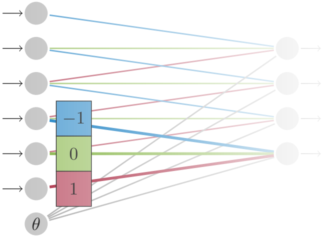
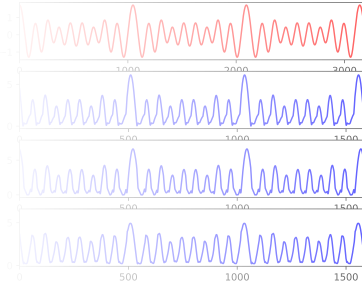
1D-CNNs

Prof. Dr. Jörg Frochte

Maschinelles Lernen

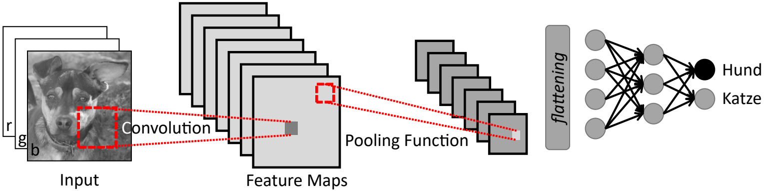




Einordnung von CNNs

- **Convolutional Neural Networks** (kurz **CNN** oder **ConvNet**) haben ihren Ursprung in den Arbeiten von Yann LeCun *Generalization and network design strategies* aus dem Jahr 1989.
- In einem CNN werden Daten, die man sinnvoll in eine Gitterstruktur bringen kann, auf eine andere Weise verarbeitet als bei den klassischen Netzen.
- Daten mit einer Gitterstruktur sind **Bilder**, aber auch **Zeitreihen** (engl. **time series**) können gelernt werden, da die Zeitreihe sich gut in einem 1D-Gitter anordnen lässt.
- Wie der Namensteil *Convolutional* andeutet kommt zu den Netz neu eine *Faltung* hinzu.
- Die grundsätzlichen Aspekte des Trainings etc. bleiben jedoch genauso erhalten, wie wir sie zuvor besprochen haben.

Grobaufbau von CNNs



- Ein MLP nutzt nicht die Lage der Datenpunkte untereinander
- Convolutional neural networks (CNN) sind hier für unstrukturierte Daten wie Bilder oft wesentlich robuster und genauer
- Einem dichten Netz ist hier eine Folge von Operationen vorgelagert, die dazu führt, dass faktisch Merkmale für unstrukturierte Daten gelernt werden
- Diese Elemente schauen wir uns nun einzeln einmal an
- Wir starten die Betrachtungen eindimensional mit Zeitreihen und machen nächstes Mal mit Bildern weiter.

Mathematische Basics

- Eine Faltung ist eine Operation zwischen zwei Funktionen, die jeweils in die reellen Zahlen abbilden.
- Nehmen wir als Beispiel einen rellwertigen Sensor oder eine Zeitreihe mit dem Wert $x(t)$.
- Eine der einfachsten Anwendungen ist ein gleitender Mittelwert.
- Wenn wir nun eine Gewichtsfunktion ω und x zusammenbringen, wie folgt zusammenbringen

$$s(t) = \int x(t-a) \cdot \omega(a) \, \mathrm{d}a \tag{1}$$

  bekommen wir eine neue Funktion s.
- Diese Operation, die durch das Aufsummieren bzw. hier Aufintegrieren von gewichteten Werten definiert ist, wird als Faltung bzw. Convolution bezeichnet.
- Kurz notiert man den Zusammenhang aus (1) mithilfe eines $*$-Symbols wie folgt:

$$s(t) = (x * \omega)(t) \tag{2}$$

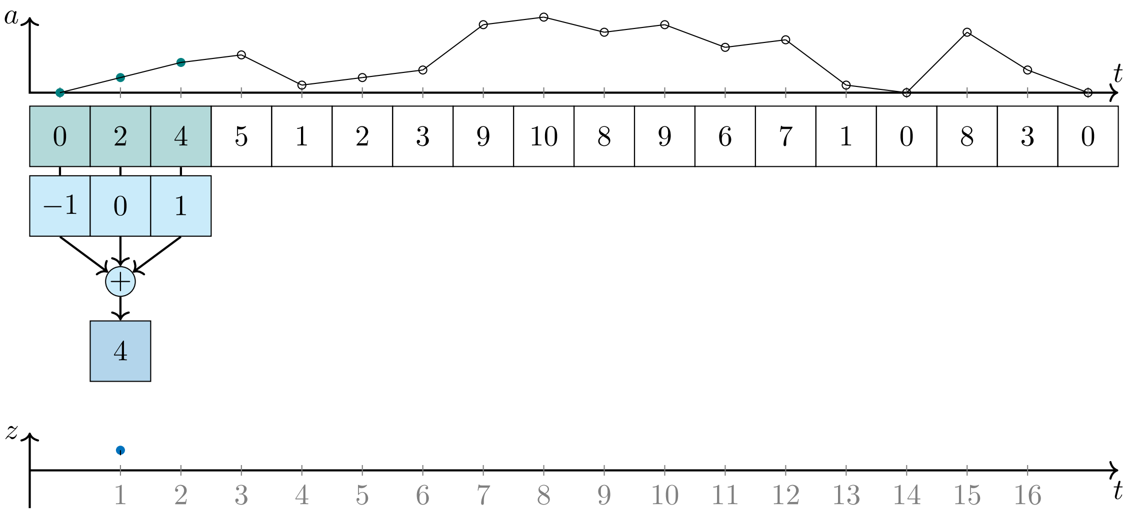
Mathematische Basics

- In den meisten technischen Umsetzungen wird nicht, wie in (1) und (2), mit Integralen gearbeitet, sondern mit Summen.
- Im Umfeld der Convolutional Neural Networks haben sich einige Begriffe um die Faltung herausgebildet. Das erste Argument – also in (2) das x – wird als **Input** I bezeichnet und das zweite – oben ω – als **Kernel** K.
- Das Ergebnis der ganzen Operation wiederum trägt den Namen **Feature Map**.
- Im Eindimensionalen fast die Gleichung (3) zusammen, was bei der Faltung auf diskret vorliegenden Daten passiert:

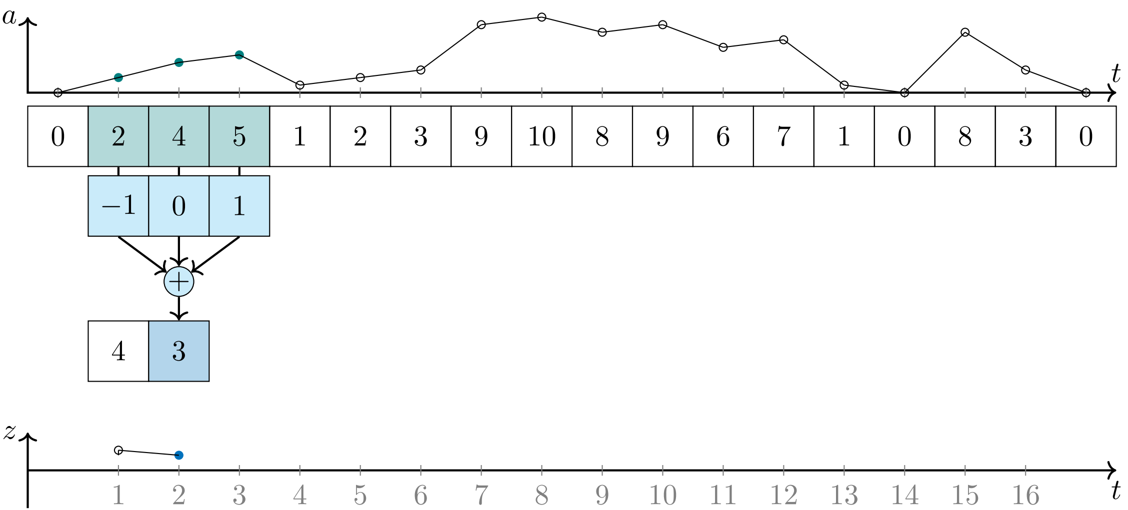
$$s[i] = \sum_{m=-(k-1)/2}^{+(k-1)/2} I[i+m] \cdot K[m] \tag{3}$$

  Die eckigen Klammern sind in Anlehnung an den Matrix-Zugriff in Python verwendet worden.
- Wir schauen uns nun einmal an, was das praktisch bedeutet.

