Typen von neuronalen Netzen Teil I: Autoencoder und Convolutional Neural Networks

Elias Saalmann

Universität Leipzig

12. Februar 2018

Inhalt

Wiederholung

- Neuronale Netze zur Klassifikation nicht-linearer Daten
- Training der Netze als Minimierungsproblem einer Fehlerfunktion
- Finden eines Minimums über Gradienten bzw. mit Backpropagation Algorithmus
- Bislang nur Supervised Learning mit Labeled Data betrachtet
- Bislang nur Fully-Connected-Netze betrachtet
- Bislang nur Feedforward-Netze betrachtet

Definition

Sei

- Eingaberaum X gleich Ausgaberaum mit Dimension d
- Zwischenraum Y mit Dimension $p \ll d$

Gesucht

- Encoder-Funktion $f : X \to F$
- Decoder-Funktion $g : F \to X$

welche für alle $x \in X$ den Fehler

$$|x - (g \circ f)(x)|$$

minimieren.

Beispiel: Linearer Autoencoder

Wähle:

- $f(x^{(i)}) = W_1 x^{(i)} + b_1$
- $g(x^{(i)}) = W_2 x^{(i)} + b_2$

Optimierungsziel:

$$
\begin{aligned}
J(W_1, b_1, W_2, b_2) &= \sum_{i=1}^{m} (x^{(i)} - (g \circ f)(x^{(i)}))^2 \\
&= \sum_{i=1}^{m} (x^{(i)} - (W_2(W_1 x^{(i)} + b_1) + b_2))^2 \\
&\rightarrow min
\end{aligned}
$$

$\rightarrow$ Neuronales Netz mit 1 Hidden Layer

Autoencoder als neuronales Netz

- Funktion kann als KNN mit d Eingabe- und Ausgabeneuronen modelliert werden
- Mindestens 1 Hidden Layer (z.B. mit p Knoten)
- Hyperparameter frei wählbar (z.B. Aktivierungsfunktion)
- Autoencoder *lernt* eine (niedrig-dimensionale) Codierung der Eingabedaten
- Dimensionsreduzierung vergleichbar mit Principal Component Analysis (PCA)
- **Unsupervised Learning**

Autoencoder Beispiel

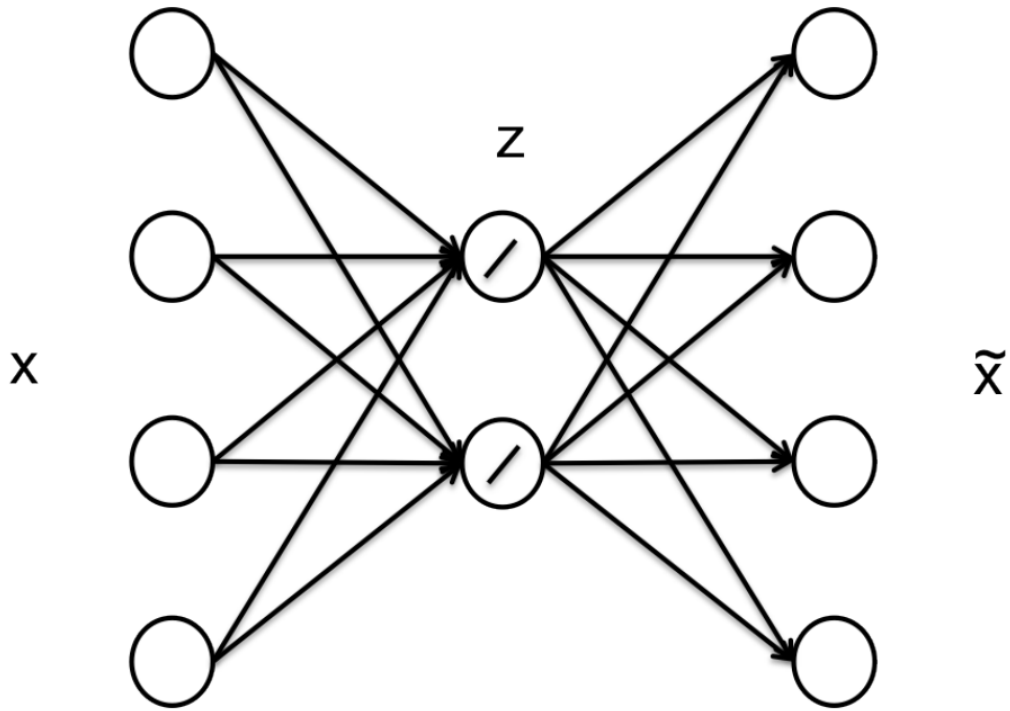

Figure: Beispiel-Autoencoder KNN [L15]

Dimensionsreduzierung für Kompression

Die niedrigdimensionale Codierung kann als Kompression verwendet werden, um z.B. Datentraffic einer App zu verringern:

Beispiel

Smartphone-App möchte $\{x_1, x_2, ..., x_m\}$ an Server senden

1. Codiere $\{x_1, x_2, ..., x_m\}$ zu $\{z_1, z_2, ..., z_p\}$ mit $p \ll m$ auf dem Smartphone
2. Sende $\{z_1, z_2, ..., z_p\}$ an den Server
3. Decodiere $\{z_1, z_2, ..., z_p\}$ zu $\{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_m\}$

