

Neuronale Netze

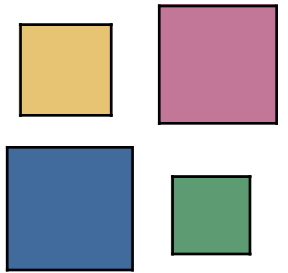
Neuronale Netze versuchen durch ein Netzwerk von einfachen Funktionen (den Neuronen) multivariate Eingabewerte \mathbf{x}_i in gewünschte Ausgabewerte \mathbf{y}_i zu verarbeiten.

Die Idee dahinter ist, dass jedes Neuron beim Trainieren des Neuronalen Netzes seinen Parameter Wert so lernt, dass der Fehler zwischen den geschätzten $\hat{\mathbf{y}}_i$ und den vorgegebenen \mathbf{y}_i möglichst klein wird.

Die einfachste Form eines Neuronalen Netzes ist:

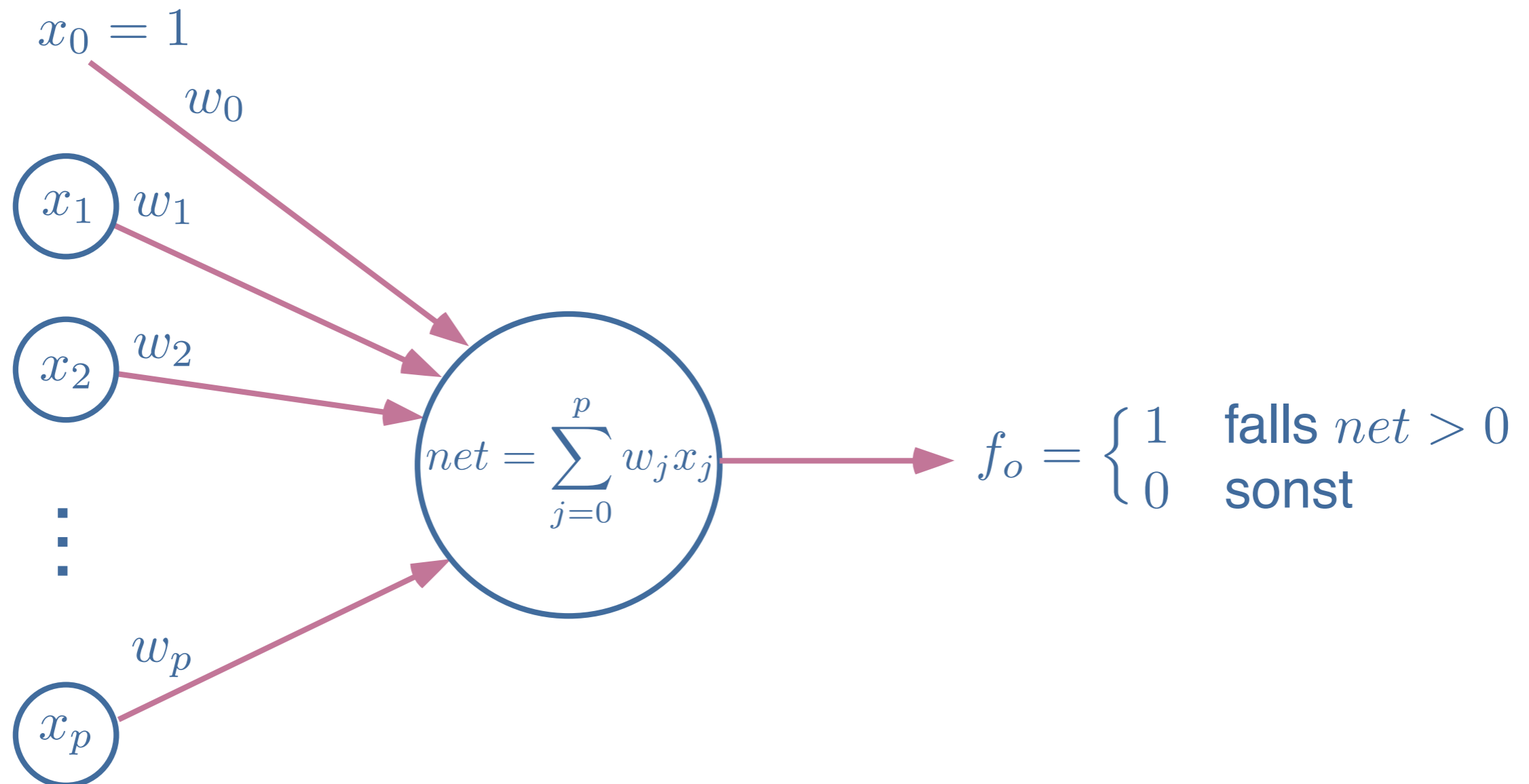
$$f_a(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^p w_j x_j$$