Faltung

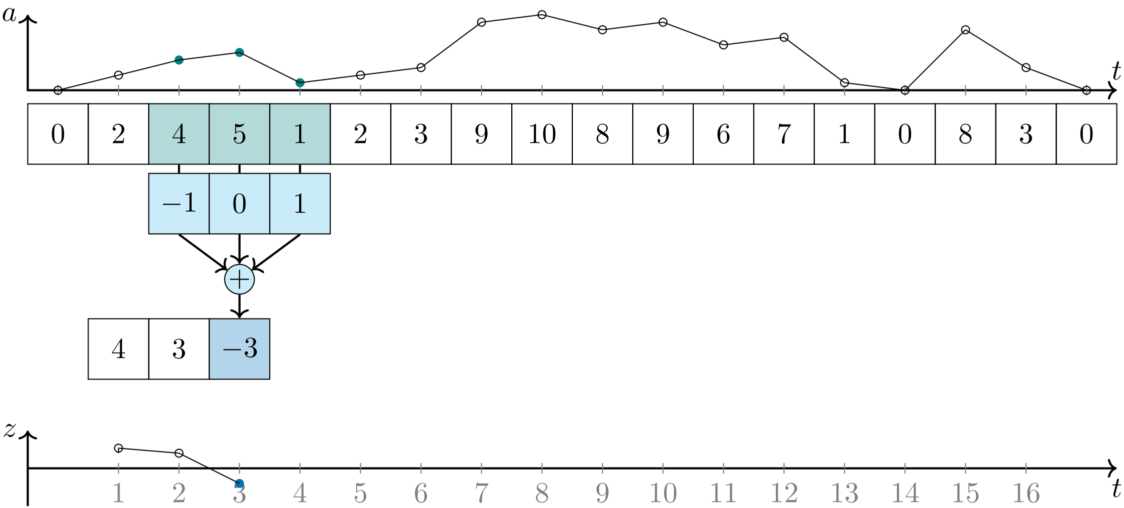

a
t
0
2
4
5
1
2
3
9
10
8
9
6
7
1
0
8
3
0
−1
0
1
+
4
z
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
t

Faltung

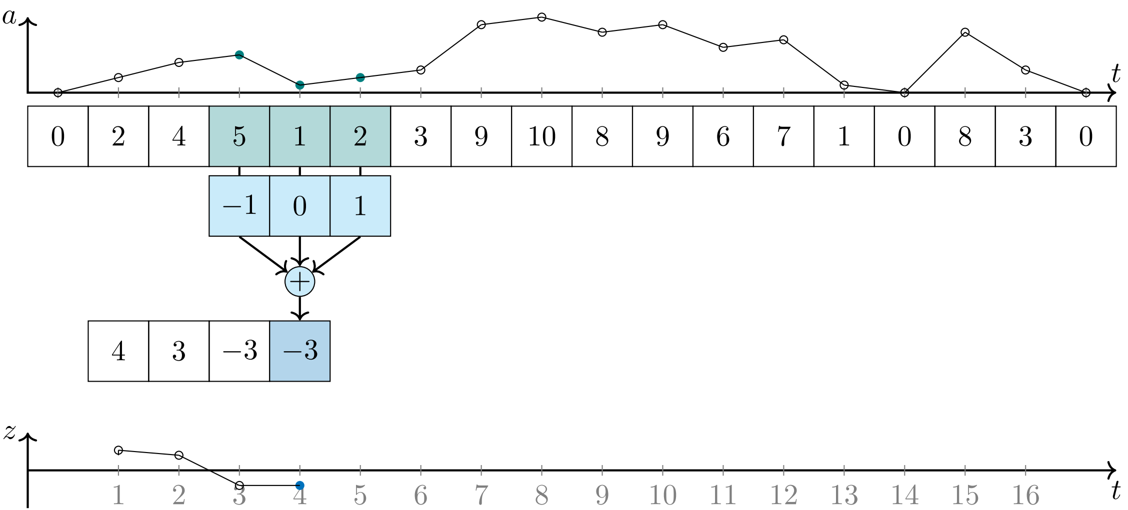


a
t
0 2 4 5 1 2 3 9 10 8 9 6 7 1 0 8 3 0
−1 0 1
+
4 3
z
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

Faltung

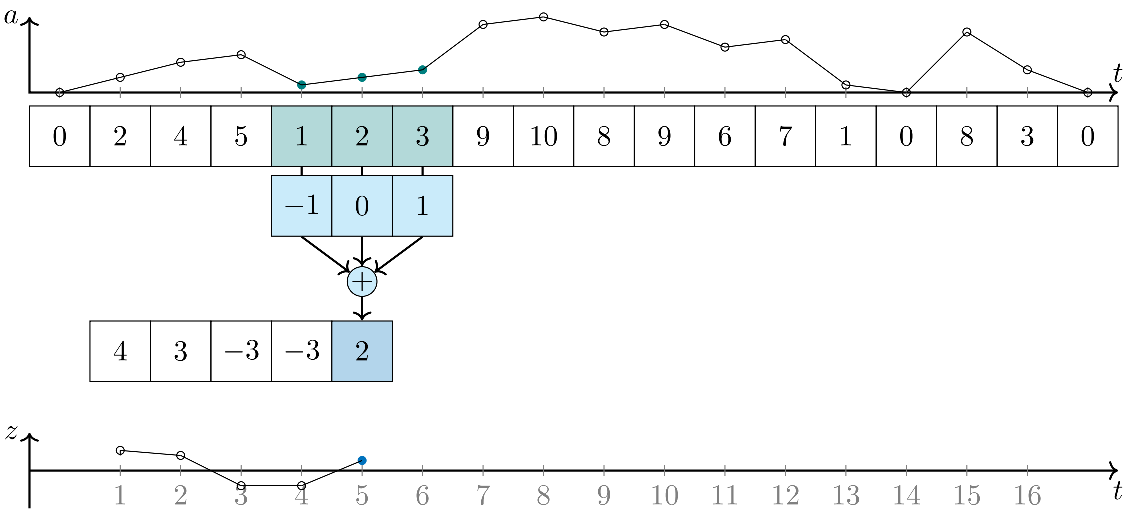

a
t
0
2
4
5
1
2
3
9
10
8
9
6
7
1
0
8
3
0
−1
0
1
4
3
−3
z
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
t

Faltung

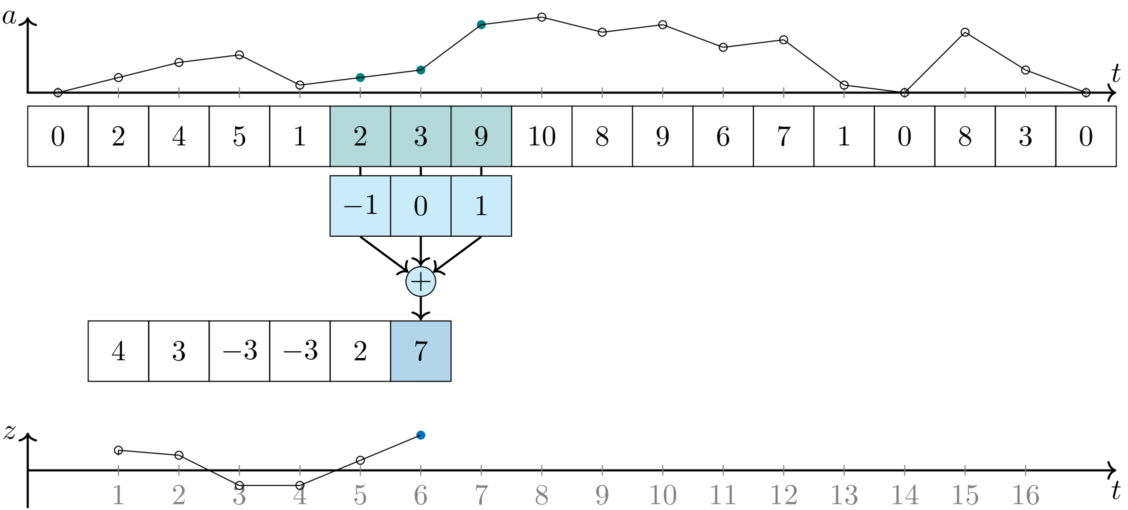


a
t
0
2
4
5
1
2
3
9
10
8
9
6
7
1
0
8
3
0
−1
0
1
4
3
−3
−3
z
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
t