Dimensionsreduzierung für Visualisierung

Ist die Codierung 2- oder 3-dimensional, können hochdimensionale Daten durch die Codierung visualisiert werden:

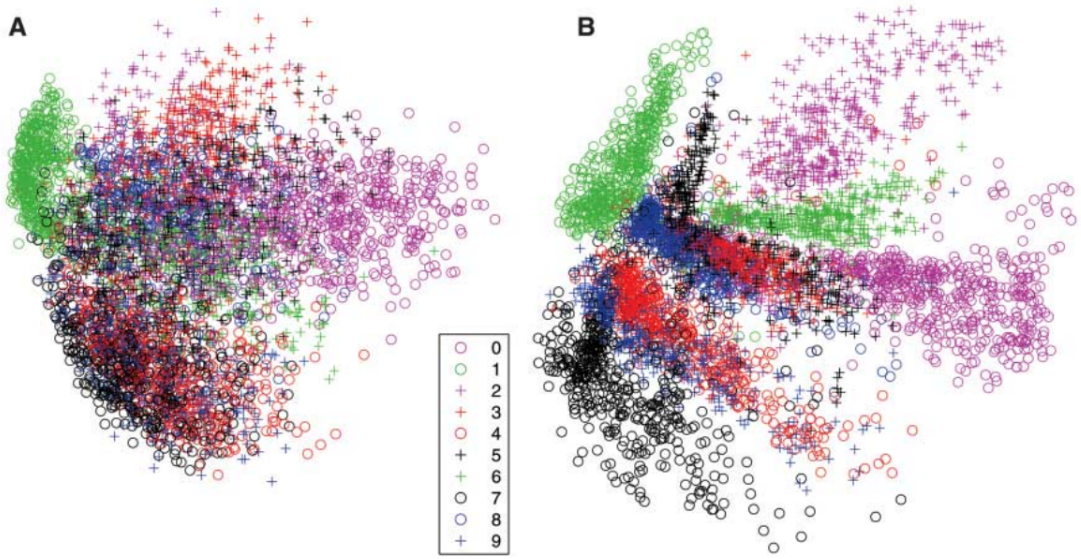

Figure: 2D-Codierung von handgeschriebenen Zahlen (MNIST):
(A): PCA
(B): 2000-500-250-125-2 Autoencoder [HS06]

Dimensionsreduzierung im Kontext der Gesichtserkennung

Figure: Rekonstruktion von codierten (30-dimensionalen) Bildern (Olivetti Gesichtsdatenbank):
(oben): Original
(mitte): Rekonstruktion mit Autoencoder
(unten): Rekonstruktion mit PCA [HS06]

Autoencoder als KNN-Initialisierung

Problem

KNN-Gewichte müssen vor Backpropagation initialisiert werden.
Zufallsinitialisierung ist nicht immer optimal.

KNN-Training mit Autoencodern

1. **Pre-Training** mit Autoencodern, jeweils 1 Layer, unsupervised
2. **Fine-Tuning 1** Training des letzten Layers, supervised
3. **Fine-Tuning 2** Training des gesamten KNN mit Backpropagation, supervised

$\rightarrow$ hat sich als vielversprechend erwiesen [EBC10]

