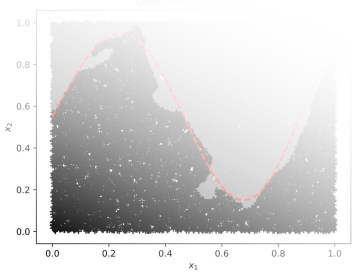
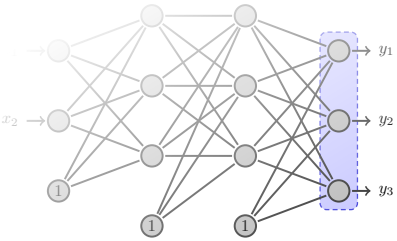
Regularisierung und Batch-Normalization

Prof. Dr. Jörg Frochte

Maschinelles Lernen





Notwendigkeit für eine Regularisierung

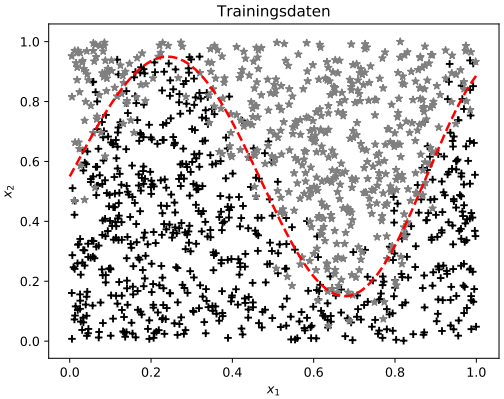
- Es gelingt selten genau die richtige Anzahl an Freiheitsgraden zu modellieren. In der Regel wird eher zu viel, als zu wenig vorgesehen.
- Damit steigt die Wahrscheinlichkeit für das Overfitting bei tiefen Netzen tendenziell an.
- Die grundlegende Idee findet man bereits bei Breiman *Bias-variance, regularization, instability and stabilization* im Jahr 1998, und diese besteht darin, dass man die Komplexität des Modells zum Teil der Fehlerfunktion macht, die man zu minimieren versucht.
- Hierzu koppelt man die Komplexität des Modells mit einem Faktor $\lambda \geq 0$ als Summand an die Zielfunktion an:

$$J = \text{Fehler auf den Trainingsdaten} + \lambda \cdot \text{Modellkomplexität} \tag{1}$$

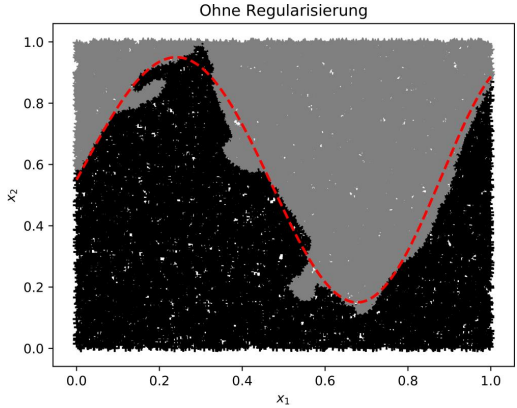
- Im Fall von neuronalen Netzen sind die Gewichte $w_{ij}^{[l]}$ das Maß der Modellkomplexität.
- l gibt dabei die jeweilige Matrix der Gewichte an. Ein Gewicht mit dem Wert Null reduziert dabei die Komplexität, da diese Verbindung nicht genutzt wird.

Overfitting

- Man kann auch zu viel Freiheitsgrade gewähren und zu lange trainieren.
- Dann entsteht ggf. ein Overfitting wie hier dargestellt.