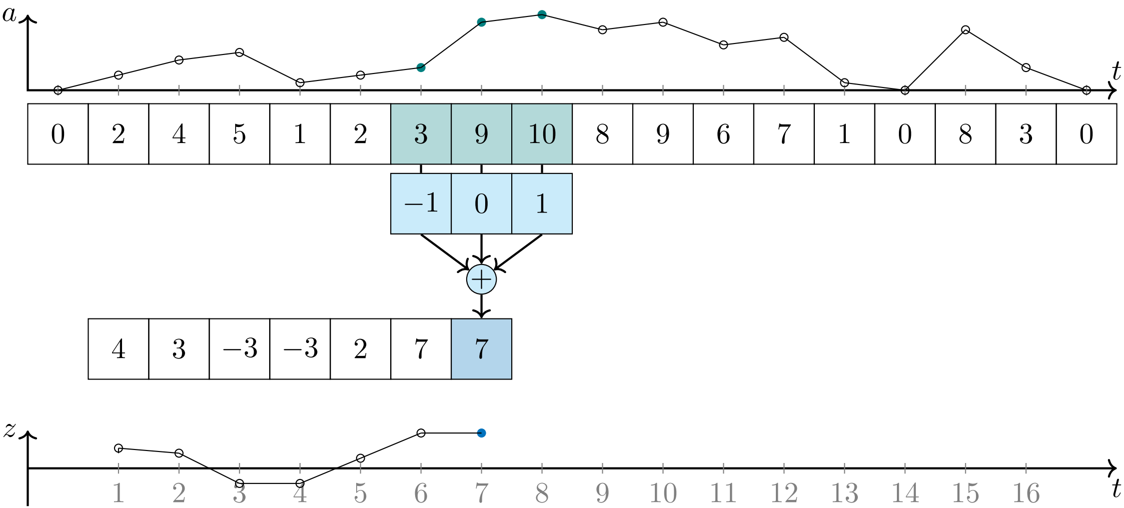
Faltung


a
t
0
2
4
5
1
2
3
9
10
8
9
6
7
1
0
8
3
0
−1
0
1
4
3
−3
−3
2
z
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

Faltung

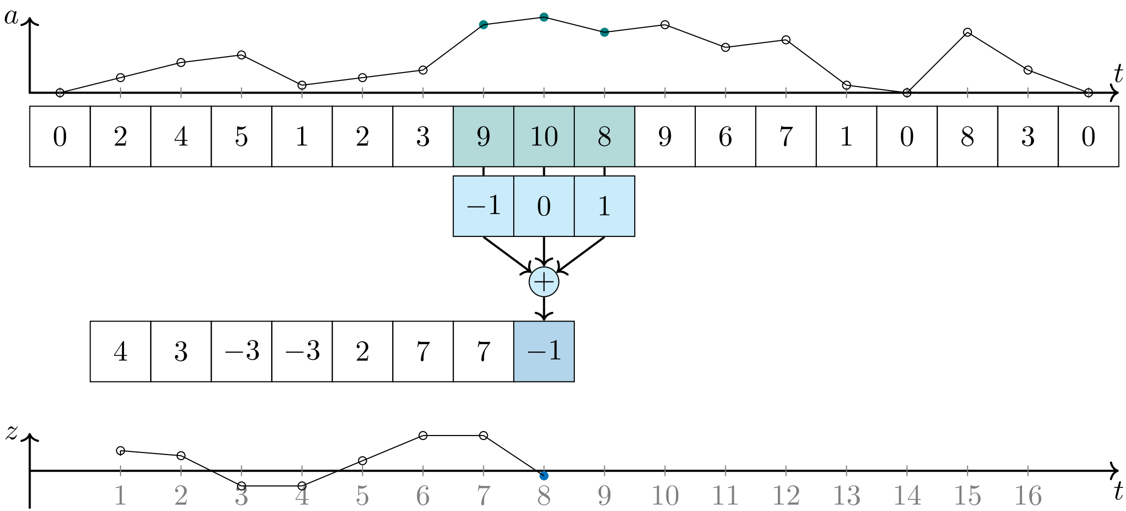

a
t
0 2 4 5 1 2 3 9 10 8 9 6 7 1 0 8 3 0
−1 0 1
4 3 −3 −3 2 7
z
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

Faltung

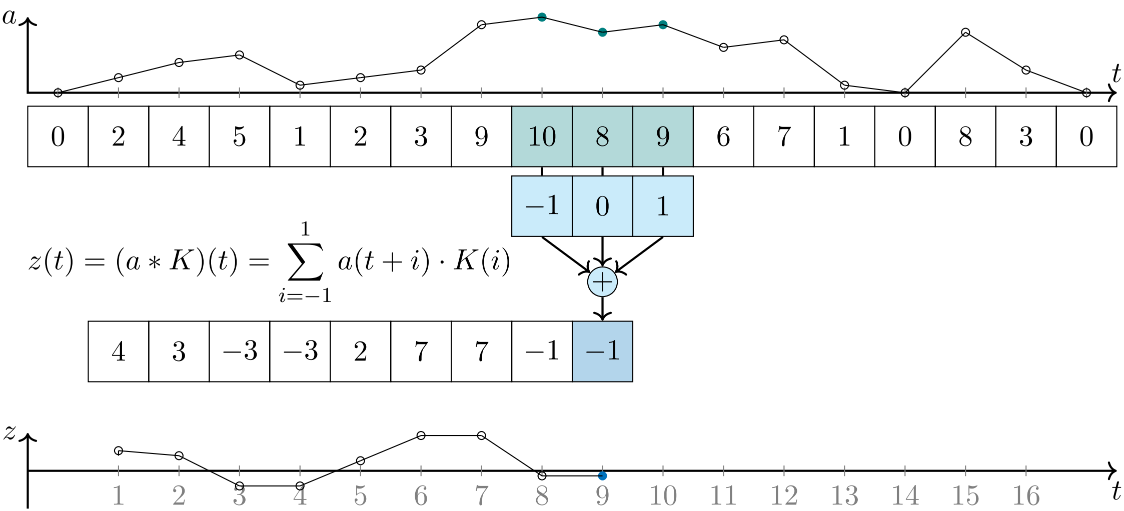


a
t
0 2 4 5 1 2 3 9 10 8 9 6 7 1 0 8 3 0
−1 0 1
+
4 3 −3 −3 2 7 7
z
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

Faltung

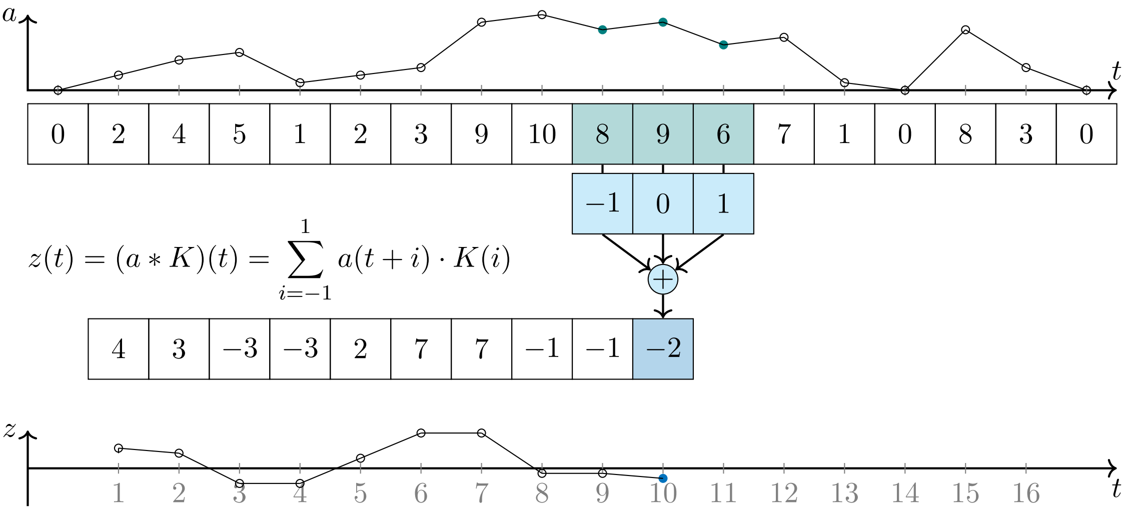

a
t
0
2
4
5
1
2
3
9
10
8
9
6
7
1
0
8
3
0
−1
0
1
4
3
−3
−3
2
7
7
−1
z
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