Pre-Training mit Autoencodern

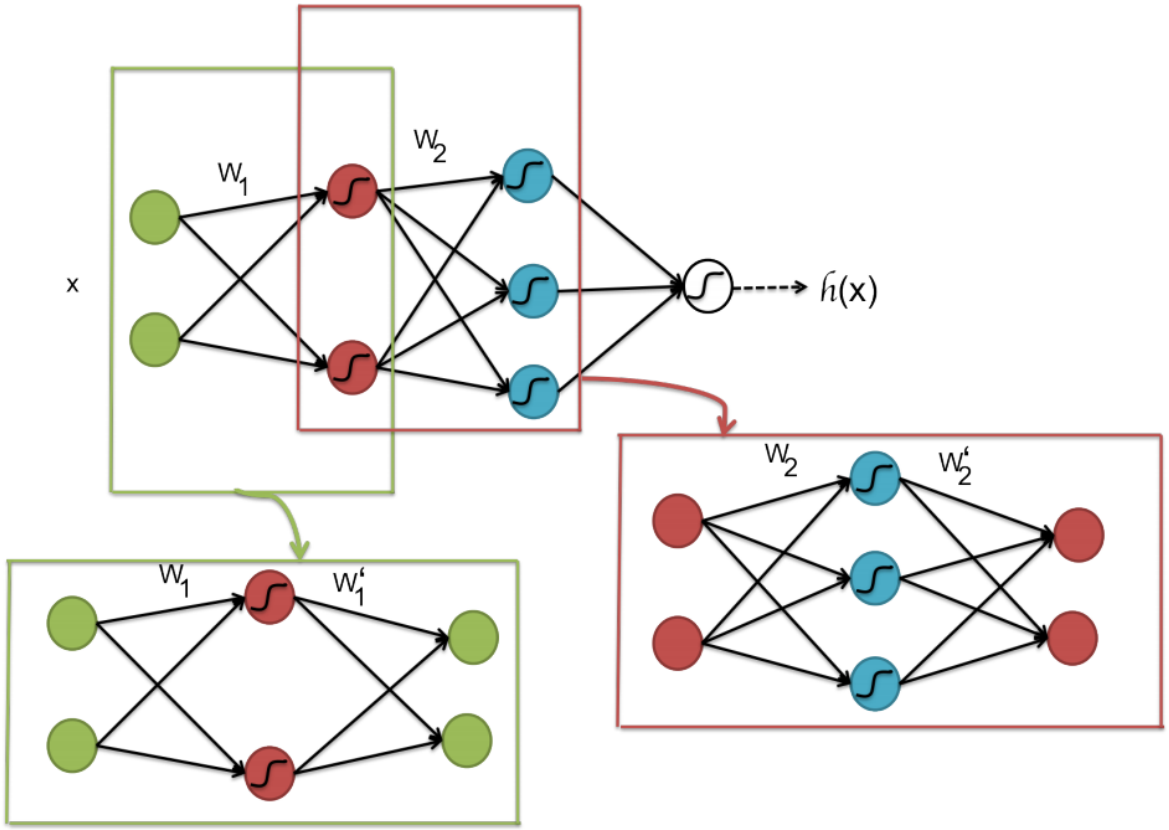

Figure: Autoencoder als Initialisierungsmethode [L15]

Motivation

Hochdimensionale natürliche Daten (z.B. Sprache, Fotos, ...)

- Fully-connectedness erzeugt große Mengen an Parametern
- Hoher Trainings-Aufwand
- Gefahr des Overfitting (Anpassung an das Trainings-Set)

$\rightarrow$ lokal verbundene neuronale Netze

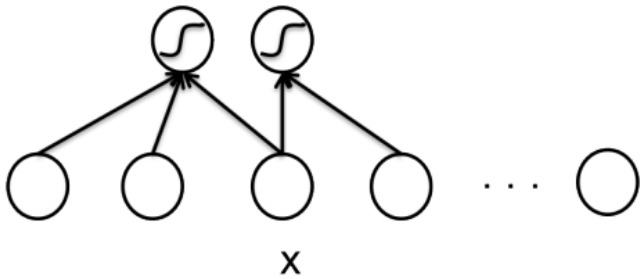

Figure: Lokal verbundenes KNN [L15]

Definition

Convolutional Neural Networks sind teilweise lokal verbundene KNN, die meistens aus folgenden Layern bestehen:

1. Convolution Layer

Angelehnt an die mathematische Faltung von Eingabesignalen

2. Pooling Layer

Berechnet Maximum oder Durchschnitt

3. Fully Connected Layer

Schließt die Berechnungen aus Convolution und Pooling als klassiche Klassifikation ab

Faltung (Convolution)

Wichtiger Operator in der Bild- und Signalverarbeitung:

Faltungs- bzw. Convolution-Operator

- Eingabevektor/ Eingabematrix A
- Kernelvektor / Kernelmatrix (auch *Filter* genannt) F
- Operation $A * F$ berechnet für jede Überlappung des Filters mit der Eingabematrix die Summe aller Element-Produkte

1D-Faltung (Convolution)

$$A = \boxed{10}\boxed{50}\boxed{60}\boxed{10}\boxed{20}\boxed{40}\boxed{30}$$

$$*$$

$$F = \boxed{1/3}\boxed{1/3}\boxed{1/3}$$

$$=$$

$$\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$$

1D-Faltung (Convolution)

$$A = \boxed{10}\boxed{50}\boxed{60}\boxed{10}\boxed{20}\boxed{40}\boxed{30}$$

$$*$$

$$F = \boxed{1/3}\boxed{1/3}\boxed{1/3}$$

$$=$$

$$10 \cdot \tfrac{1}{3} + 50 \cdot \tfrac{1}{3} + 60 \cdot \tfrac{1}{3} = 40$$

$$\boxed{40}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$$

1D-Faltung (Convolution)

$$A = \boxed{10}\boxed{50}\boxed{60}\boxed{10}\boxed{20}\boxed{40}\boxed{30}$$

$$*$$

$$F = \boxed{1/3}\boxed{1/3}\boxed{1/3}$$

$$=$$

$$\boxed{40}\boxed{40}\boxed{30}\boxed{20}\boxed{30}$$

2D-Faltung (Convolution): Kantenerkennung

$$\underbrace{\begin{vmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{vmatrix}}_{\text{Eingabematrix A}} * \underbrace{\begin{vmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{vmatrix}}_{\text{Filter F}} = \begin{vmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{vmatrix}$$

2D-Faltung (Convolution): Kantenerkennung

$$\begin{vmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{vmatrix} * \begin{vmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{vmatrix} = \begin{vmatrix} 0 & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{vmatrix}$$

$$10 \cdot 1 + 10 \cdot 1 + 10 \cdot 1 + 10 \cdot 0 + 10 \cdot 0 + 10 \cdot 0 + 10 \cdot -1 + 10 \cdot -1 + 10 \cdot -1 = 0$$