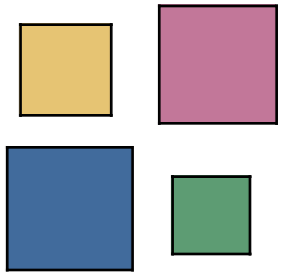
Die w_i sind dabei die zu lernenden Parameter im jeweiligen Neuron.



NNets: Schema

Minimale Darstellung eines Neuronales Netzes mit konstantem Input und binärer Output Funktion.





Lineare Separierbarkeit

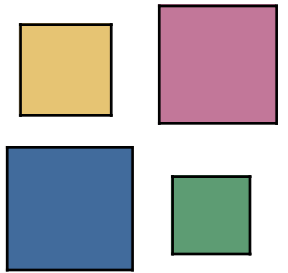
Welche logischen Funktionen lassen sich von einem minimalen (d.h. nur ein Neuron) Neuronalen Netz lernen?

| Funktion | w_0 | w_1 | w_2 |
|----------|-------|-------|-------|
| Negation | 0.5 | -1 | - |
| Oder | -0.5 | 1 | 1 |
| Und | -1.5 | 1 | 1 |
| XOr | ? | ? | ? |

Die “XOr” Funktion lässt sich nicht abbilden, da

$$w_0 + w_1x_1 + w_2x_2 = 0$$

eine lineare Funktion im \mathbb{R}^2 darstellt, d.h. alle Outputs einer “Sorte” müssen auf der gleichen Seite der Geraden liegen \Rightarrow lineare Separierbarkeit.



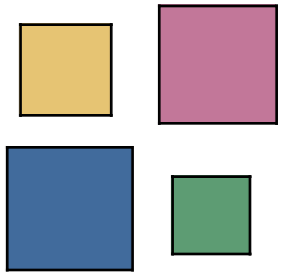
Trainieren des minimalen NNet

Im Fall des Neuronalen Netzes ohne hidden Layer können die w_j aus den Eingaben x_j folgendermaßen gelernt werden:

1. Initialisiere die w_j zufällig.
2. Berechne $\Delta w_j = \eta(y_i - \hat{y}_i)x_j$
3. Addiere Δw_j zu w_j
4. Wiederhole 2. und 3. bis $\sum \Delta w_j < \epsilon$

Dieser Algorithmus konvergiert, falls

- Die Trainingsdaten linear separierbar sind
- Die Schrittweite η klein genug gewählt ist.
- Keine versteckten Neuronen vorhanden sind