Faltung



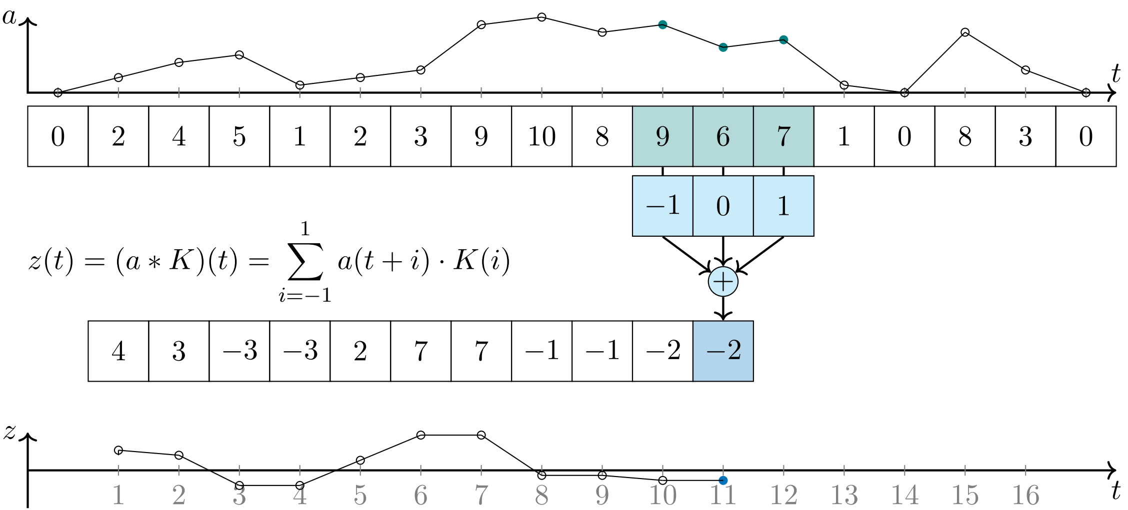
$$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$$

Faltung



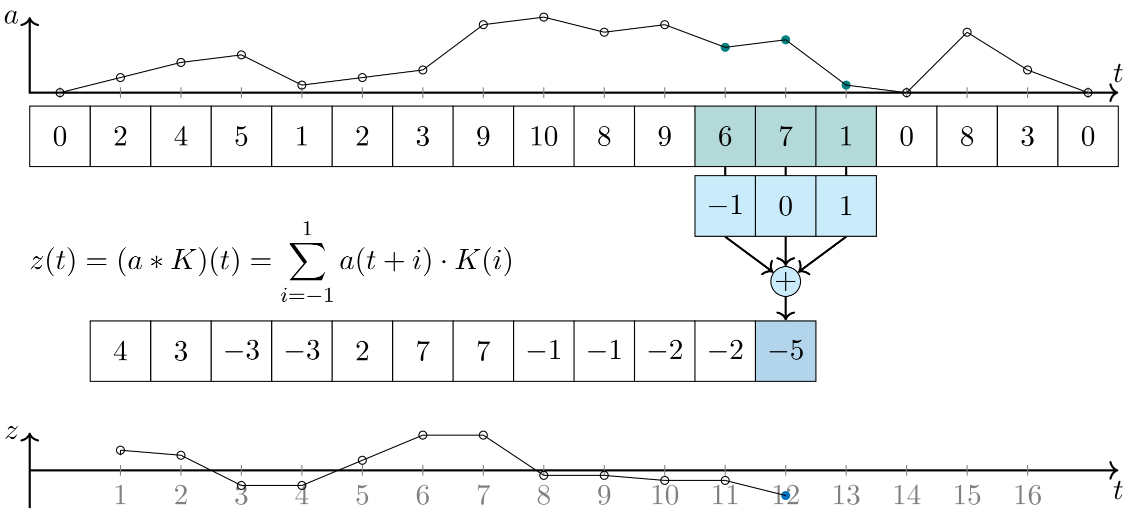
$$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$$

Faltung



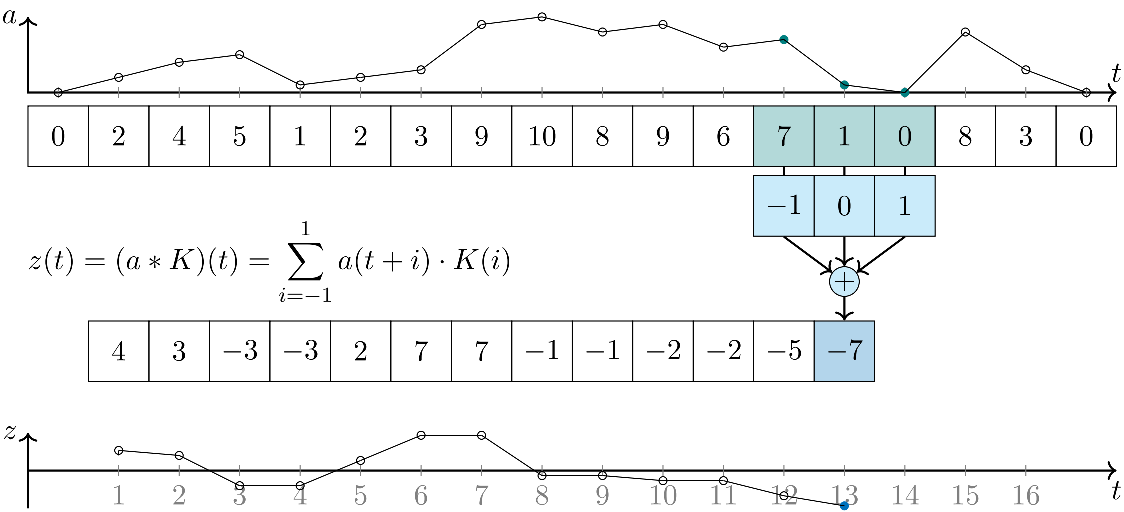
$$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$$

Faltung



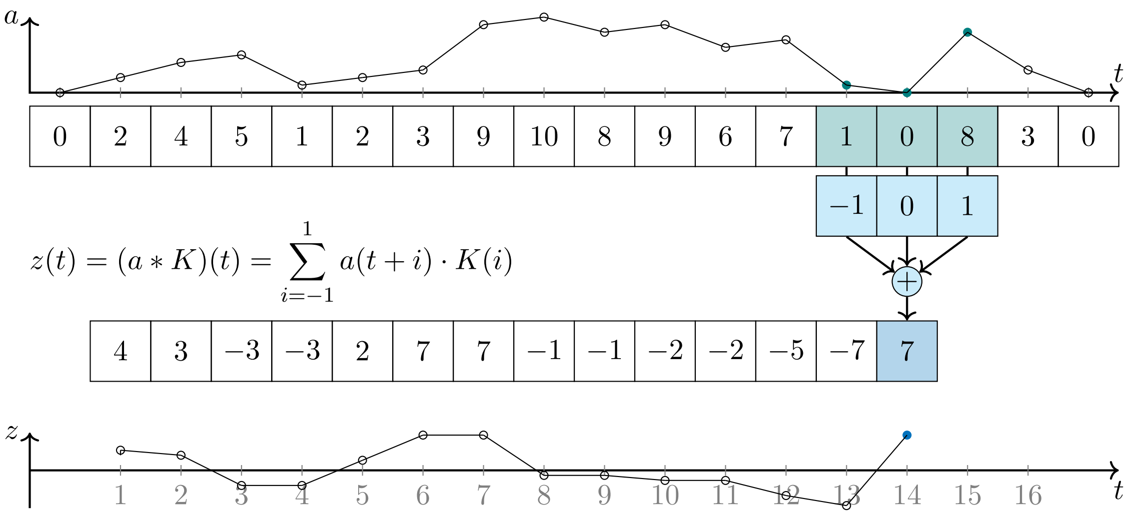
$$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$$

