Methoden des Deep Learning im Bereich Convolutional Neural Networks

Matthias Hermann

HTWG Konstanz,
Institut für Optische Systeme (IOS)

23. Oktober 2015

Inhalt

Ziel der Arbeit

Analyse von Convolutional Neural Networks (CNNs)

- Untersuchung des klassischen neuronalen Lernmodels (Feedforward-Netze)
- Aufbereitung der CNNs im Bereich Deep Learning

Entwicklung eines CNN-Prototyp

- Klassifikation von Bilddaten (Bilderkennung)
- Unüberwachtes Vortraining mittels Autoencoder
- Implementierung von Techniken zur Visualisierung
- Python-Schnittstelle mit wenigen Abhängigkeiten zu Bibliotheken

Deep Learning

Motivation

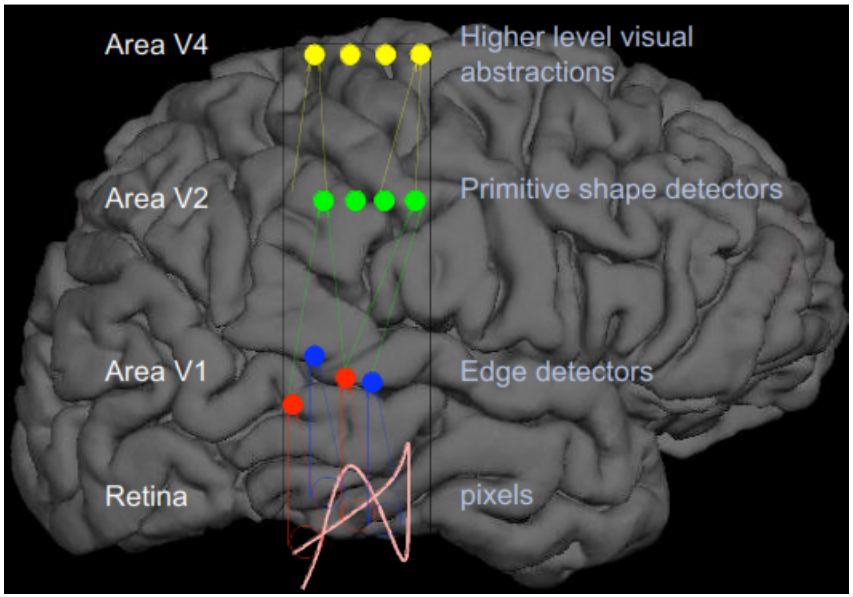


Abbildung : Tiefe Architekturen von biologischen Gehirnen am Beispiel des visuellen Cortex (Bengio und LeCun (2009))

Motivation

Restriktionen moderner Lernarlorithmen

- Fluch der Dimensionalität durch Variationen im Eingaberaum (Affine Transformationen, Verstärkung, Ausprägung)
- Selektion von Merkmalen
- Schlechte Skalierung mit steigender Anzahl an Trainingsdaten
- $\Rightarrow$ **Lokale Generalisierung**

Lösungsansatz mit Künstlichen Neuronalen Netzen (KNN)

$$f(x) = f''(f'(x))$$

Distributed Representation

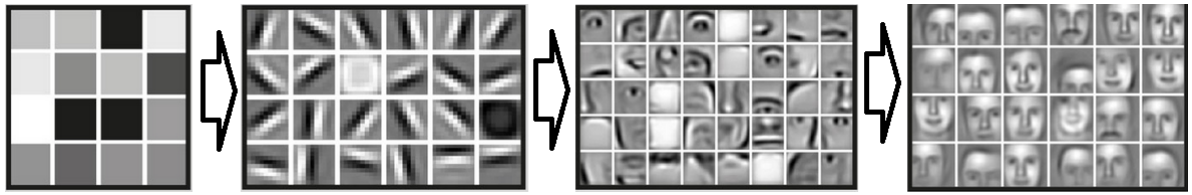


Abbildung : Partitionierung des Eingaberaums in unterschiedlichen Abstraktionsebenen (*Distributed Representation*) (Andrew Ng, Stanford Artificial Intelligence Lab)

Relevante Anwendungsfelder

- Bilderkennung
- Natürliche Sprache (NLP)
- Robotik

Renaissance im Deep Learning

- Bis 2006 keine nennenswerte Erfolge beim Training tiefer Architekturen
- Ausnahme: Convolutional Neural Networks
  LeNet 5 von LeCun et al. (1998a)

Durchbruch durch schichtweises Vortraining

- Hinton, G. E., Osindero, S., und Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets.
  Neural computation, 18(7):1527–1554

- Verallgemeinerung auf $\mathbb{R}$ (Bengio et al. (2007))
- Verallgemeinerung auf Bildausschnitte (Ranzato et al. (2006))

Forschungsfelder im Deep Learning

- Recurrent Neural Networks (RNN)
  (Hochreiter und Schmidhuber (1997))
- Convolutional Neural Networks (CNN)
  (LeCun et al. (1998a))
- Deep Belief Networks (DBN)
  (Bengio et al. (2007))
- Stacked Denoising Autoencoders (SDA)
  (Vincent et al. (2008))
- Deep Reinforcement Learning (DRL)
  (Mnih et al. (2013))

Convolutional Neural Networks

Multilayer Perceptrons (MLP)