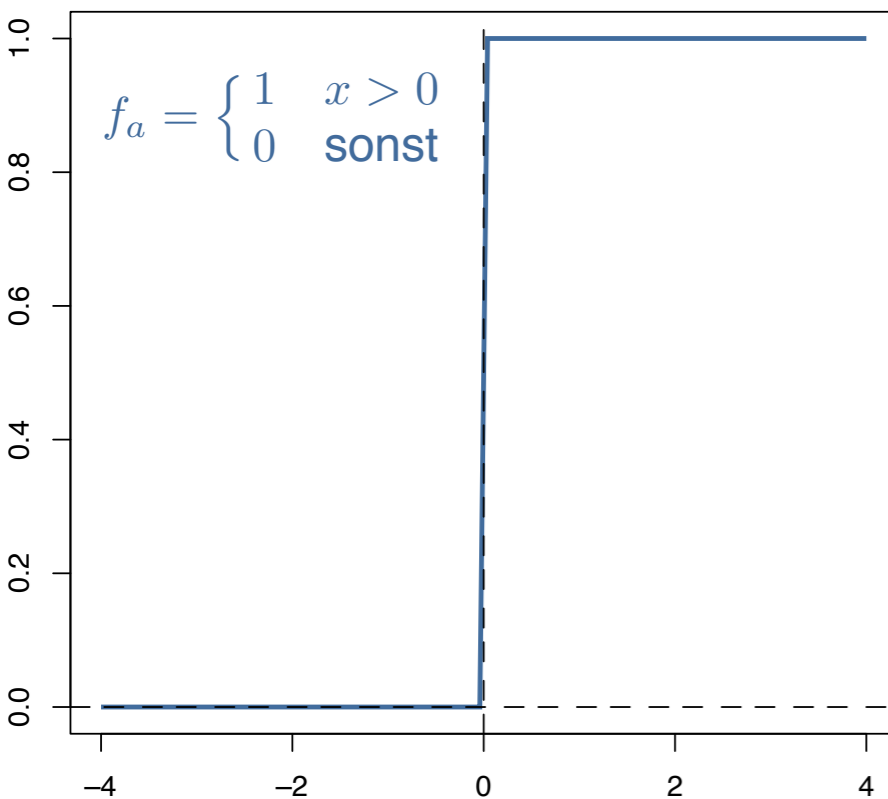


Aktivierungsfunktionen

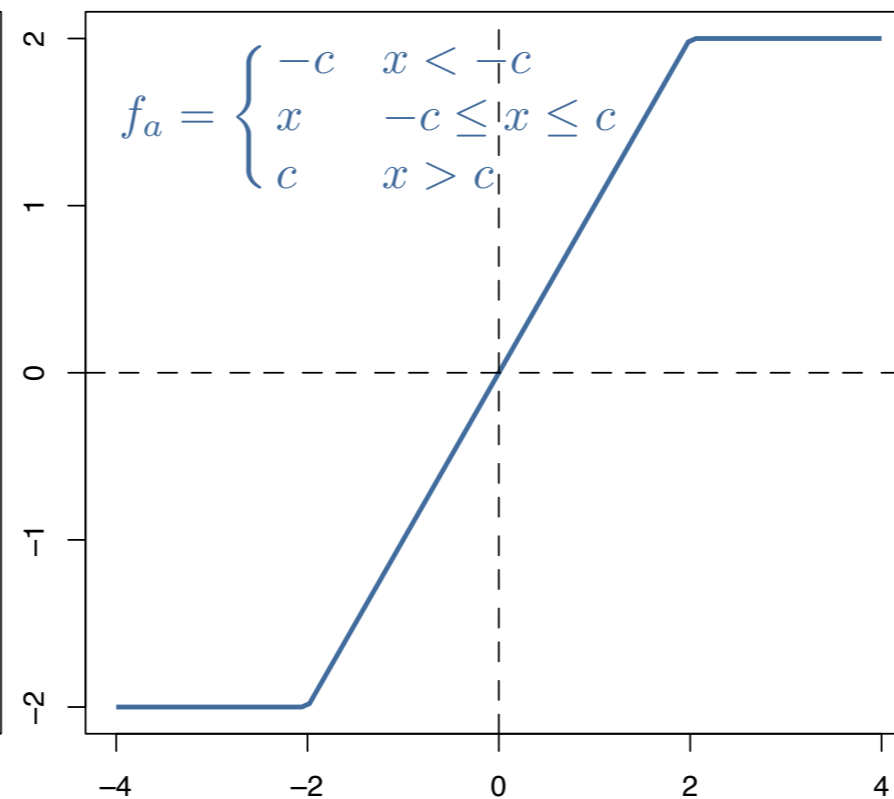
Als Ausgabefunktion, bzw. Aktivierungsfunktion f_a kann jede sigmoide Funktion verwendet werden, d.h. sie erfüllt:

- $f_a(x_1) \leq f_a(x_2), \forall x_1 \leq x_2$, d.h. f_a ist monoton wachsend
- $\lim_{x \rightarrow -\infty} f_a(x) \leq c_1$ und $\lim_{x \rightarrow \infty} f_a(x) \geq c_2$ und $c_1 < c_2$