Faltung



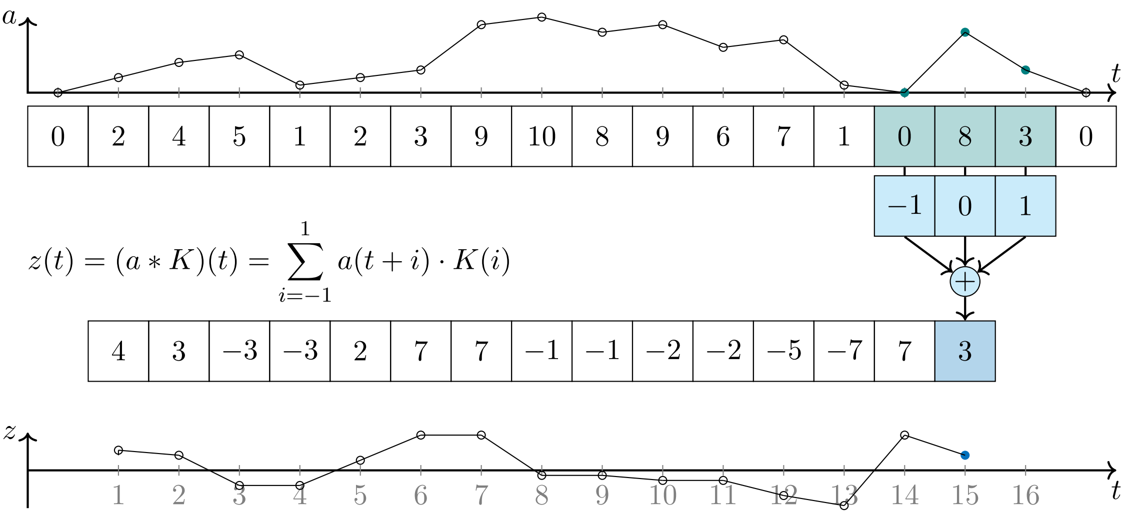
$$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$$

Faltung



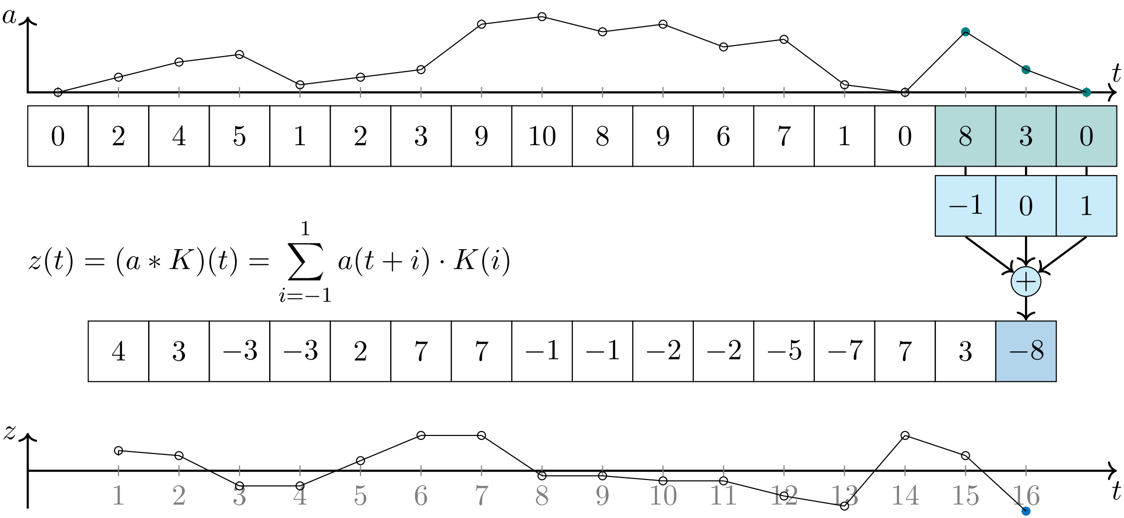
$$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$$

Faltung



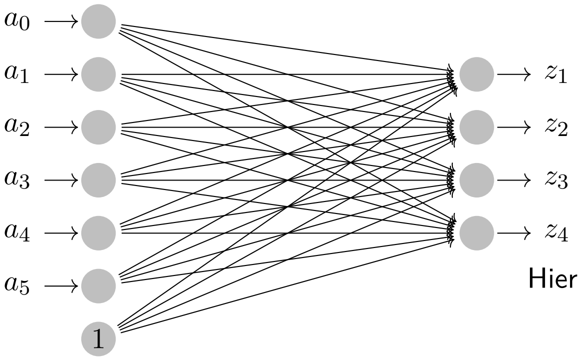
$$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$$

Faltung


a
t
0 2 4 5 1 2 3 9 10 8 9 6 7 1 0 8 3 0
−1 0 1
$z(t) = (a * K)(t) = \sum_{i=-1}^{1} a(t+i) \cdot K(i)$
4 3 −3 −3 2 7 7 −1 −1 −2 −2 −5 −7 7 3 −8
z
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

Dünnbesetzte Verbindungen und geteilte Parameter

- Bei einem klassischem dichtem Netz sind alle Neuronen einer Schicht mit allen Neuronen der folgenden Schicht verbunden.



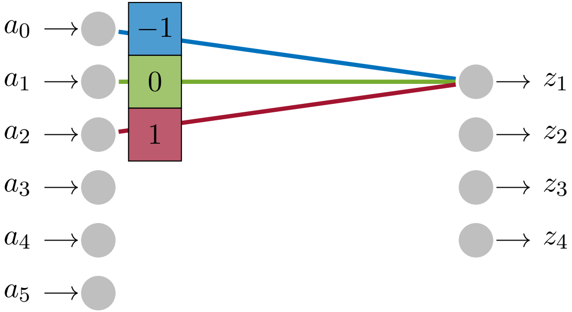
$z = W\, a + \theta$ mit W **voll besetzt**.

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} \end{pmatrix}$$

Hier sind $6 \cdot 4 + 4 = 28$ Gewichte zu trainieren.

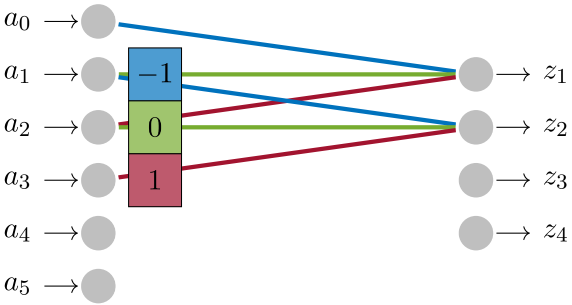
Dünnbesetzte Verbindungen und geteilte Parameter

- Bei einem klassischem dichtem Netz sind alle Neuronen einer Schicht mit allen Neuronen der folgenden Schicht verbunden.
- Bei einer Faltungsschicht (Convolutional Layer) ist das anders.



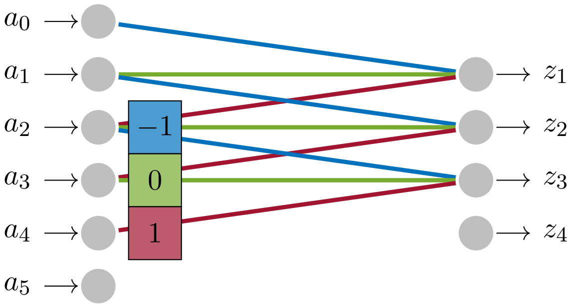
Dünnbesetzte Verbindungen und geteilte Parameter

- Bei einem klassischem dichtem Netz sind alle Neuronen einer Schicht mit allen Neuronen der folgenden Schicht verbunden.
- Bei einer Faltungsschicht (Convolutional Layer) ist das anders.