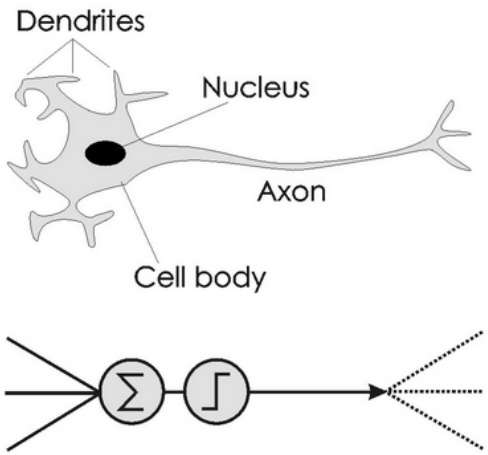


Abbildung : Schematischer Vergleich von biologischen und künstlichen Neuronen von McCulloch und Pitts (1943)

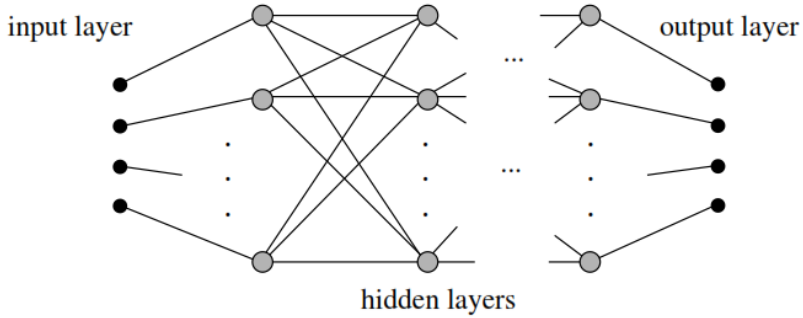


Abbildung : Generische mehrschichtige Architektur eines MLP (Rojas (1996))

Multilayer Perceptrons (MLP)

Einfachster Fall eines MLP:

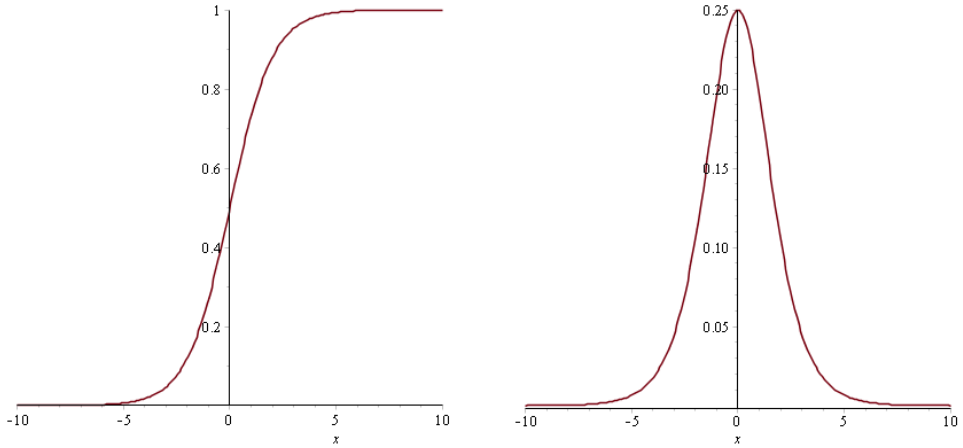
- Ein Hidden-Layer h mit einem Neuron
- Ein Output-Layer o mit einem Neuron

Formale Beschreibung

$$f(x) = \phi(\langle w_o, \phi(\langle w_h, x \rangle + b_h) \rangle + b_o)$$

- Eingabevektor x
- Gewichtsvektor w
- Schwellwert b
- Aktivierungsfunktion $\phi(\cdot)$

Logistische Sigmoid-Aktivierungsfunktion



- Funktion: $sig(x) = \frac{1}{1+\exp^{-t}}$
- Ableitung: $sig'(x) = sig(x)(1 - sig(x))$

Backpropagation-Algorithmus

- Verallgemeinerung der Delta-Regel im LMS-Algorithmus
- Verfahren zur Berechnung partieller Ableitungen mittels Kettenregel (Fehlerrückführung)

Beispiel Output-Layer L

Kostenfunktion: $J(W,b) = \frac{1}{2N}\sum_{i=1}^{N}(f(x_i) - y_i)^2$

Delta-Regel: $\delta^L = (f(x_i) - y_i) \circ \phi'(z^L)$

Rückpropagierung: $\delta^L_{message} = (W^L)^T \delta^L$

Partielle Ableitungen Output-Layer L

$\frac{\partial J(W,b)}{\partial W^L} = \frac{1}{N}\sum_{i=1}^{N} \delta^L (x_i^L)^T$ $\quad$ $\frac{\partial J(W,b)}{\partial b^L} = \frac{1}{N}\sum_{i=1}^{N} \delta^L$