2D-Faltung (Convolution): Kantenerkennung

$$\begin{vmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{vmatrix} * \begin{vmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{vmatrix} = \begin{vmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & 30 & ? \\ ? & ? & ? & ? \end{vmatrix}$$

$$10 \cdot 1 + 10 \cdot 1 + 10 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot -1 + 0 \cdot -1 + 0 \cdot -1 = 30$$

2D-Faltung (Convolution): Kantenerkennung

$$\begin{vmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{vmatrix} * \underbrace{\begin{vmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{vmatrix}}_{\text{Prewitt-Operator}} = \begin{vmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{vmatrix}$$

Faltungs-Operator: Erweiterungen

Padding

- Problem: Bisher war das Ergebnis stets kleiner als die Eingabe
- Hinzufügen eines mit 0en gefüllten Randes ermöglicht ein Beibehalten der Größe

Strided Convolution

- Problem: Manchmal ist eine *Kompression* bzw. *Vergröberung* erwünscht
- Größere Abstände bzw. Sprünge des Filters auf der Eingabe führen zu einer kleineren Ausgabe

Convolution-Layer

Grundprinzip von Convolutional Neural Networks

- Die Filter-Matrix ist zunächst unbekannt
- Das KNN soll die Werte der Filter-Matrix je nach Anwendung lernen

$$F = \begin{vmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{vmatrix}$$

Die Werte/Gewichte der Matrix gelten für die gesamte Eingabe, d.h. es müssen nur wenige Parameter gelernt werden.

Convolution-Layer (cont'd)

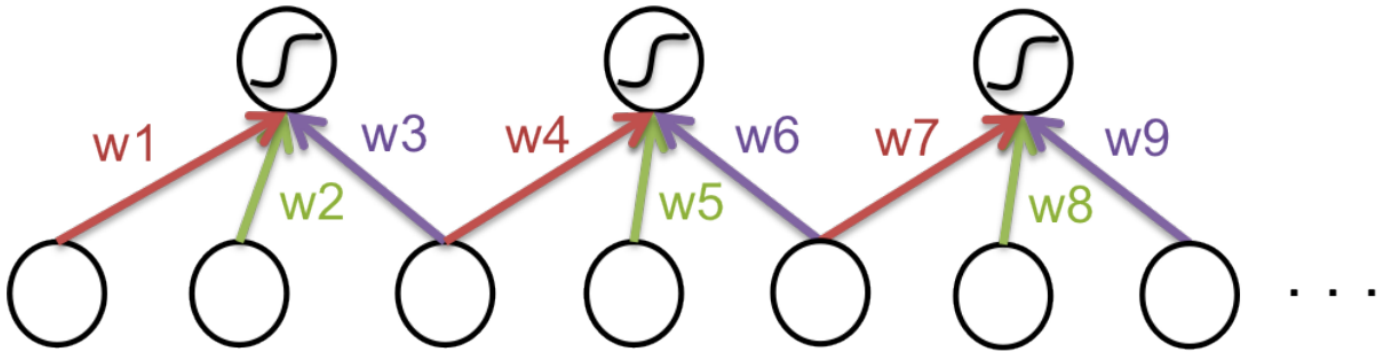

Figure: 1D-Faltung als KNN [L15]

Es soll gelten:

- $w_1 = w_4 = w_7$
- $w_2 = w_5 = w_8$
- $w_3 = w_6 = w_9$

$\rightarrow$ die Gewichte werden 'geteilt', Anpassung Backpropagation

Convolution-Layer: Anpassungen an Backpropagation

Vorwärtsdurchlauf

Betrachte alle w_i mit ihren konkreten Werten

Rückwärtsdurchlauf

Berechne für alle w_i deren Gradienten

Aktualisierung der w_i

Passe alle zusammengehörenden w_i gemäß des Durchschnitts aller zugehörigen berechneten Gradienten an

Pooling-Layer (Subsampling-Layer)

Motivation

- Oftmals genügen *grobe* Informationen
- Verschiebung der Eingabedaten *(Translationsinvarianz)*

Beispiel Maximum-Pooling

$$\begin{vmatrix} 1 & 3 & 2 & 1 \\ 2 & 9 & 1 & 1 \\ 1 & 3 & 2 & 3 \\ 5 & 6 & 1 & 2 \end{vmatrix} \rightarrow_{MAX} \begin{vmatrix} 9 & 2 \\ 6 & 3 \end{vmatrix}$$