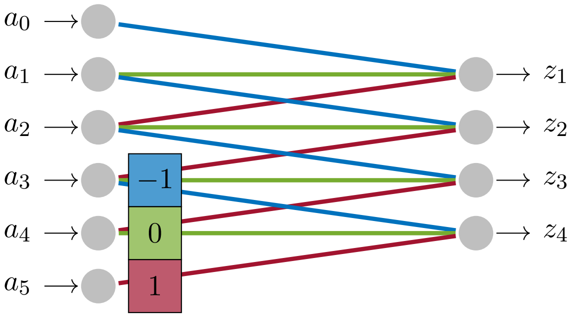
Dünnbesetzte Verbindungen und geteilte Parameter

- Bei einem klassischem dichtem Netz sind alle Neuronen einer Schicht mit allen Neuronen der folgenden Schicht verbunden.
- Bei einer Faltungsschicht (Convolutional Layer) ist das anders.



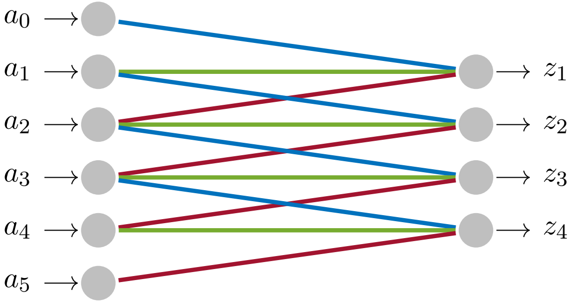
Dünnbesetzte Verbindungen und geteilte Parameter

- Bei einem klassischem dichtem Netz sind alle Neuronen einer Schicht mit allen Neuronen der folgenden Schicht verbunden.
- Bei einer Faltungsschicht (Convolutional Layer) ist das anders.



Dünnbesetzte Verbindungen und geteilte Parameter

- Bei einem klassischem dichtem Netz sind alle Neuronen einer Schicht mit allen Neuronen der folgenden Schicht verbunden.
- Bei einer Faltungsschicht (Convolutional Layer) ist das anders.
- Die Gewichtsmatrix ist dünn besetzt und enthält nur wenige verschiedene Parameter.



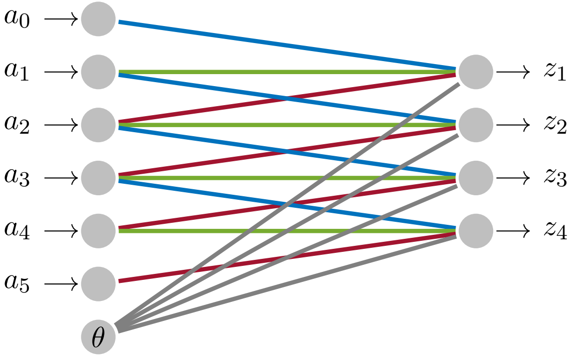
$$z = W\,a$$

$$W = \begin{pmatrix} F_{-1} & F_0 & F_1 & 0 & 0 & 0 \\ 0 & F_{-1} & F_0 & F_1 & 0 & 0 \\ 0 & 0 & F_{-1} & F_0 & F_1 & 0 \\ 0 & 0 & 0 & F_{-1} & F_0 & F_1 \end{pmatrix}$$

Hier gibt es nur die **3 Gewichte** des Faltungskernels und das Bias-Gewicht zu trainieren.

Dünnbesetzte Verbindungen und geteilte Parameter

- Bei einem klassischem dichtem Netz sind alle Neuronen einer Schicht mit allen Neuronen der folgenden Schicht verbunden.
- Bei einer Faltungsschicht (Convolutional Layer) ist das anders.
- Die Gewichtsmatrix ist dünn besetzt und enthält nur wenige verschiedene Parameter.

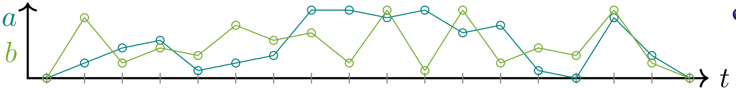


$$z = W\,a + \theta\,\mathbf{1}$$

$$W = \begin{pmatrix} F_{-1} & F_0 & F_1 & 0 & 0 & 0 \\ 0 & F_{-1} & F_0 & F_1 & 0 & 0 \\ 0 & 0 & F_{-1} & F_0 & F_1 & 0 \\ 0 & 0 & 0 & F_{-1} & F_0 & F_1 \end{pmatrix}$$

Hier gibt es nur die **3 Gewichte** des Faltungskernels und das Bias-Gewicht zu trainieren.

Faltung – mehrere Eingangssignale bzw. Kanäle

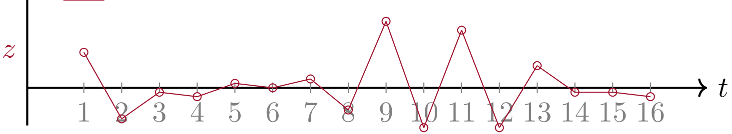


0	2	4	5	1	2	3	9	9	8	9	6	7	1	0	8	3	0
0	8	2	4	3	7	5	6	2	9	1	9	2	4	3	9	2	0

1	−1	1
−1	1	−1

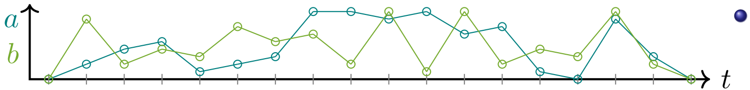
$$z(t) = (S * K)(t) = \sum_{i_c=1}^{2} \sum_{i_t=-1}^{1} S(i_c, t + i_t) \cdot K(i_c, i_t)$$

8	7	−1	−2	1	0	2	−5	15	−9	13	−9	5	−1	−1	−2



- Oft haben Eingangssignale mehrere Kanäle. Bei Bildern entspricht das R, G, B.
- Mehrere Kanäle vergrößern den Faltungskernel entsprechend.
- Aus einer Faltung entsteht nur ein einzelnes Signal.

Faltung – mehrere Eingangssignale bzw. Kanäle

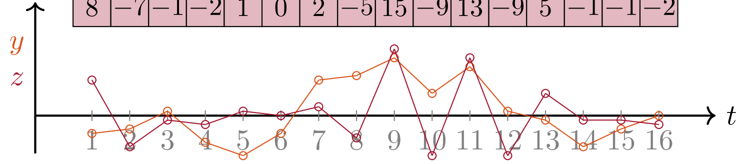


0	2	4	5	1	2	3	9	9	8	9	6	7	1	0	8	3	0
0	8	2	4	3	7	5	6	2	9	1	9	2	4	3	9	2	0

1	−1	1
−1	1	−1

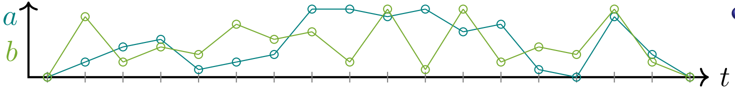
1	1	1
−1	−1	−1

−4	−3	1	−6	−9	−4	8	9	13	5	11	1	−1	−7	−3	0
8	−7	−1	−2	1	0	2	−5	15	−9	13	−9	5	−1	−1	−2



- Oft haben Eingangssignale mehrere Kanäle. Bei Bildern entspricht das R, G, B.
- Mehrere Kanäle vergrößern den Faltungskernel entsprechend.
- Aus einer Faltung entsteht nur ein einzelnes Signal.
- Es ist üblich mehrere Faltungskernel parallel zu verwenden um mehrere Signale / Kanäle zu erzeugen, welche verschiedene Arten von Features repräsentieren können.