Beispiel Funktionsapproximation

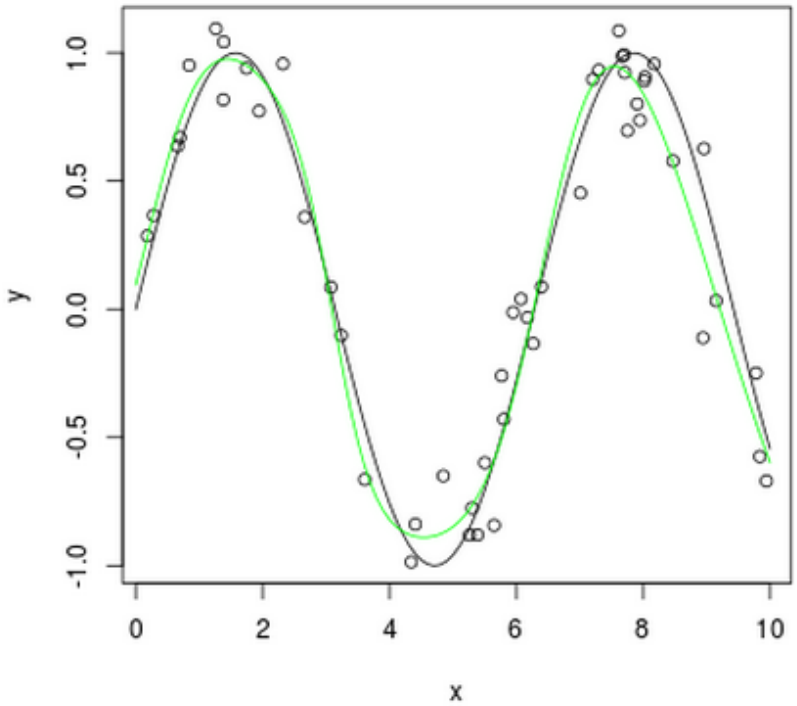


Abbildung : Approximierte Sinuskurve mit einem Hidden-Layer mit 6 Neuronen (grün)

Beispiel Logistische Regression

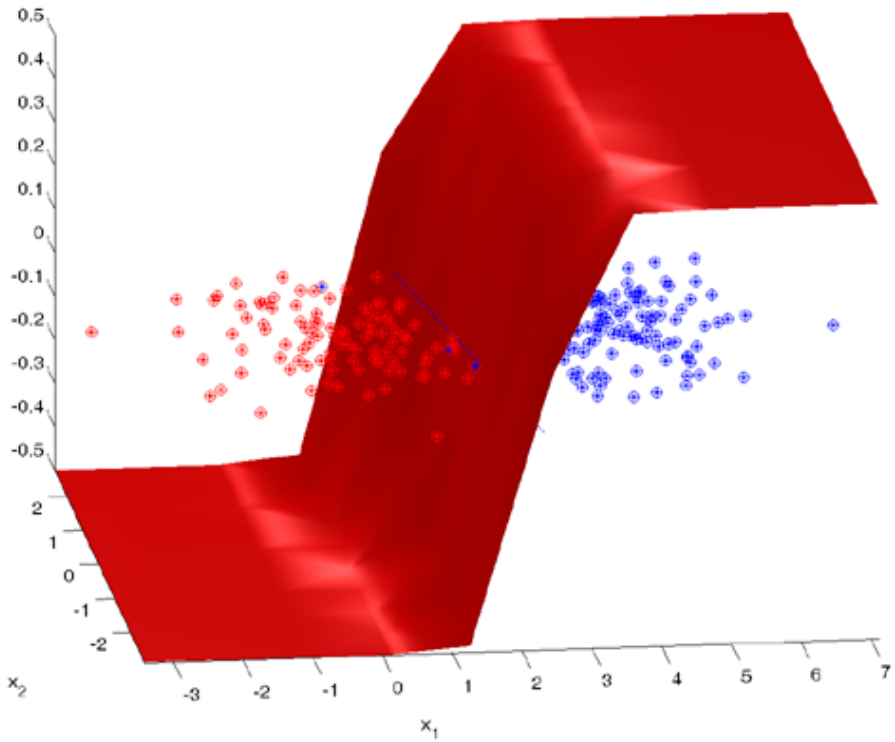


Abbildung : Logistische Regression mit zwei Klassen (Vadim Strijov, Computing Centre of the Russian Academy of Sciences)

Output-Layer

- Anzahl Neuronen im Output-Layer bestimmen Dimension der Ausgabe
- Lineare Aktivierungsfunktion für Regression
- Softmax-Regression für Klassifikation

Kostenfunktion Logistische Regression

$$-ln(p(y|X,\hat{W})) = -\sum_{i=1}^{N} y_i ln(f(x_i)) + (1-y_i) ln(1-f(x_i))$$

mit $f(x_i) = \frac{1}{1+\exp^{-\hat{W}x_i}}$

- Negative Log-Likelihood (Cross-Entropy-Fehlermaß)

Convolutional Neural Networks (CNNs)

- Spezielle Klasse von MLPs
- Zusätzliche Convolution-Layer mit *Feature-Maps* als Ausgabe
- Training mittels Backpropagation

Berechnung einer *Feature-Map* i im Convolution-Layer

$$z_i = \sum_{j=0}^{m} x_j * W_{ij}$$

$$a_i = \phi(z_i + b_i)$$

- Eingabe x
- Faltungsmaske W_{ij}
- Schwellwert b_i
- Aktivierungsfunktion $\phi(\cdot)$