Pooling-Layer

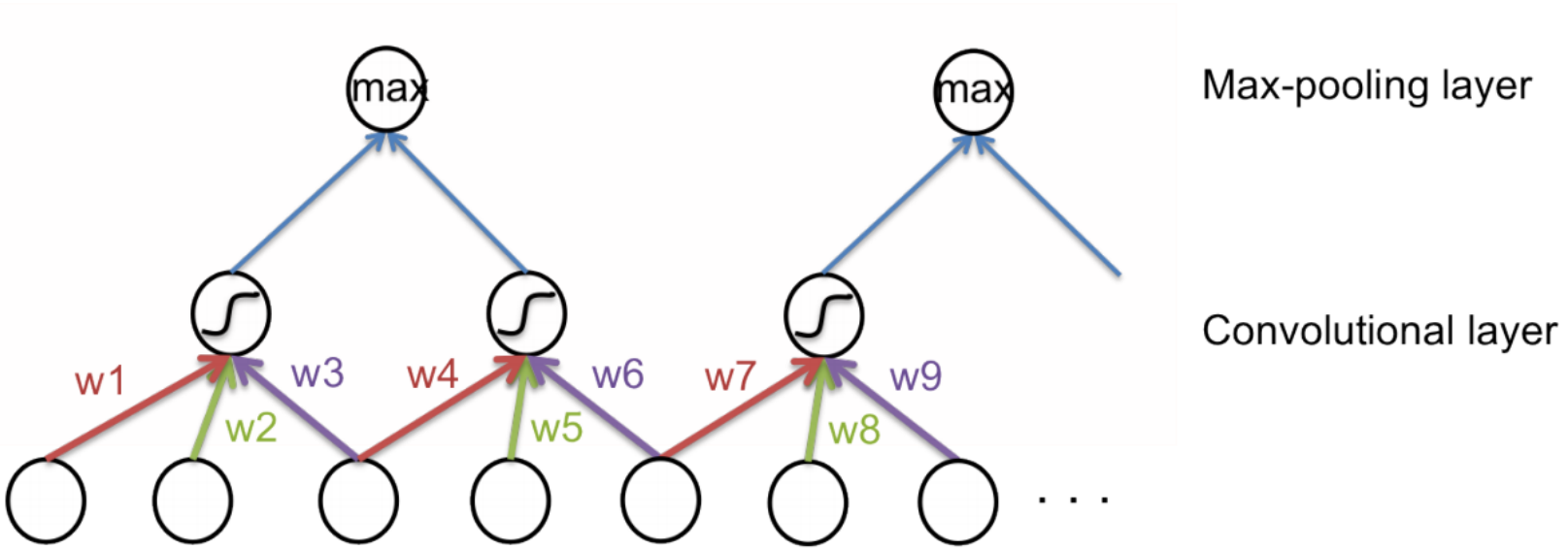

Figure: Beispiel Pooling-Layer [L15]

- Hat sich in vielen Anwendungen als vielversprechend erwiesen
- Es müssen keine Gewichte im Pooling-Layer gelernt werden
- AVG-Pooling möglich