Faltung – mehrere Eingangssignale bzw. Kanäle



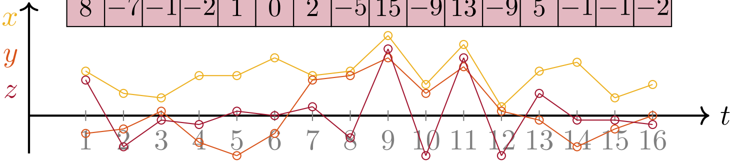
0	2	4	5	1	2	3	9	9	8	9	6	7	1	0	8	3	0
0	8	2	4	3	7	5	6	2	9	1	9	2	4	3	9	2	0

1	−1	1
−1	1	−1

1	1	1
−1	−1	−1

1	−1	1
0	1	0

10	5	4	9	9	13	9	10	18	7	16	2	10	12	4	7
−4	−3	1	−6	−9	−4	8	9	13	5	11	1	−1	−7	−3	0
8	−7	−1	−2	1	0	2	−5	15	−9	13	−9	5	−1	−1	−2



- Oft haben Eingangssignale mehrere Kanäle. Bei Bildern entspricht das R, G, B.
- Mehrere Kanäle vergrößern den Faltungskernel entsprechend.
- Aus einer Faltung entsteht nur ein einzelnes Signal.
- Es ist üblich mehrere Faltungskernel parallel zu verwenden um mehrere Signale / Kanäle zu erzeugen, welche verschiedene Arten von Features repräsentieren können.

Informationsverdichtung

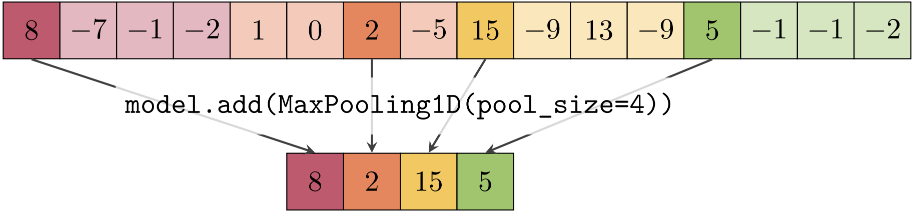
- Nach einer Faltung folgt oft ein Max-Pooling um die Informationen zu verdichten.
- Faltung und Max-Pooling erfolgen im Wechsel. Dabei wird die Anzahl Filter immer größer.
- Dazu wird eine Pool-Größe definiert, z. B. mit 4 würde dann aus jeweils 4 Zeitschritten der größte Wert ausgewählt.
- Es gibt auch noch Average-Pooling, welches den Mittelwert bildet. Max-Pooling ist aber vorteilhaft, da es ja um die Aktivierung von Neuronen geht und dabei der größte Wert am wichtigsten ist.

8	-7	-1	-2	1	0	2	-5	15	-9	13	-9	5	-1	-1	-2

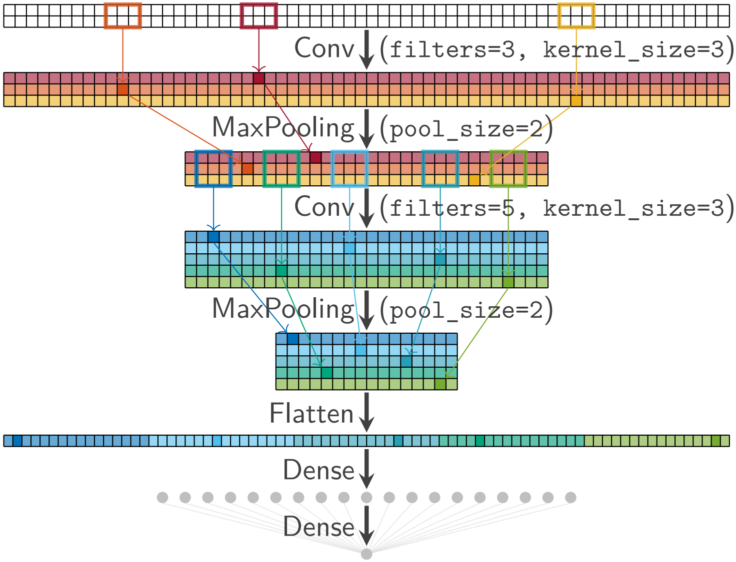
```
model.add(MaxPooling1D(pool_size=4))
```

Informationsverdichtung

- Nach einer Faltung folgt oft ein Max-Pooling um die Informationen zu verdichten.
- Faltung und Max-Pooling erfolgen im Wechsel. Dabei wird die Anzahl Filter immer größer.
- Dazu wird eine Pool-Größe definiert, z. B. mit 4 würde dann aus jeweils 4 Zeitschritten der größte Wert ausgewählt.
- Es gibt auch noch Average-Pooling, welches den Mittelwert bildet. Max-Pooling ist aber vorteilhaft, da es ja um die Aktivierung von Neuronen geht und dabei der größte Wert am wichtigsten ist.



- Wenn die Informationsdichte hoch genug ist und die Features ausdrucksstark genug sind, folgt ein dichtes neuronales Netz um die eigentliche Aufgabe zu erledigen.
- Die Werte nach dem letzten Max-Pooling sind die Inputs für das dichte Netz.
- Um das Netz anzuschließen, werden die "Feature-Maps" hintereinander als 1D-Vektor geschrieben ("Flatten"). Jedes Pixel wird jeweils mit jedem Neuron verbunden.



Beispielanwendung Klassifikation – Künstliche Signale erzeugen

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)

nrows = 10000
nsteps = 5000
timeseries = np.zeros((nrows, nsteps, 1))
t = np.linspace(0, 30, num=nsteps)
for i in range(nrows):
    a = 2*(np.random.rand(11) - 0.5)
    b = 2*(np.random.rand(11) - 0.5)
    timeseries[i,:,0] = a[0]/2
    for k in range(1, 11):
        timeseries[i,:,0] += a[k] * np.cos(t*k)
        timeseries[i,:,0] += b[k] * np.sin(t*k)
    timeseries[i,:,0] = 2*timeseries[i,:,0]/np.max(np.abs(timeseries[i,:,0]))
```

$$\frac{a_0}{2} + \sum_{k=1}^{10} \left(a_k \cos\left(kt\right) + b_k \sin\left(kt\right)\right)$$

Beispielanwendung Klassifikation – Störungen hinzufügen

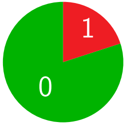
Wir wählen zufällig 20% der Signale aus, die eine Störung erhalten sollen.

```

Y = np.zeros(nrows)
yTrue = np.random.choice(nrows, int(nrows*0.2), replace=False)
Y[yTrue] = 1
stoerung = np.array([0.1, 0.2, 0.4, 0.4, 0.3,0.4,0.4,0.2,0,0,0,0,0,0,0,0,
                    -0.1,-0.2,-0.4,-0.2,-0.2,-0.4,-0.2,-0.1] * 2)
for i in yTrue:
    xpos = np.random.randint(0, nsteps - len(stoerung))
    timeseries[i, xpos:xpos+len(stoerung), 0] += stoerung

TrainSet = np.random.choice(timeseries.shape[0],
                            int(timeseries.shape[0]*0.80), replace=False)
XTrain   = timeseries[TrainSet,:]; YTrain   = Y[TrainSet]
TestSet  = np.delete(np.arange(len(Y)), TrainSet)
XTest    = timeseries[TestSet,:]; YTest    = Y[TestSet]
```

Beispielanwendung Klassifikation – Daten gewichten

- Die Samples mit Störnung sind klar in der Minderheit.
- Ohne Gegenmaßnahmen würde das Netz oft lernen grundsätzlich eine 0 vorherzusagen und würde damit eine Genauigkeit von 80% erreichen.
- Daher gewichten wir die Gradienten eines Mini-Batches bei der Optimierung der Klasse 0 mit 0.25 und die der Klasse 1 mit 1, weil es $4\times$ so viele Samples der Klasse 0 gibt, wie die der Klasse 1:

$$-\nabla \sum_i w_i \, y_i \, \log(\hat{y}(\mathbf{x}_i, \mathbf{W})) \text{ mit } w_i = \begin{cases} 0.25 & \text{für } y_i = 0 \\ 1 & \text{für } y_i = 1 \end{cases}$$




```
sampleWeight = np.ones_like(YTrain)
sampleWeight[YTrain==0] = 0.3
```

- Wir kommen jetzt zum Aufbau des Netzes
- Die Anzahl der Filter steigt hier nicht, sondern nimmt ab – ungewöhnlich, funktioniert hier aber besser und macht Effekte transparent.

Beispielanwendung Klassifikation – Aufbau des 1D-CNN