Lokales Rezeptives Feld

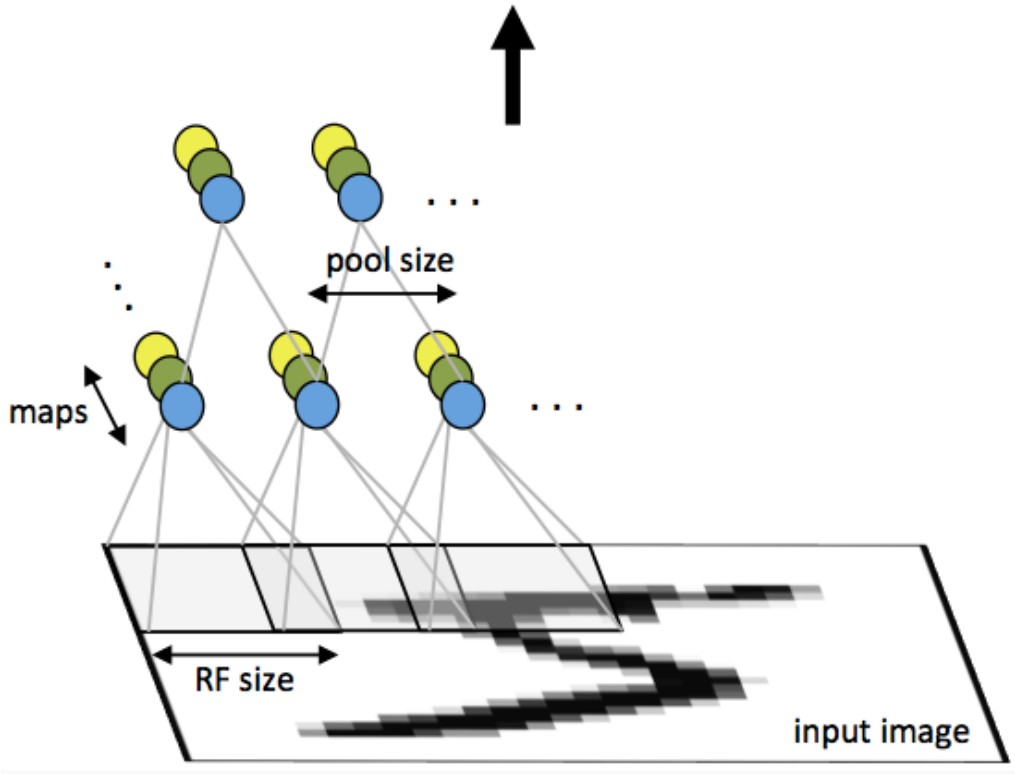


Abbildung : Lokales Rezeptives Feld auf Basis der Entdeckungen von Hubel und Wiesel (1962) hinsichtlich des visuellen Cortex (Fei-Fei und Karpathy (2014))

Convolution-Layer

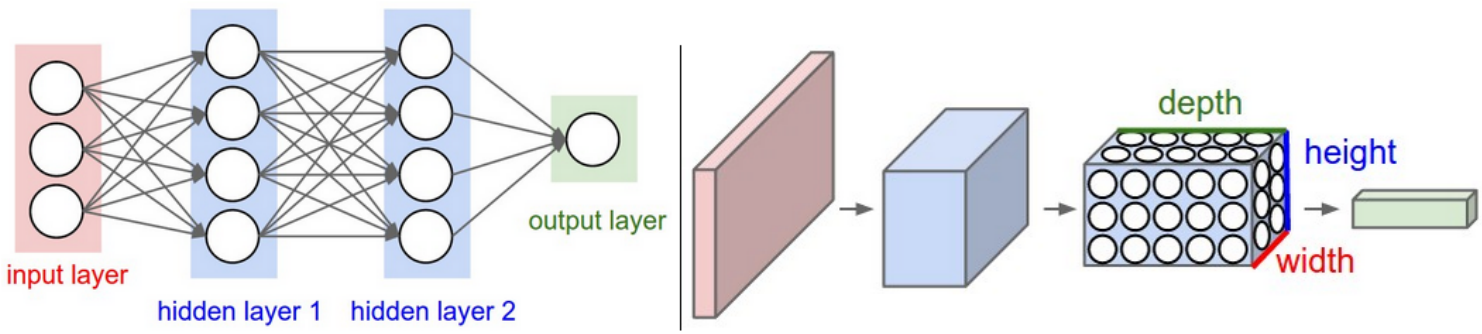


Abbildung : Beispiel Convolution-Layer (Fei-Fei und Karpathy (2014))

Pooling-Layer

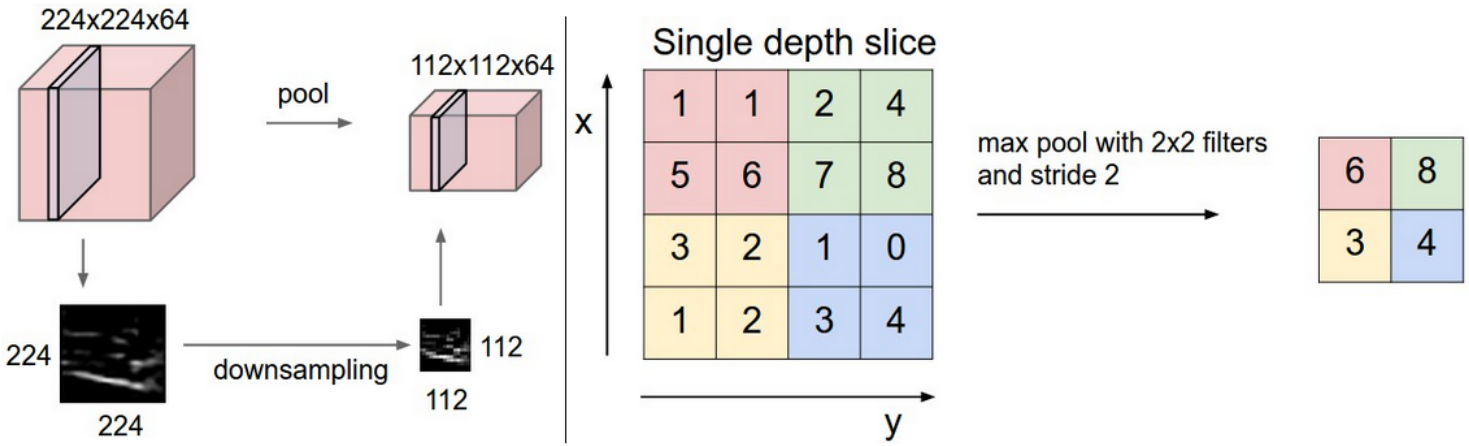


Abbildung : Beispiel Max-Pooling-Layer (Fei-Fei und Karpathy (2014))