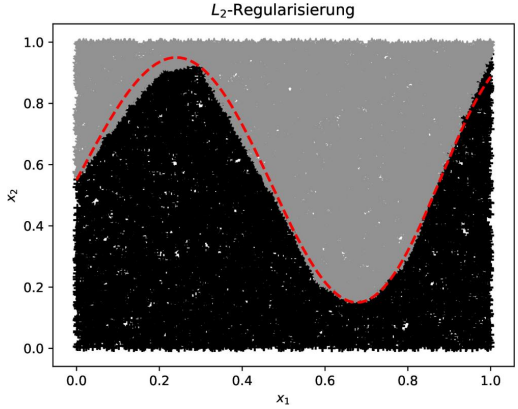
Trainingsset mit verrauschtem Rand

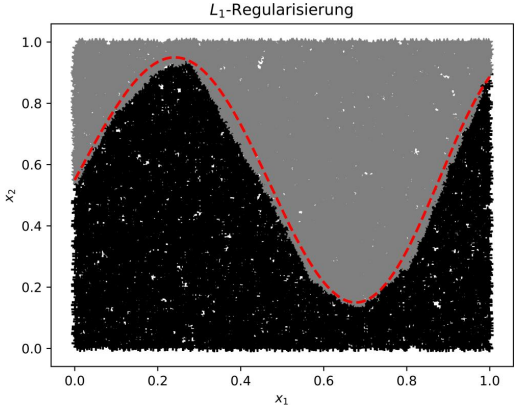


Klassifikation ohne Regularisierung

L_1- und L_2-Regularisierung

$$J(W) = \sum_{(x_d, y_d) \in D} \frac{1}{2} \left(y_d - y(x_d, W)\right)^2 + \sum_{l=1}^{n} \lambda_l \cdot \|W^{(l)}\|^p \quad \text{mit} \quad \|W^{(l)}\|^p = \sum_{i=1}^{h^{(l)}} \sum_{j=1}^{h^{(l-1)}} \left| w_{ij}^{(l)} \right|^p \quad \text{und} \quad W^{(l)} \in \mathbb{R}^{h^{(l)} \times h^{(l-1)}}$$





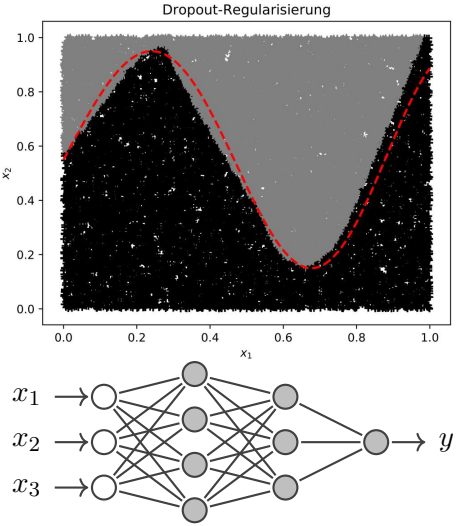
L_1- und L_2-Regularisierung in Keras

```
from tensorflow.keras import regularizers
myANN = Sequential()
myANN.add(Dense(256,input_dim=2,kernel_initializer='normal',activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)))
myANN.add(Dense(256,kernel_initializer='random_uniform',activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)))
myANN.add(Dense(128,kernel_initializer='random_uniform',activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)))
myANN.add(Dense(128,kernel_initializer='random_uniform',activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)))
myANN.add(Dense(2,kernel_initializer='normal',activation='sigmoid'))
myANN.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
myANN.fit(XTrain,YTrain, epochs=500,batch_size=20)
```

- Bei der L_1- und L_2-Regularisierung wird der nötige Parameter dem Layer zusätzlich übergeben.

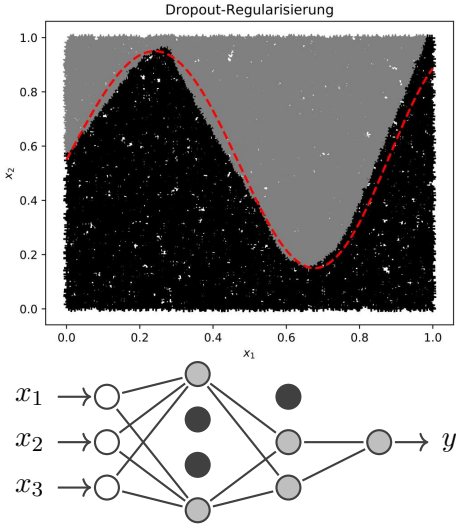
Dropout-Strategie

- Das Prinzip ist, beim Training zufällig bestimmte Neuronen in einer Schicht zu deaktivieren, indem deren Outputs auf Null gesetzt werden.
- Häufig wird als Parameter bis zu 50% gewählt:
  `ann.add(Dropout(0.5))`