```python

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv1D,MaxPooling1D
from tensorflow.compat.v1.random import set_random_seed
set_random_seed(42)

model = Sequential()
model.add(Conv1D(8, kernel_size=10, activation='relu', use_bias=False,
                 input_shape=(timeseries.shape[1],1)))
model.add(MaxPooling1D(pool_size=4))
model.add(Conv1D(6, kernel_size=8, activation='relu', use_bias=False))
model.add(MaxPooling1D(pool_size=4))
model.add(Conv1D(1, kernel_size=4, activation='relu', use_bias=False))
model.add(MaxPooling1D(pool_size=4))
model.add(Flatten())
model.add(Dense(10, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

Beispielanwendung Klassifikation – Abschluss des Listings und Genauigkeit

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(XTrain, YTrain, epochs=30, verbose=True,
                    sample_weight=sampleWeight)
model.summary()
print(model.evaluate(XTest, YTest, verbose=False))
```

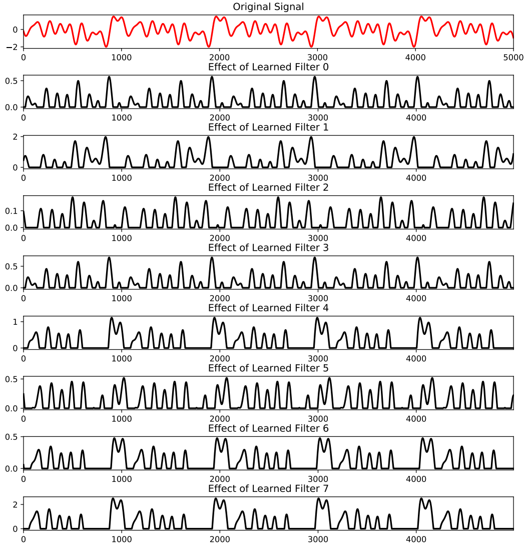
- Die Genauigkeit liegt bei 99.96% auf der Trainingsmenge und bei 99.95% auf der Testmenge.
- Hier wurde eine Variante mit Kreuzentropie verwendet. Mit einem Ausgabeneuron muss dann `binary_crossentropy` verwendet werden.

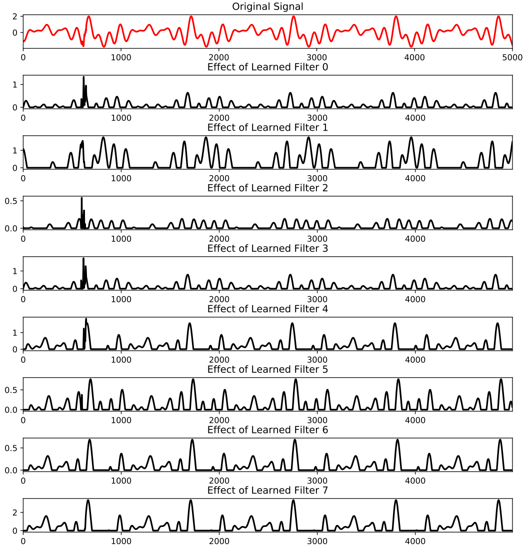
Filtergrößen

- Auf die Werte der Filter kann man nach dem Training wie folgt zugreifen:
```
kernelLayer0 = model.layers[0].get_weights()[0]
kernelLayer2 = model.layers[2].get_weights()[0]
```
- Die Dimensionen von `kernelLayer0` lauten (10, 1, 6) bzw. von `kernelLayer2` (8, 6, 3).
- Das Signal hat 1 Kanal, wir haben 6 Filter mit jeweils 10 Gewichten bestellt.
- Bei der zweiten Faltung arbeitet jeder Filter nun auf allen zuvor generierten Feature Maps. Da wir 6 davon generiert haben, ändert sich die Dimension hier von 1 zu 6.
- Die Anzahl der Filter aus dem ersten Schritt entspricht also der Tiefe des neuen Kernels.
- Es wird also bei der 1D-Faltung nun ein Tensor angewendet. Sei die Kernelbreite k_w, die Anzahl Signale im vorigen Layer bzw. Tiefe d, die Anzahl Feature Maps m, dann gilt:

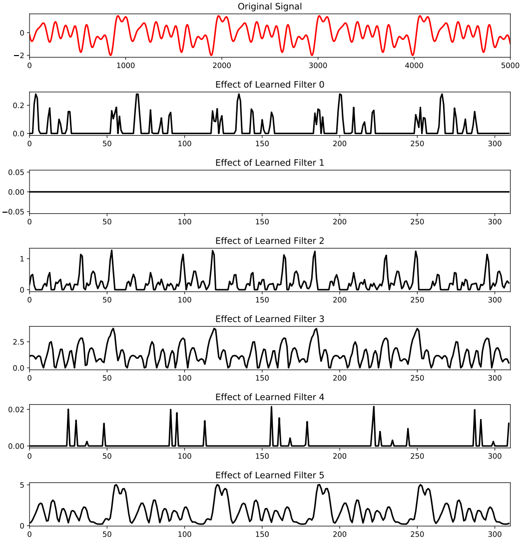
$$I'_j(x) = \sum_{i_x=1-\lceil \frac{k_w}{2} \rceil}^{\lfloor \frac{k_w}{2} \rfloor} \sum_{i_c=1}^{d} I(x+i_x, i_c) \cdot F_j(i_x, i_c) \;\forall\; j = 1, \ldots, m$$

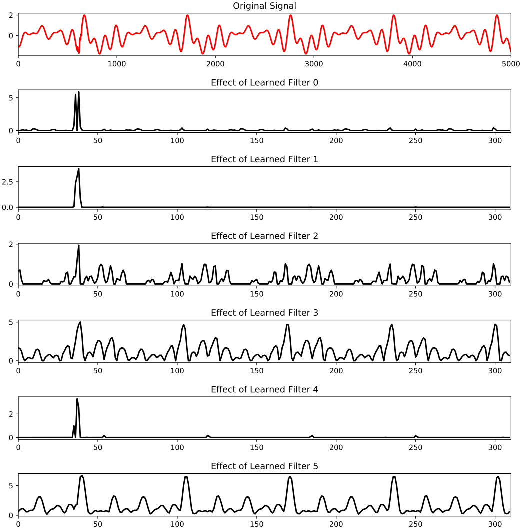
1. Faltungslayer : Ein Signal der Klasse 0 (l.) & eines der Klasse 1 (r.)



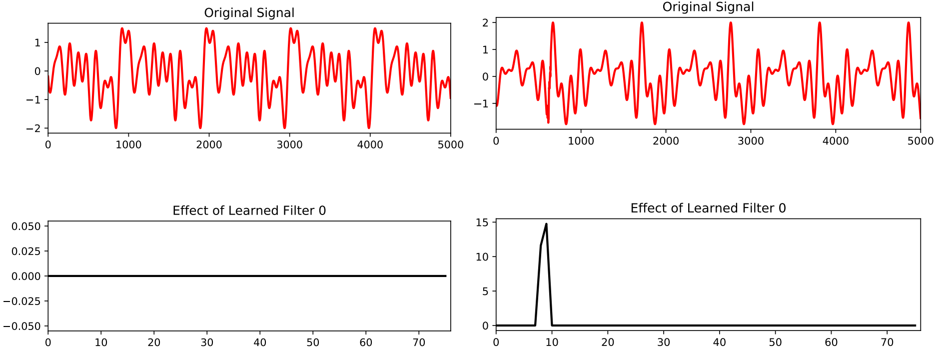


4. Faltungslayer : Ein Signal der Klasse 0 (l.) & eines der Klasse 1 (r.)





Gefaltete Signale nach letzten Conv-Layer



- Am Schluss entsteht ein Signal, das stark bei der Position der Störung ausschlägt.
- Durch das Max-Pooling sind für das Signal natürlich weniger Rasterpunkte vorhanden.
- Die Größe des Kernels muss nicht mit der Breite der Störung übereinstimmen, aber globale Eigenschaften, welche nicht mit lokalen korrelieren, sind schwer zu entdecken.