Eigenschaften der CNN-Architektur

- Lokale Extraktion von Merkmalen (*Local Feature Extraction*)
- Translationsinvarianz hinsichtlich Eingabedaten (geringe Skalierungs- und Rotationsinvarianz)
- Einfacheres Training durch weniger Parameter und Verbindungen als MLP mit gleich vielen Neuronen (*Parameter Sharing*)
- Nicht-lokale Generalisierung durch Verschachtlung nichtlinearer Funktionen

(Vgl. LeCun et al. (1998a), Bengio und LeCun (2007) und Zeiler und Fergus (2014)))

Beispielarchitektur

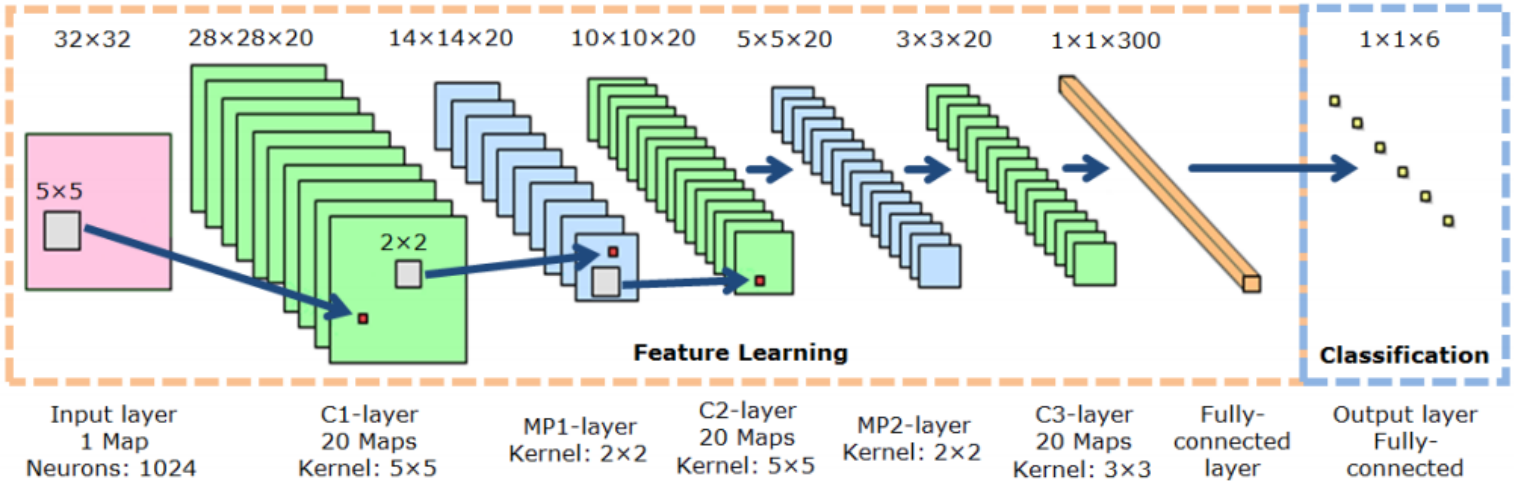


Abbildung : Schematisches Convolutional Neural Network (Nagi et al. (2011))

Beispielarchitektur: AlexNet

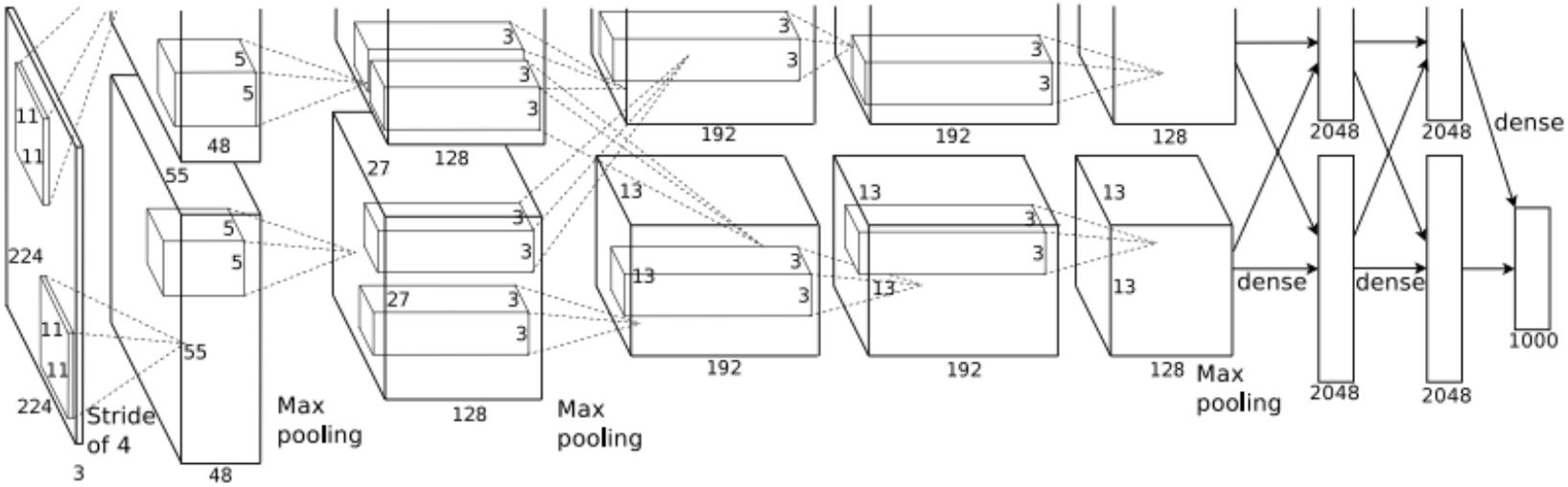


Abbildung : Architektur von Krizhevsky et al. (2012) für ImageNet 2012 (Klassifikationsproblem mit 200 Klassen und 450.000 482×415 Pixel großen Trainingsbeispielen)

Deep Learning und CNNs?