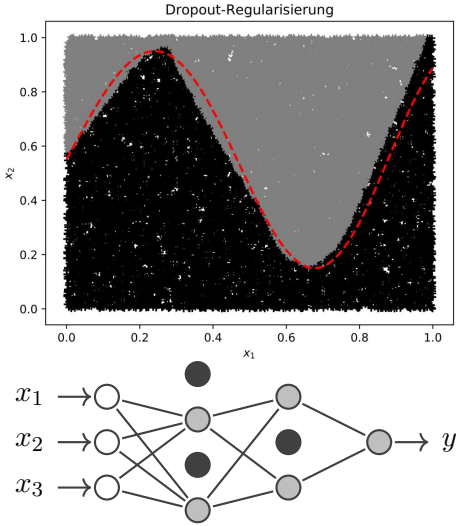
Dropout-Strategie

- Das Prinzip ist, beim Training zufällig bestimmte Neuronen in einer Schicht zu deaktivieren, indem deren Outputs auf Null gesetzt werden.
- Häufig wird als Parameter bis zu 50% gewählt: `ann.add(Dropout(0.5))`
- Durch deaktivierten Neuronen fließen keine Gradienten.
- Folglich werden redundante Wege im Netz ausgebildet und es wird dadurch robuster.



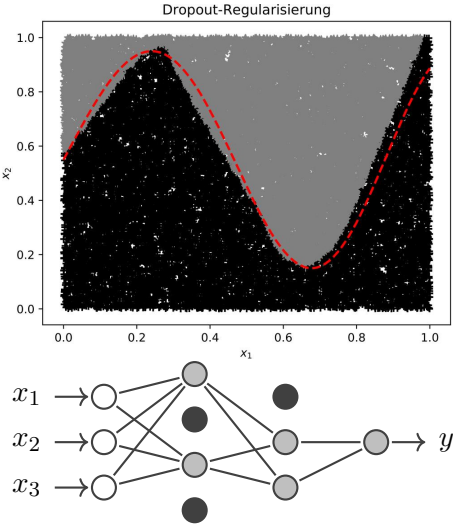
Dropout-Strategie

- Das Prinzip ist, beim Training zufällig bestimmte Neuronen in einer Schicht zu deaktivieren, indem deren Outputs auf Null gesetzt werden.
- Häufig wird als Parameter bis zu 50% gewählt: `ann.add(Dropout(0.5))`
- Durch deaktivierten Neuronen fließen keine Gradienten.
- Folglich werden redundante Wege im Netz ausgebildet und es wird dadurch robuster.



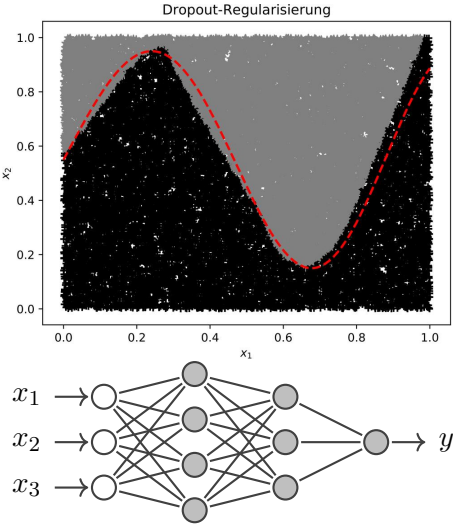
Dropout-Strategie

- Das Prinzip ist, beim Training zufällig bestimmte Neuronen in einer Schicht zu deaktivieren, indem deren Outputs auf Null gesetzt werden.
- Häufig wird als Parameter bis zu 50% gewählt: `ann.add(Dropout(0.5))`
- Durch deaktivierten Neuronen fließen keine Gradienten.
- Folglich werden redundante Wege im Netz ausgebildet und es wird dadurch robuster.
- Eine weitere Interpretation ist, dass man die Teilnetze als Ensemble Lerner auffassen kann.



