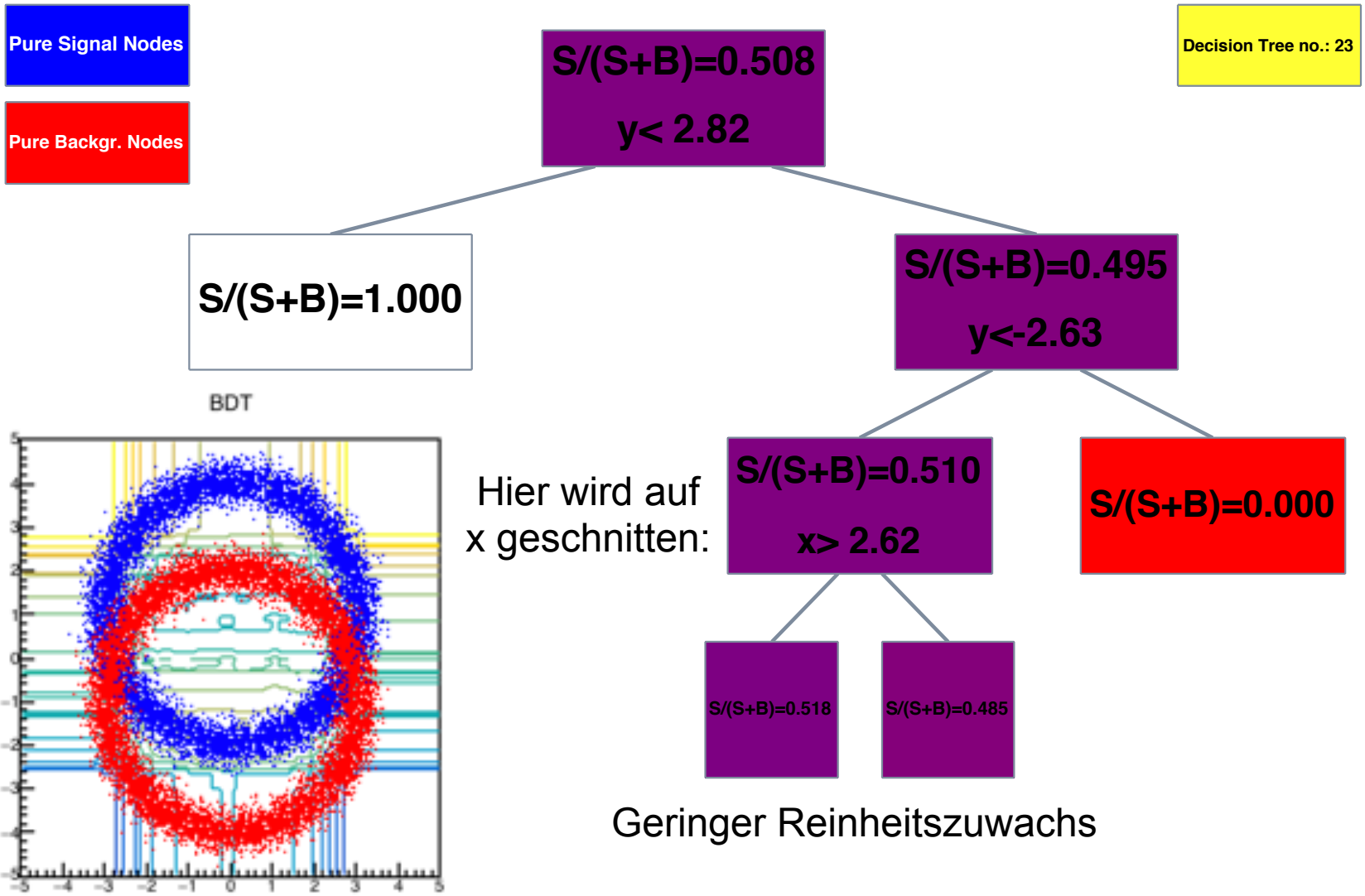




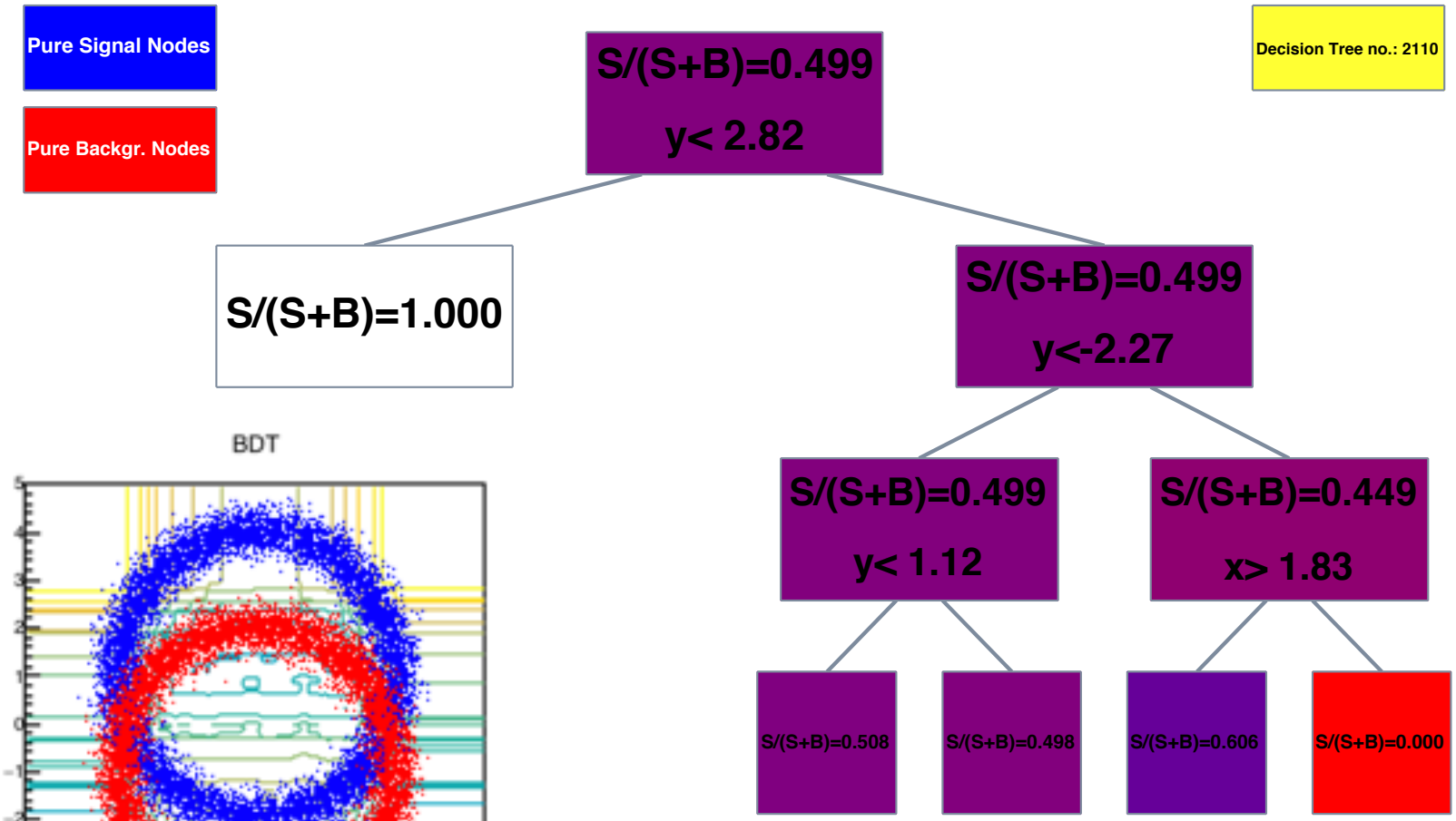
Vermutlich: Reinheitszuwachs durch nächsten Schnitt zu gering