Pooling-Layer: Verschiebung der Eingabedaten

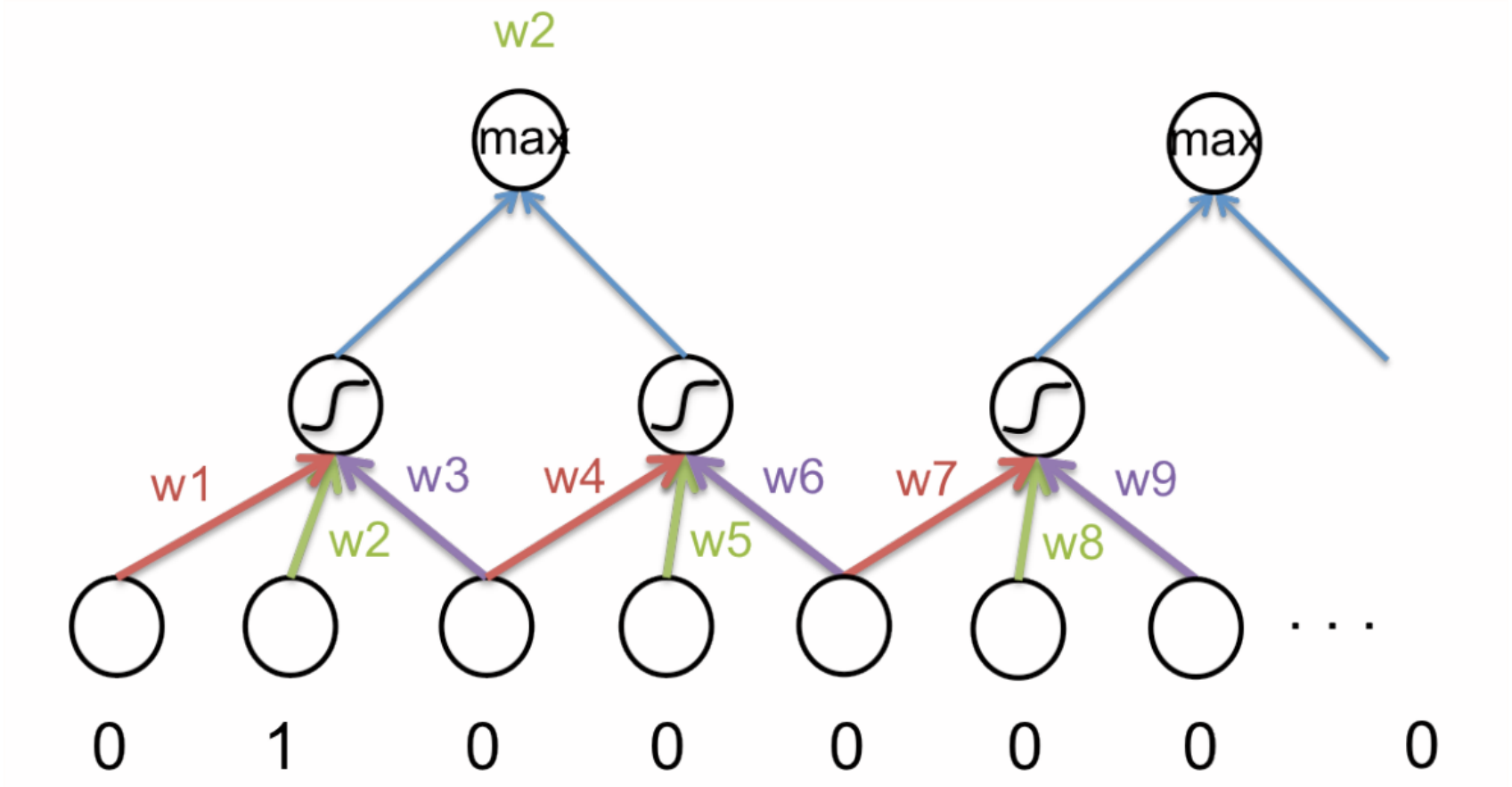

Figure: Beispiel Pooling-Layer [L15]

Pooling-Layer: Verschiebung der Eingabedaten

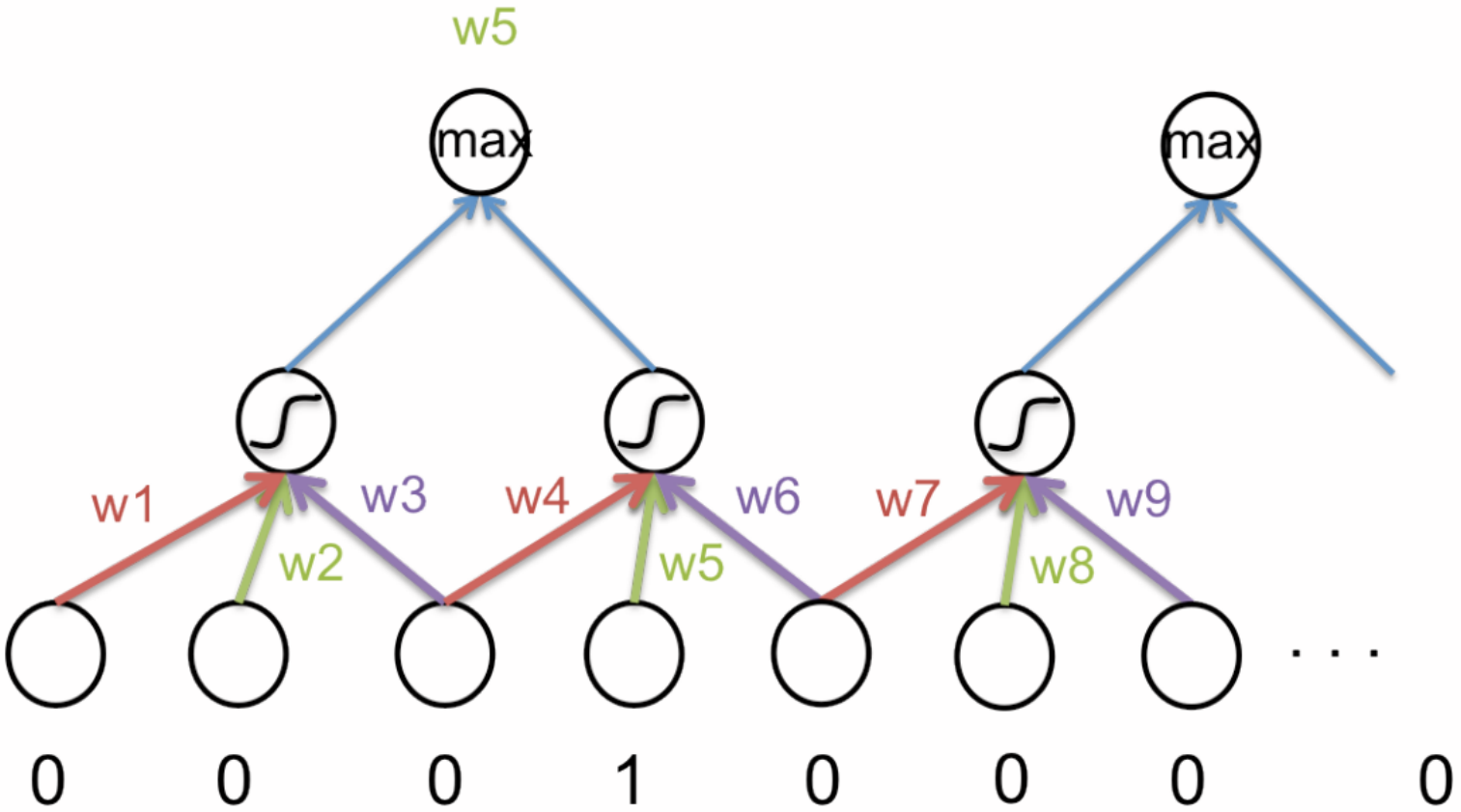

Figure: Beispiel Pooling-Layer [L15]