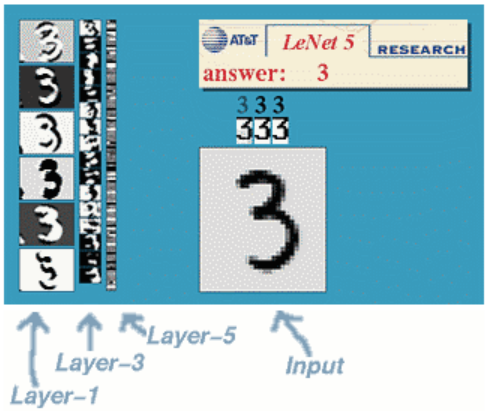


Abbildung : LeNet 5 von LeCun et al. (1998a)

Fehlerraten von LeNet 5 auf dem MNIST-Datensatz

- 0.95 % ohne erweiterten Trainingsdaten
- 0.85 % mit erweiterten Trainingsdaten
- 20 Epochen Training (Iterationen über Trainingsset)

Deep Learning und CNNs!

Auch durch die Verwendung von CNNs bleiben dennoch zentrale Schwierigkeiten im Deep Learning bestehen:

- Verschwinden des Gradienten (*Vanishing Gradient*) in tiefen Netzen (vgl. Hochreiter (1991))
- Gradientenabstieg bei nicht-konvexer Zielfunktion (vgl. Martens (2010) und Dauphin et al. (2014))
- *Overfitting* bei großen Netzen, insbesondere in nachgeschalteten MLPs (vgl. Hinton et al. (2012))

Vanishing Gradient-Effekt

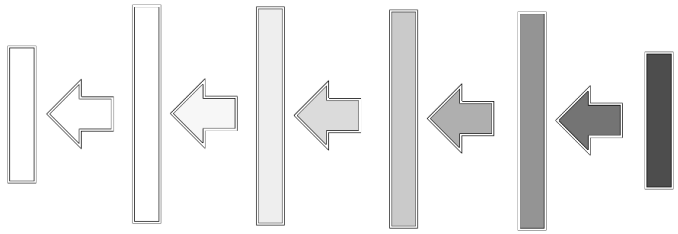


Abbildung : Der Vanishing Gradient-Effekt beschreibt bei der Rückpropagierung das abklingende Fehlersignal von Output- hin zu Input-Layer

Backward-Pass im Backpropagation-Algorithmus

$$\delta^l_{message} = (W^l)^T(\delta^{l+1}_{message} \circ \phi'(z^l))$$

mit $\phi'(z^l) <= 1$

Xavier-Initialisierung I

- Normalisierte Signalpropagierung (Forward-Pass und Backward-Pass) von Glorot und Bengio (2010)
- Zufällige Initialisierung mit unterschiedlicher Varianz pro Schicht
- Wirkt dem Vanishing-Gradient-Effekt entgegen

Formale Definition

$$W \sim \mathcal{U}[-\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}]$$

basierend auf $Var(W) = \frac{2}{fan_{in}+fan_{out}}$

Xavier-Initialisierung II

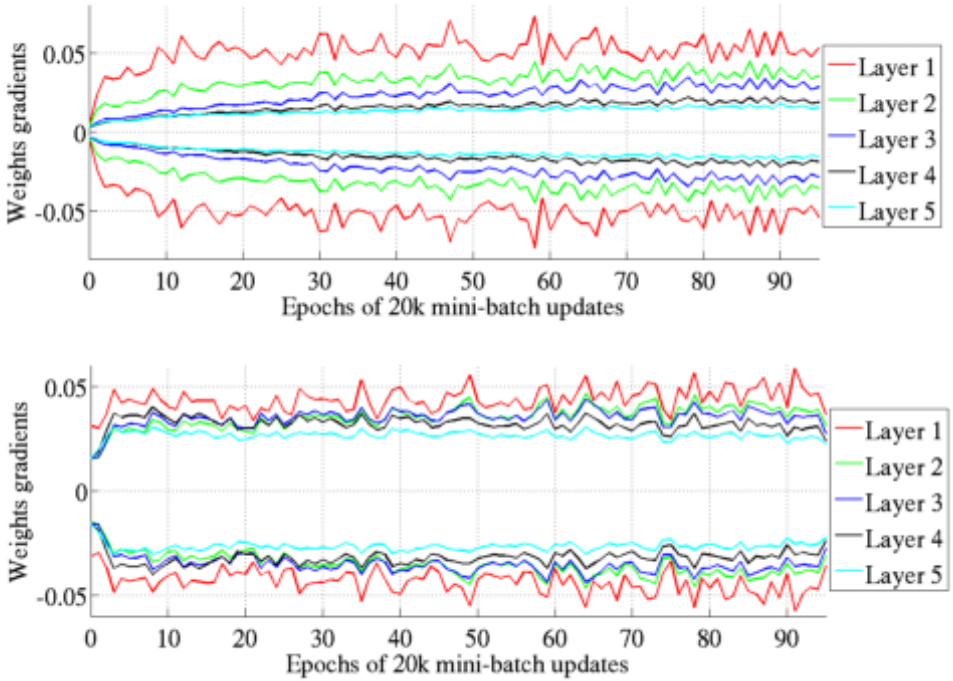
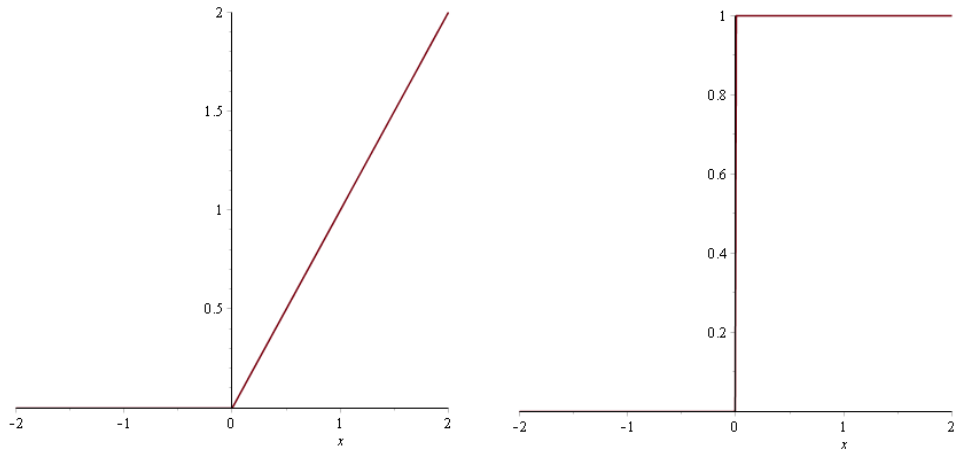