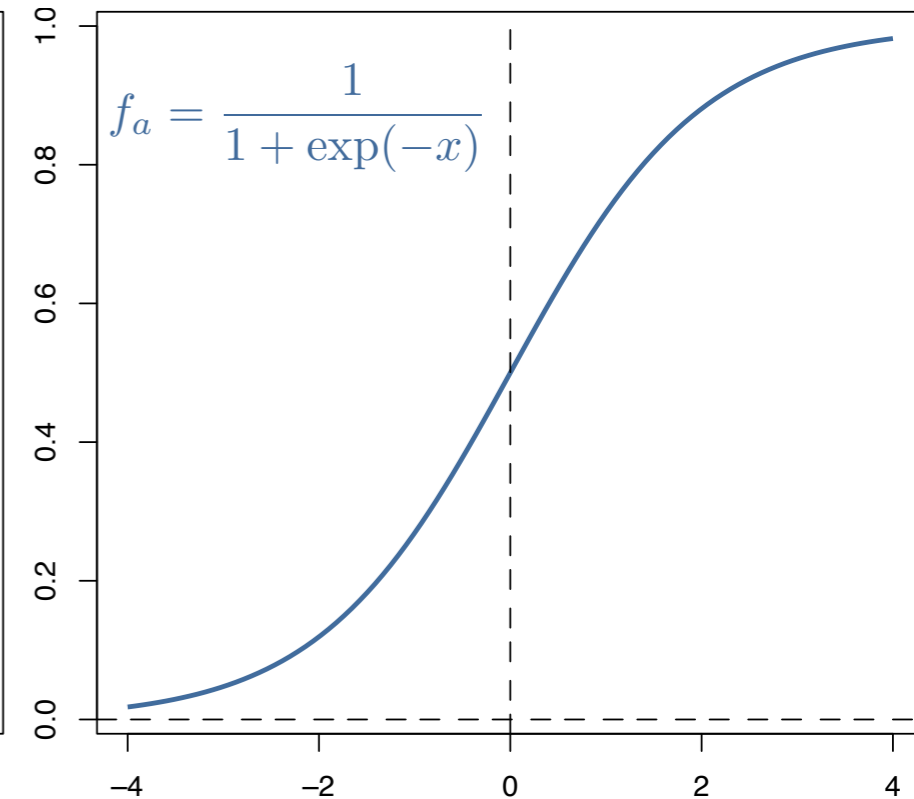
binär

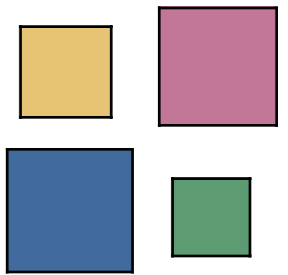


semi linear



logistisch





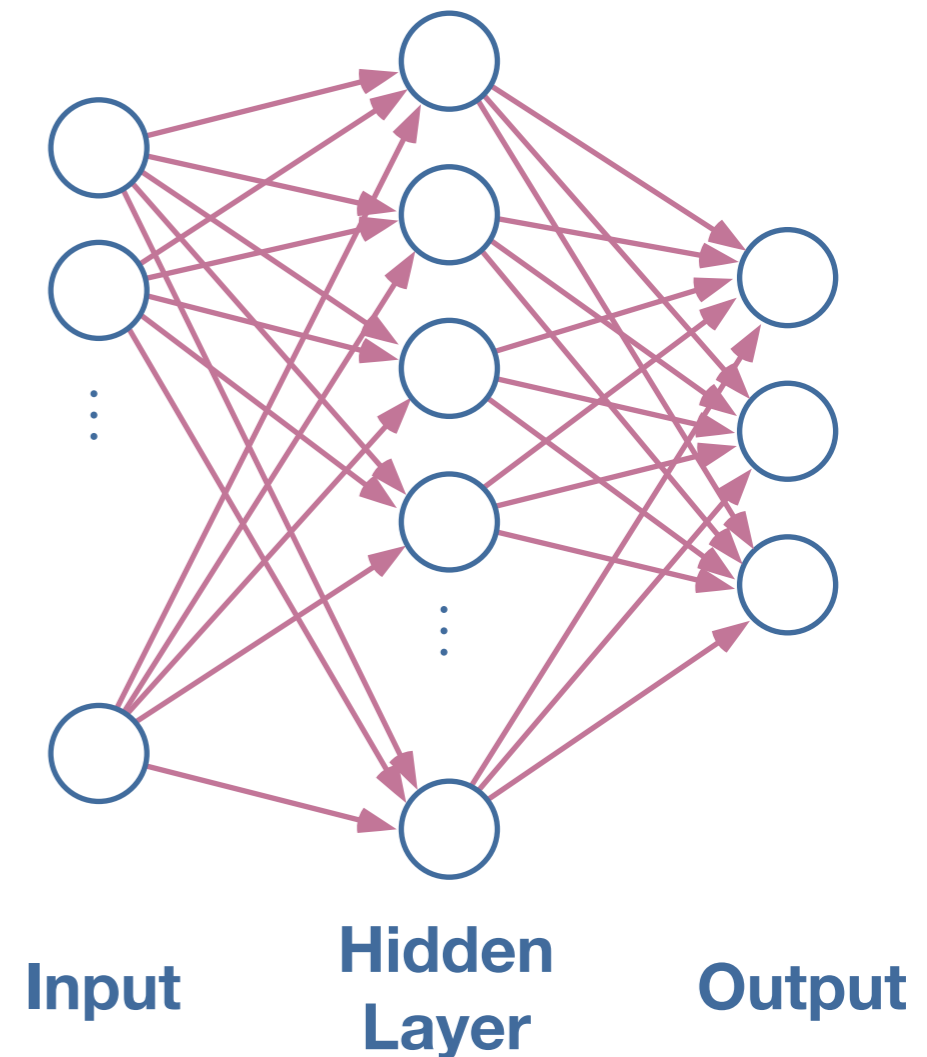
Hidden Layer

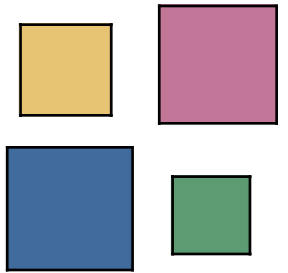
In realen Anwendungen ist eine Neuronales Netz mit nur einem Neuron uninteressant.

Daher werden sog. *hidden Layer* eingefügt, die die Komplexität der lernbaren Funktionen deutlich erhöht.

Für einen Output sieht die formale Darstellung eines Neuronales Netz mit p Inputs, s Neuronen im hidden Layer und einer Aktivierungsfunktion ϕ folgendermaßen aus.

$$\hat{y} = \phi\left(w_0 + \sum_{h=1}^s w_h \phi\left(w_{0h} + \sum_{j=1}^p w_{jh} x_j\right)\right)$$