Convolutional Neural Networks: Multichannel

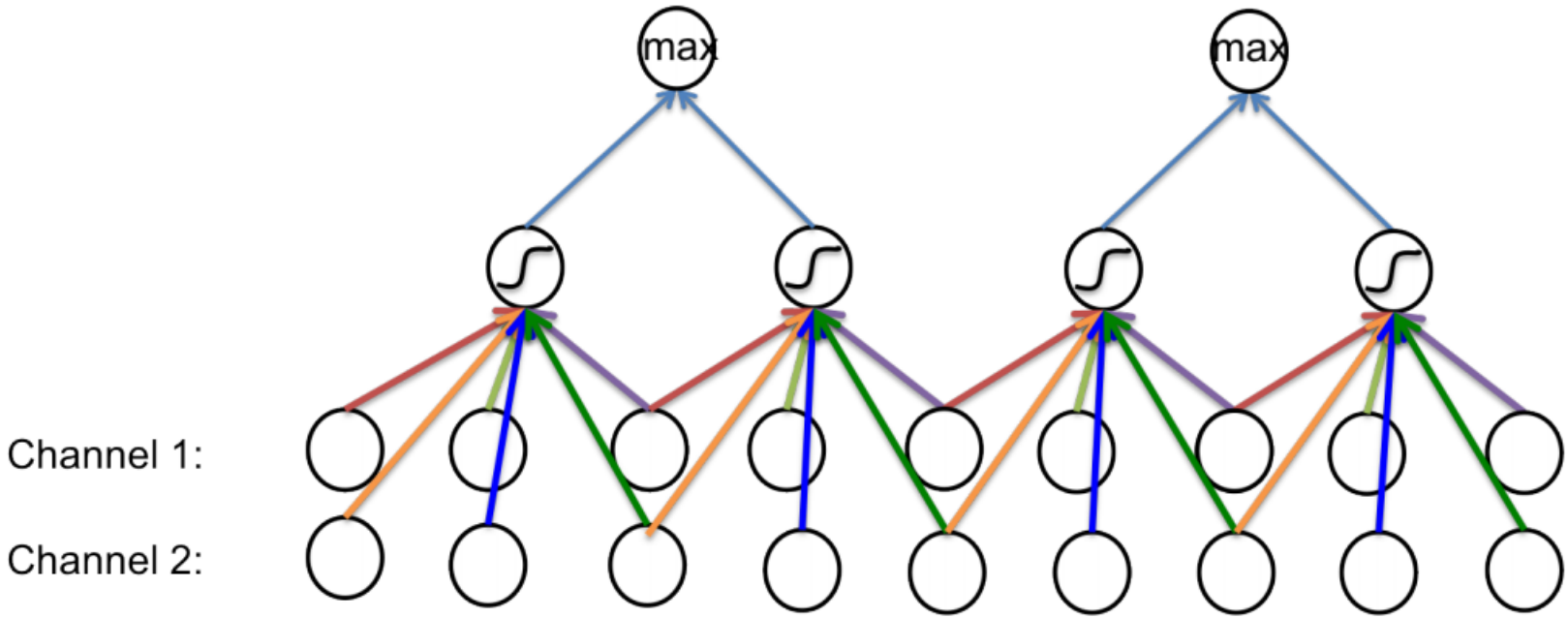

Figure: Beispiel Multichannel Convnet [L15]