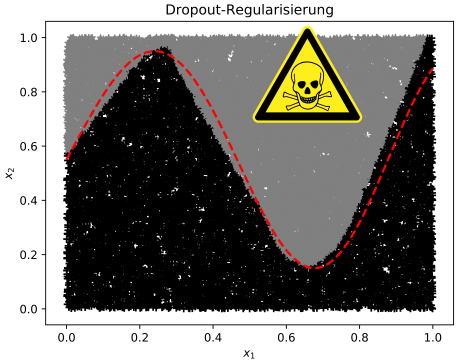
Dropout-Strategie

- Das Prinzip ist, beim Training zufällig bestimmte Neuronen in einer Schicht zu deaktivieren, indem deren Outputs auf Null gesetzt werden.
- Häufig wird als Parameter bis zu 50% gewählt: `ann.add(Dropout(0.5))`
- Durch deaktivierten Neuronen fließen keine Gradienten.
- Folglich werden redundante Wege im Netz ausgebildet und es wird dadurch robuster.
- Eine weitere Interpretation ist, dass man die Teilnetze als Ensemble Lerner auffassen kann.
- Nach dem Training wird der Dropout automatisch nicht mehr angewendet.



Dropout-Regularisierung und Softwarepatente

- Die Dropout-Technik wurde von Geoffrey Hinton 2014 in *Dropout: A simple way to prevent neural networks from overfitting* veröffentlicht und gewann schnell an Bedeutung.
- Erst langsam wurde nach der akademischen Veröffentlichung offenbar, dass diese Technik im Dezember 2012 von u. a. Hinton zum Patent angemeldet wurde.
- Negativbeispiel für Software-Patente in diesem Bereich und damit Patente auf Mathematik



- Tensorflow gehört zu Google und steht unter der Apache 2.0 Lizenz, welche auch die Verwendung von Patenten umfasst.
- Apache Lizenz zweitbeste Möglichkeit bei Softwarepatenten, die Beste ist Prior Art schaffen und Patenten verhindern.

Dropout-Regularisierung in Keras

```
from keras.layers import Dropout
myANN = Sequential()
myANN.add(Dense(256,input_dim=2,kernel_initializer='normal',activation='relu'))
myANN.add(Dropout(0.5))
myANN.add(Dense(256,kernel_initializer='random_uniform',activation='relu'))
myANN.add(Dropout(0.5))
myANN.add(Dense(128,kernel_initializer='random_uniform',activation='relu'))
myANN.add(Dropout(0.5))
myANN.add(Dense(128,kernel_initializer='random_uniform',activation='relu'))
myANN.add(Dense(2,kernel_initializer='normal',activation='sigmoid'))
myANN.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
myANN.fit(XTrain,YTrain, epochs=500,batch_size=20)
```

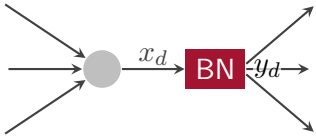
- Dropout-Regularisierung wird mittels einem eigenen Layer realisiert.

Batch-Normalization

Batch-Normalization[1] (BN) standardisiert den Output eines Neurons. Beim Training:

- Die Outputs eines Neurons x_d (Sample d) werden über einen gesamten Mini-Batch $D = \{x_1, x_2, \dots\}$ gesammelt.
- Der Batch-Mittelwert μ_D und die Batch-Standardabweichung σ_D werden bestimmt und der Output wird standardisiert.
- Zwei Parameter β und γ werden beim Training gelernt.
- Diese legen nach der Standardisierung den Erwartungswert β und die Standardabweichung γ fest.