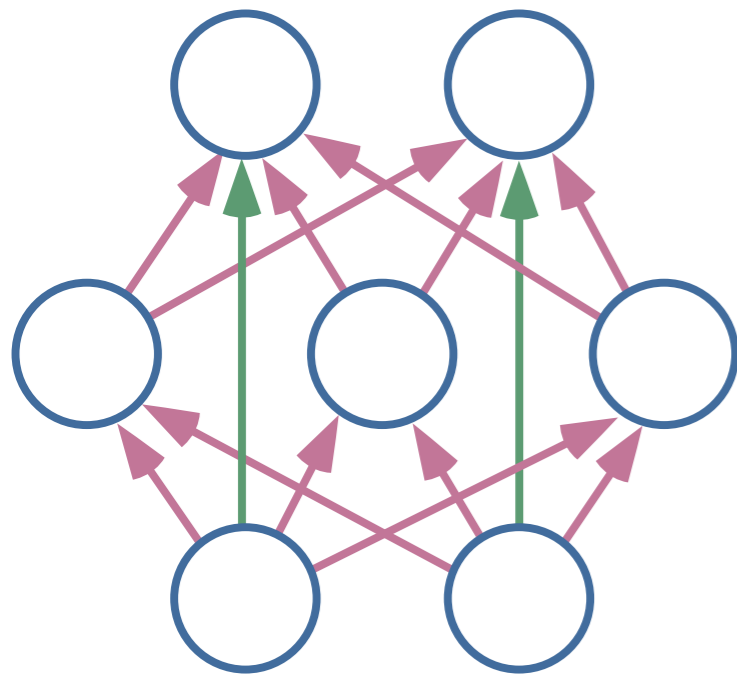




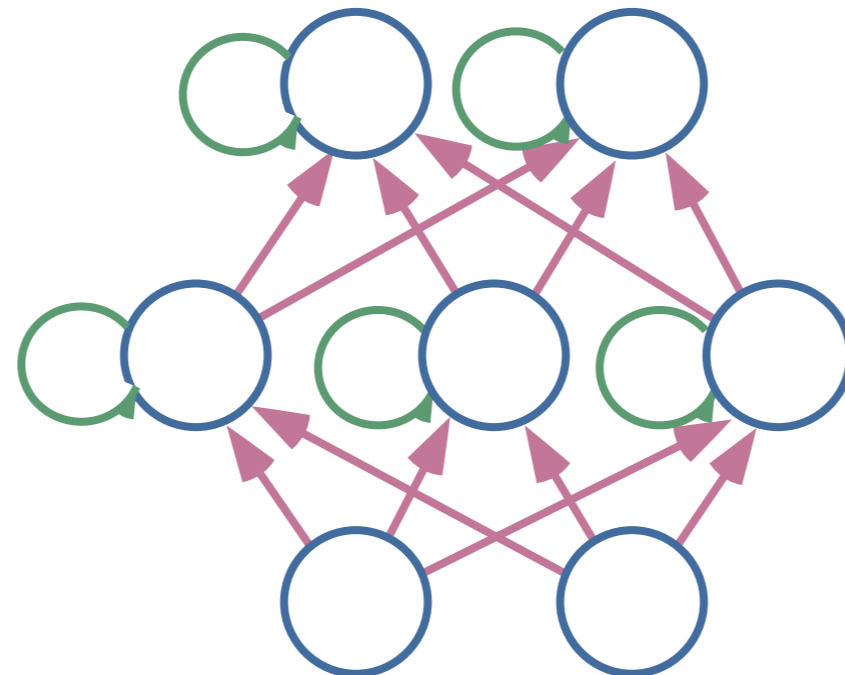
Feed Forward Netze

- Alle bisher betrachteten Neuronalen Netze waren *Feed Forward Netze*, d.h. jedes Neuron bekommt nur Inputs aus dem Layer direkt vor dem Neuron.
- Andere Formen sind auch möglich, z.B.