Abbildung : Vergleich der Varianz der Gradienten im Laufe des Trainings (Standard-Initialisierung (oben) und Xavier-Initialisierung (unten)): Layer 1 entspricht dem Output-Layer mit der größten Varianz im Gradient (Glorot und Bengio (2010))

ReLu-Aktivierungsfunktionen

Rectified Linear Units (ReLus) können nicht sättigen und verhindern somit ein Verblassen des Fehlersignals (Glorot et al. (2011)).



- Funktion: $f(x) = max(0, x)$
- Ableitung: $f'(x) = \begin{cases} 0 & \text{für } 0 \geq x \\ 1 & \text{für } 0 < x \end{cases}$

Unüberwachtes Vortraining mittels Denoising Autoencoder

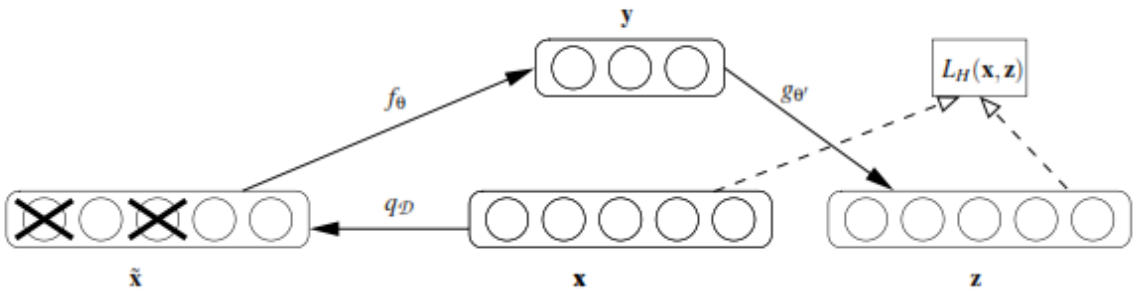


Abbildung : Schemadarstellung des Denoising Autoencoders (Vincent et al. (2010))

Formale Definition

$$h = \phi(W\hat{x} + b)$$

$$y = \phi(W'h + b') \; Loss = (x - y)^2$$

Convolutional Autoencoder I

Verschiedene Herangehensweisen

- Sequenzielle Verarbeitung von Bildausschnitten in Filtergröße und Training mittels Standard Autoencoder (vgl. Ranzato et al. (2006))
- Berechnung aller Bildausschnitte gleichzeitig mittels Faltung und einem dem Backward-Pass ähnlichen Verfahren (vgl. Masci et al. (2011))

- Unüberwachtes Vortraining (schichtweise) führt zu einem besser konditionierten Lernproblem und kann so die Performanz verbessern
- Bei CNNs, aufgrund von *Parameter Sharing*, nicht so wirkungsvoll wie bei MLPs (Abdel-Hamid et al. (2013))

Convolutional Autoencoder II

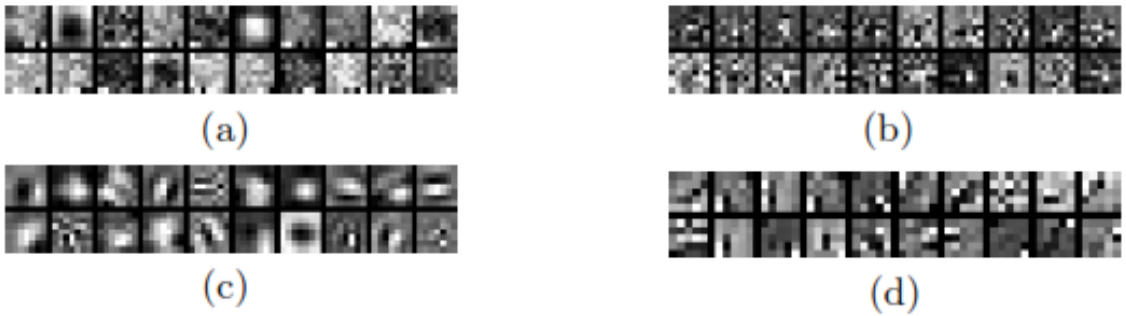


Abbildung : Filter der ersten Schicht eines Convolutional Autoencoders (a) Kein Max-Pooling, 0 % Rauschen, (b) Kein Max-Pooling, 50 % Rauschen, (c) Max-Pooling, (d) Max-Pooling , 30 % Rauschen (Masci et al. (2011))