Dies passiert für jedes Neuron unabhängig.



$$\hat{x}_d = \frac{x_d - \mu_D}{\sigma_D}$$

$$y_d = \gamma\,\hat{x}_d + \beta$$

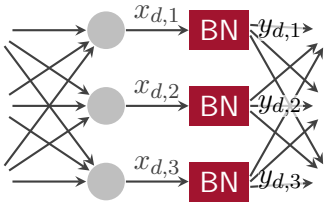
[1] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015

Batch-Normalization

Batch-Normalization[1] (BN) standardisiert den Output eines Neurons. Beim Training:

- Die Outputs eines Neurons $x_{d,j}$ (Sample d, Feature j) werden über einen gesamten Mini-Batch $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ gesammelt.
- Der Batch-Mittelwert $\mu_{D,j}$ und die Batch-Standardabweichung $\sigma_{D,j}$ werden für Feature j bestimmt und der Output wird standardisiert.
- Zwei Parameter β_j und γ_j werden beim Training gelernt.
- Diese legen nach der Standardisierung den Erwartungswert β_j und die Standardabweichung γ_j fest.

Dies passiert für jedes Neuron unabhängig.



$$\hat{x}_{d,j} = \frac{x_{d,j} - \mu_{D,j}}{\sigma_{D,j}}$$

$$y_{d,j} = \gamma_j \, \hat{x}_{d,j} + \beta_j$$

[1] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015

Batch-Normalization

Effekte:

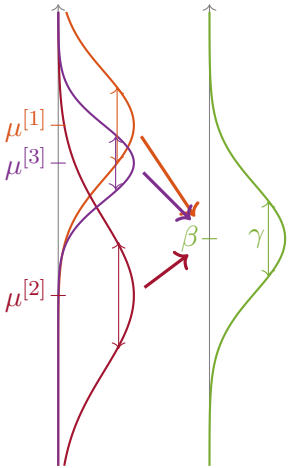
- Die Gewichts-Updates der vorderen Schichten beeinflussen die Verteilung der Input-Werte der hinteren Schichten kaum noch.
- Das Netz konvergiert viel schneller, insb. für sehr tiefe Netze.
- Es gibt eine leicht regularisierende Wirkung.
- Batch-Normalization wird meistens bei ReLU angewendet, kann negative Effekte bei $\tanh$ haben.

Nach dem Training:

- Ein gleitender Mittelwert über die Batch-Mittelwerte $\bar{\mu}$ und ein gleitender Mittelwert über die Batch-Standardabweichungen $\bar{\sigma}$ werden zur Standardisierung verwendet.

Umsetzung (vor oder nach Aktivierung):

`ann.add(BatchNormalization())` ggf. bei ReLU `scale=False`



Batch-Renormalization

Problem bei Batch-Normalization:

- Nach dem Training wird anders standardisiert als während des Trainings.
- Führt evtl. zu schlechter Genauigkeit – auch auf der Trainingsmenge!

Batch-Renormalization[1] als Lösung:

- Während des Trainings wird mit den gleitenden Mittelwerten $\bar{\mu}, \bar{\sigma}$ standardisiert, sodass es keinen Unterschied zwischen der Standardisierung während des Trainings und danach gibt.
- Dazu werden Korrekturfaktoren r und s gebildet, die als konstant behandelt werden:

$$\hat{x}_d = \frac{x_d - \mu_D}{\sigma_D} \cdot r + s, \text{ mit } r = \frac{\sigma_D}{\bar{\sigma}} \text{ und } s = \frac{\mu_D - \bar{\mu}}{\bar{\sigma}}$$

Umsetzung (vor oder nach Aktivierung):

```
ann.add(BatchNormalization(renorm=True))
```
ggf. bei ReLU `scale=False`

[1] S. Ioffe. Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models. Advances in Neural Information Processing Systems 30, 2017

Initialisierung

Aktivierungsfunktionen und Zufallsinitialisierungen der Gewichte:

- Mit sigmoid konvergiert ein Netz nur sehr langsam. $\Rightarrow$ $\tanh$ ist besser.
- Für $\tanh$ sollten die Gewichte mit Glorot-Initialisierung initialisiert werden (Default).
- Die Gewichte für ReLU sollten He-gleichverteilt initialisiert werden, z. B.:

```
ann.add(Dense(50, activation='relu', kernel_initializer='he_uniform'))
```

- Rechts ist der Loss eines Netzes mit ReLU-Aktivierungen mit Glorot- und He-Initialisierung über 100 Epochen gezeigt.
- Mit der richtigen Initialisierung konvergiert das Netz schneller.
- Batch-Normalization gleicht die Initialisierung aus.