Anwendung von CNN in der Texterkennung

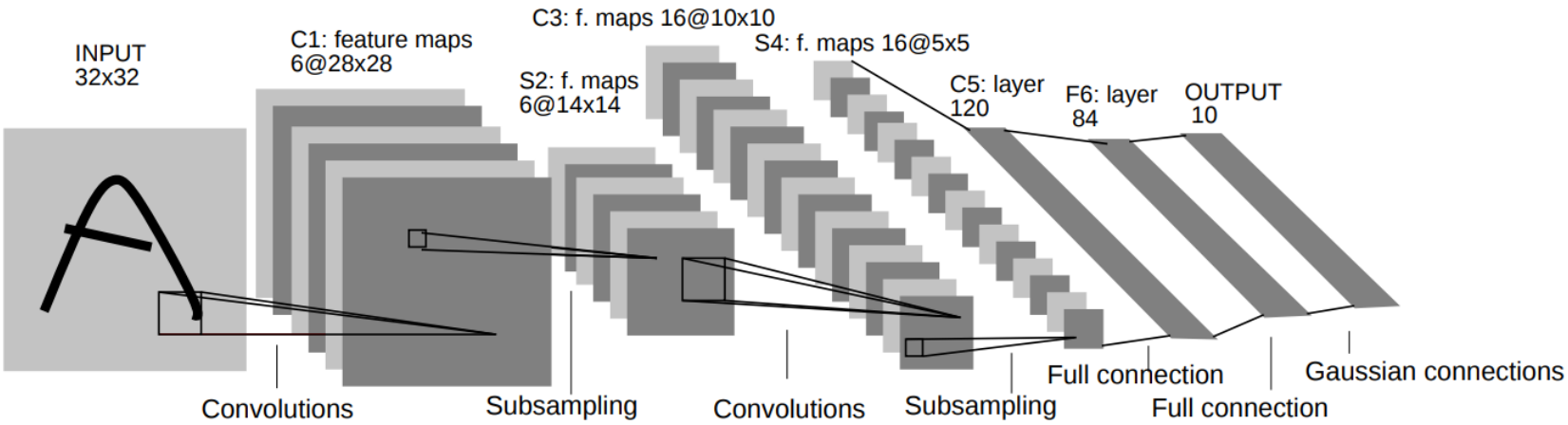

Figure: Architektur von LeNet-5 (LeCun et. al. [LBBH98])

Anwendung von CNN in der Bildklassifikation

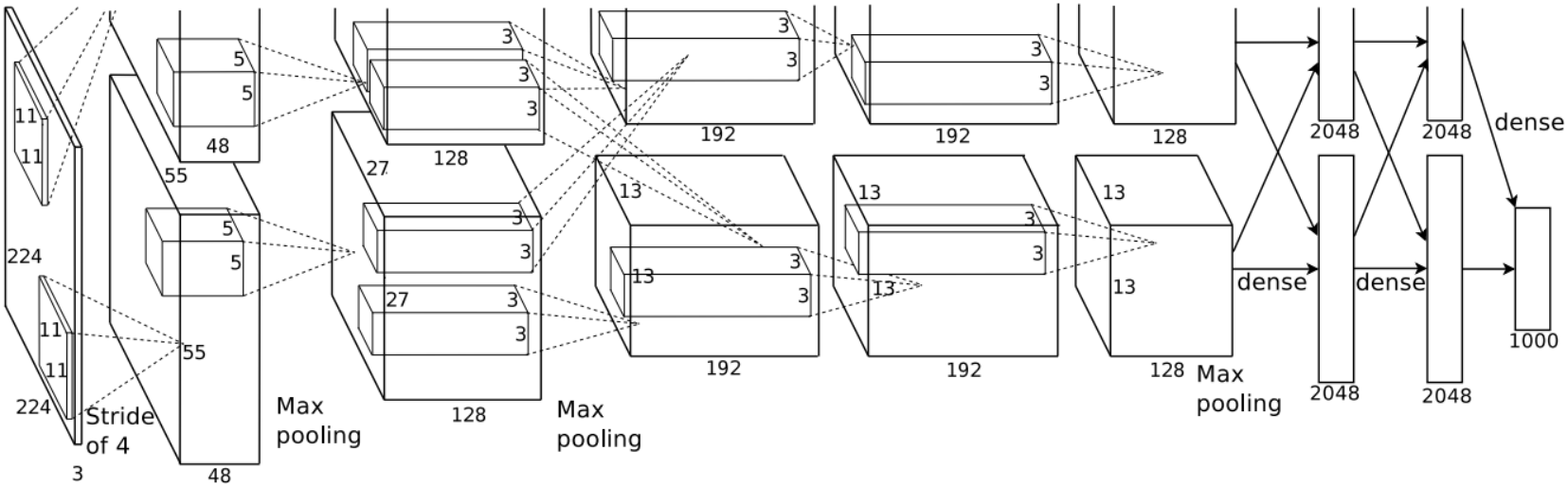

Figure: Architektur zur Objekterkennung (Krizhevsky et. al. [KSH12])

Zusammenfassung

Autoencoder

- Dimensionsreduzierung
- Kompression oder Visualisierung
- Verwendung als Initialisierungsmethode für KNN

Convolutional Neural Networks

- Angelehnt an den Convolution-Operator
- Verarbeitung natürlicher hochdimensionaler Daten
- State-of-the-art in Objekterkennung, Texterkennung, etc.

Literatur

[L15] Quoc V Le et al. (2015)

A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks

Google Brain

[HS06] Hinton, Geoffrey E and Salakhutdinov, Ruslan R (2003)

Reducing the dimensionality of data with neural networks

Science (313), 5786, 504-507

[EBC10] Erhan, Dumitru and Bengio, Yoshua and Courville, Aaron and Manzagol, Pierre-Antoine and Vincent, Pascal and Bengio, Samy (2010)

Why does unsupervised pre-training help deep learning?

Journal of Machine Learning Research (11), Feb, 625-660

Literatur

[LBBH98] LeCun, Yann and Bottou, Léon and Bengio, Yoshua and Haffner, Patrick (1998)

Gradient-based learning applied to document recognition

Proceedings of the IEEE 86 (11), 2278-2324

[KSH12] Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E (2012)

Imagenet classification with deep convolutional neural networks

Advances in neural information processing systems, 1097-1105

Vielen Dank!