Reinheitskriterium erreicht

Entscheidungsbäume



Entscheidungsbäume



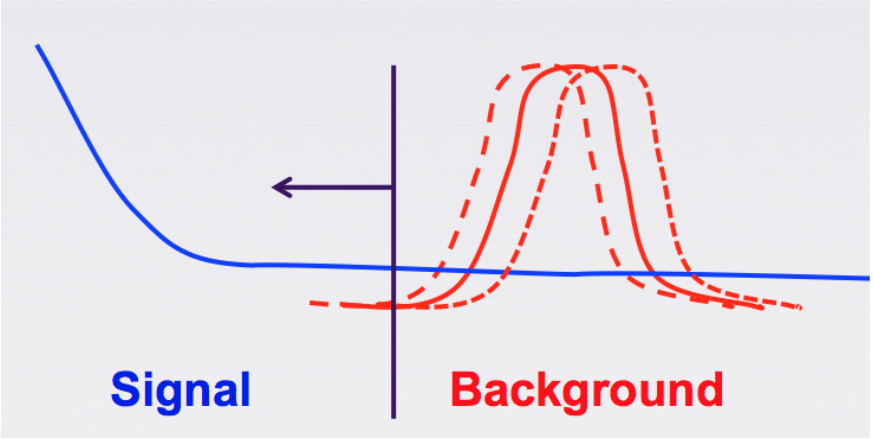
Hier kein Untergrund mehr nach Schnitt auf x

Zusammenfassung BDT

- Ensemble-Methode: Mittelwert aus vielen verschiedenen einfachen Modellen (weak learners) liefert ein komplexes Modell.
- Wälder aus verstärkten Entscheidungsbäumen (BDT) sind in der Teilchenphysik sehr populär
- Eigenschaften:
  - Lokal 1-dimensionale Entscheidungen, funktioniert auch in vielen Dimensionen
  - Schnelles Verwerfen offensichtlicher Untergrundereignisse, robust gegenüber Outliern u.dergleichen
  - Keine Abhängigkeit von Metrik, keine Vorverarbeitung (Normierung etc.) notwendig
  - Relativ wenige Parameter, wenig Tuning-Bedarf
  - Vergleichsweise schnelles Training
  - Bäume (mit wenigen Ästen) sind leicht verständlich
  - Variablen sollten möglichst dekorreliert sein - in Physik kein echtes Problem
  - TMVA liefert Ranking (Relevanz) der Variablen
  - Über-Training - kann kontrolliert werden

MVA und systematischer Fehler

- Bisher Diskussion auf rein statistischer Basis. Systematischer Fehler ist in der Praxis oft wichtiger, d.h. größer, als statistischer Fehler.
- Behandlung wie bei nicht-multivariaten Methoden: Variation der Input-Verteilungen im Analyse-Sample. Klassifikator bleibt unverändert.
- Problem: Variablen mit großer systematischer Unsicherheit können auf die Ausgabegröße durchschlagen → Lösung: Ausschluss, oder künstliche Abschwächung einer Input-Größe im Training.



H. Voss, Root-Workshop 2013

- Bemerkungen:
  - MVA müssen nicht im globalen Optimum verwendet werden
  - Über- oder Untertraining liefert keinen systematischen Fehler, ist nur nicht optimal.
  - Beispiel: Ereignisse mit negativem Gewicht → Ausschluss der Ereignisse oder Ignorieren der Gewichte

Zusammenfassung

- MVA sind leistungsfähige Werkzeuge.
  - Machine-learning ist ein hoch-aktuelles Feld. Immer mehr Anwendungen.
  - Viele Aspekte sind nicht vollständig erforscht
- Vorgestellte Methoden aus dem Bereich Supervised Learning:
  - Fisher Disk., k-Nearest Neighbours, ANN, BDT, SVM
  - Optimale Anwendung erfordert ein gutes Verständnis der Methode.
- Wahl der Methode, der Merkmalvektoren und der Parameter
  - Konfiguration spielt eine zentrale Rolle: Vorverarbeitung für Metrik, Normierung etc.
  - Training und Test der Ergebnisse
  - Die meisten MVA liefern ein Ranking der Variablen, berechnet aus (Inter-)Korrelationen und Effekt der Variablen auf das Ergebnis
- Ausführliche Untersuchungen am konkreten Problem sind zentraler Bestandteil von Datenanalysen.