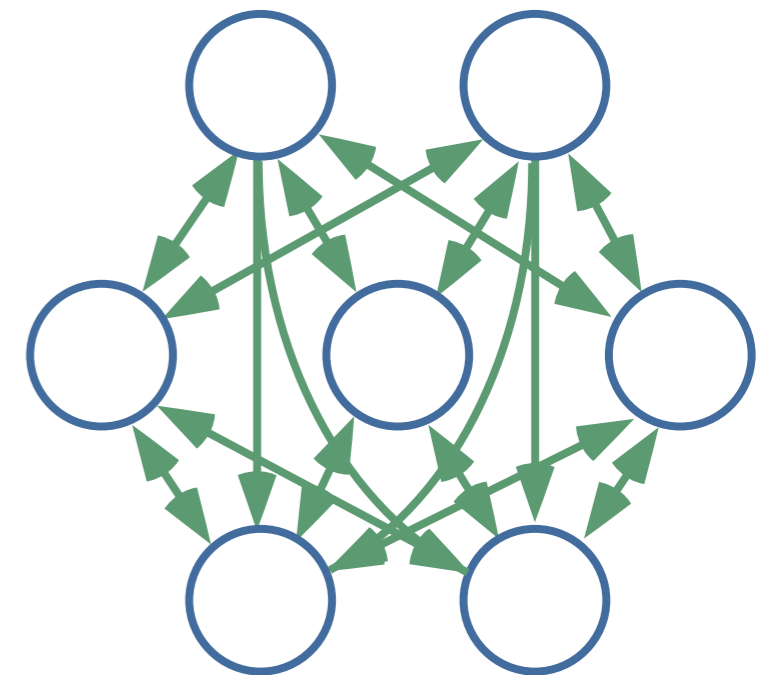
Skip Layer

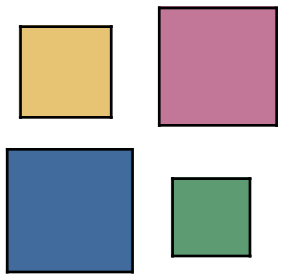


Direct Feedback



Indirect Feedback

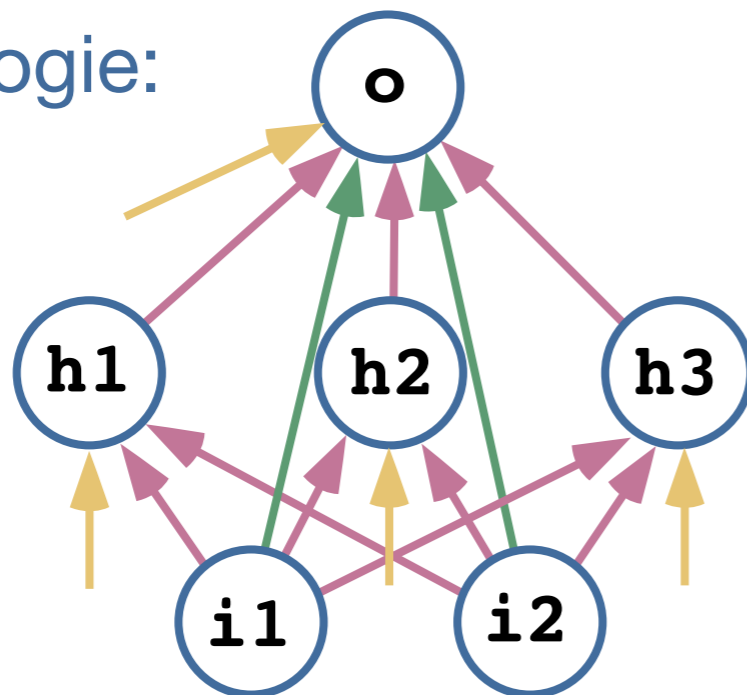




Beispiel Netz

- Neuronales Netz zur Vorhersage der Regionen der Olive Oil Daten aus den 2 Variablen *linoleic* und *eicosenoic*.
- 2 Inputs, 3 Units im Hidden Layer, 1 Output, Skip Layer.

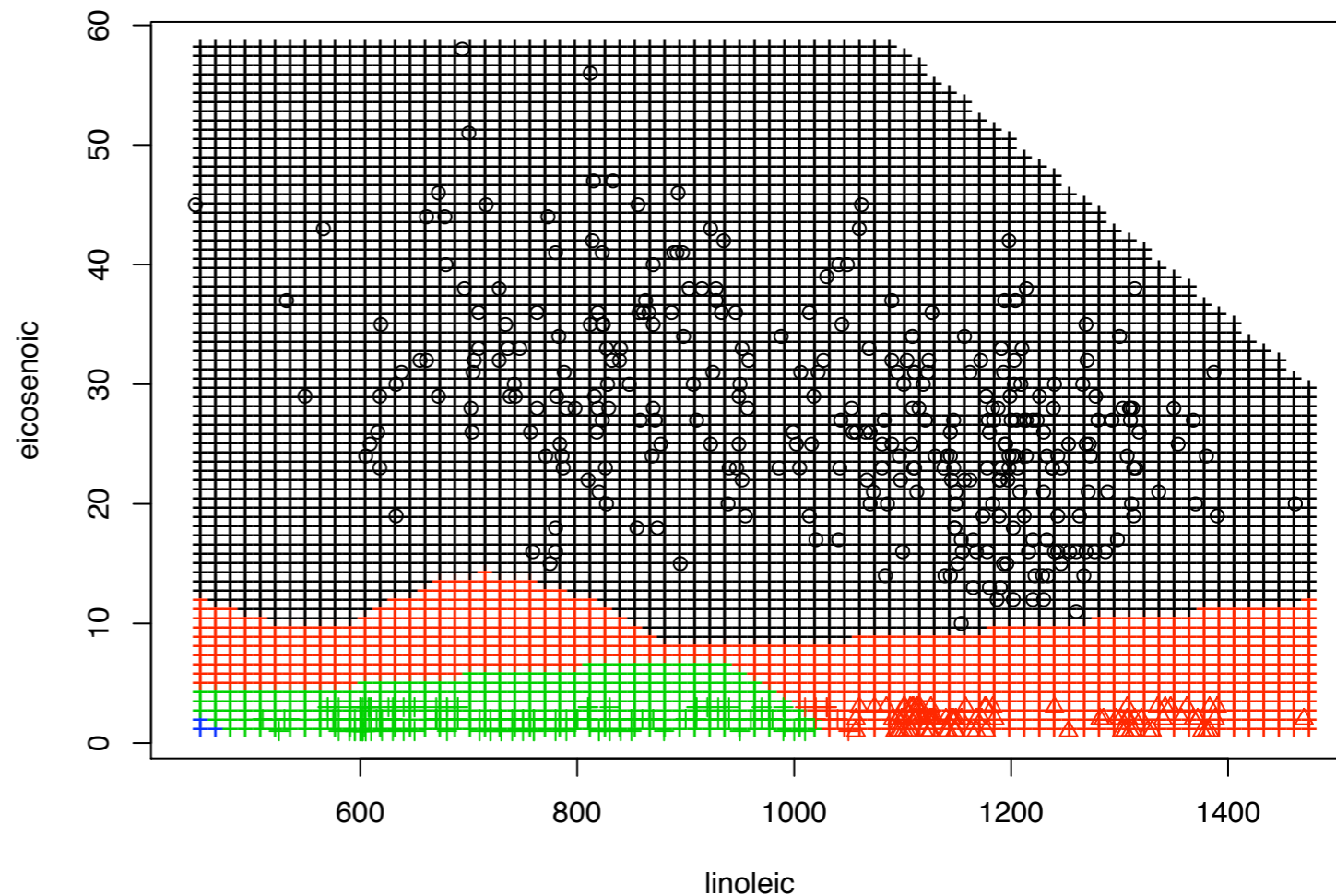
- Topologie:

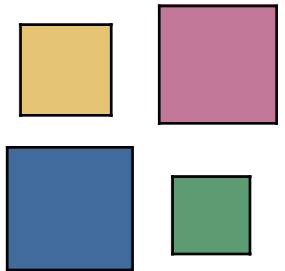


- Konfusion:

| NNet | Region | | |
|------------|------------|-----------|------------|
| | 1 | 2 | 3 |
| 1 | 323 | 0 | 0 |
| 2 | 0 | 98 | 7 |
| 3 | 0 | 0 | 144 |
| Sum | 323 | 98 | 151 |

- Regionen:





Beispiel Netz: R-Code

```
> library(nnet)
> attach(olives)
> on<-nnet(cbind(olives[,8], olives[,11]), olives[,2],
+         size=3, decay=0.0005, linout=T, skip=T, maxit=1000)
# weights: 15
initial value 225115861.092822
iter 10 value 67.700016
...
iter 360 value 9.631796
final value 9.631796
converged

> summary(on)
a 2-3-1 network with 15 weights
options were - skip-layer connections linear output units decay=5e-04
b->h1 i1->h1 i2->h1
-6.89  0.01  0.08
b->h2 i1->h2 i2->h2
-5.39  0.00  0.05
b->h3 i1->h3 i2->h3
 1.66  0.03 -3.47
b->o h1->o h2->o h3->o i1->o i2->o
 7.07  3.85 12.74  1.07 -0.01 -0.15

> table(Region, round(predict(on)))
Region 1  2  3
      1 323  0  0
      2  0  98  0
      3  0  7 144

> rast.x<- rep(seq(min(linoleic), max(linoleic),length=75), 75)
> rast.y<- rep(seq(min(eicosenoic), max(eicosenoic), length=75), rep(75, 75))
> pred<-predict(on, newdata=cbind(rast.x, rast.y))
> plot(linoleic, eicosenoic, col=round(predict(on)), pch=olives$Region)
> points(rast.x, rast.y, col=round(pred), pch="+")
```