Beispiel Fehlerlandschaft

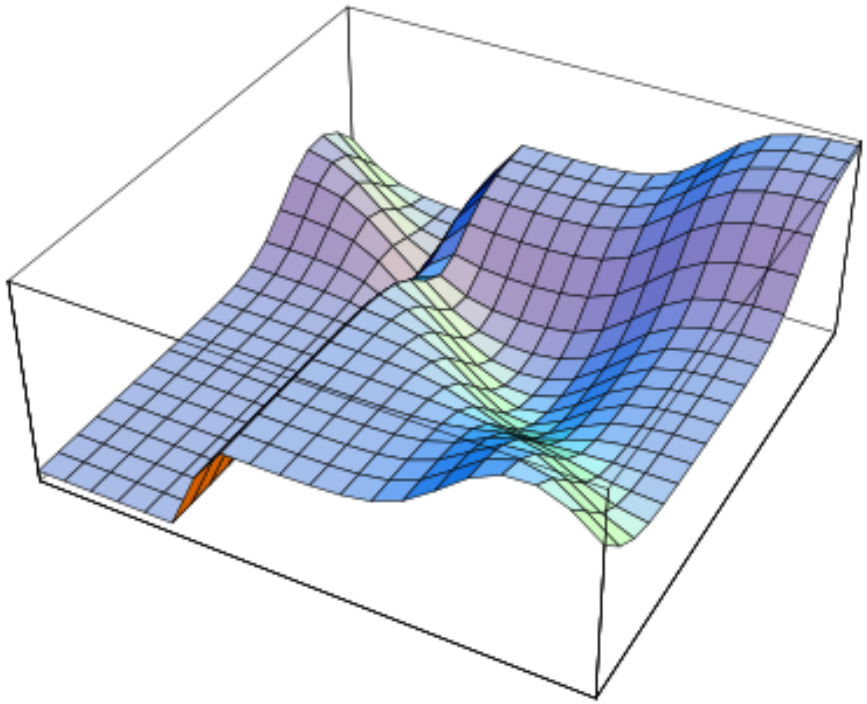


Abbildung : Fehlerfunktion beziehungsweise Zielfunktion im Gewichtsraum mit lokalem Minimum (Rojas (1996))

Optimierung mittels Gradientenabstieg

Stochastischer Gradientenabstieg (SGD)

$$\nabla J(W, b) = \mathbb{E}[\nabla(f(x_i) - y_i)^2] = \frac{1}{M} \sum_i^M \nabla(f(x_i) - y_i)^2$$

$$W_{t+1} = W_t - \eta \frac{1}{M} \sum_{i=1}^M \nabla J(x_i)$$

mit Lernrate η und M Trainingsbeispielen

- Online-Training: $M = 1$
- Batch-Training: $M = N$
- Averaged-SGD: $M \epsilon [1, N]$ (Mini-Batch)

(Nesterov) Momentum-Methode

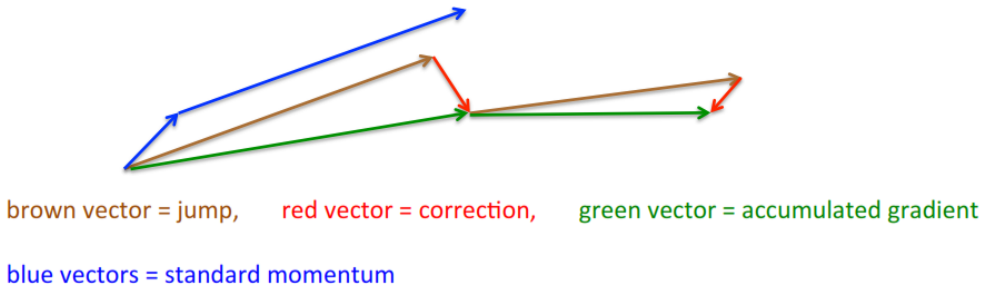


Abbildung : Das Nesterov-Momentum korrigiert im Unterschied zur Momentum-Methode Fehler im Nachhinein (Hinton et al. (2015))

Formale Definition der Momentum-Methode

$$\Delta W_t = \mu \Delta W_{t-1} - \eta \nabla J(W_t, b_t)$$

$$W_{t+1} = W_t + \Delta W_t$$

mit Momentum-Parameter μ

Adaptive Lernrate I

Newton-Schritt

$$W_{t+1} = W_t - H_t^{-1} \nabla J(W_t, b_t)$$

- Quasi-Newton-Verfahren führt bei konvexen Zielfunktionen in zum globalen Minimum
- Hesse-Matrix sehr teuer zu berechnen
- $\rightarrow$ **Approximation der diagonalen Hesse-Matrix**

Adaptiver Lernrate II

Genannte Verfahren schätzen unterschiedliche Formen der diagonalen Hesse-Matrix: $|diag(H)|$, $\sqrt{diag(H)^2}$ oder $\sqrt{diag(H^2)}$

Beispiele

- Stochastischer LMA (LeCun et al. (1998b))
- Rprop / RMSprop (Riedmiller und Braun (1992) und Hinton et al. (2015))
- AdaDelta (Zeiler (2012))
- Equilibrium SGD (Dauphin et al. (2015))

RMSprop

RMSprop approximiert $\sqrt{diag(H^2)}$ und stellt eine stochastische Variante von Rprop dar.

Formale Definition

$$\hat{H}_{t+1} = \rho \hat{H}_t + (1 - \rho) \nabla J(W_t, b_t)^2$$

$$\eta_i = \frac{\eta}{\mu + \sqrt{\hat{H}_{ii}}}$$

mit Faktor ρ für exponentielle Glättung und Dämpfungsfaktor μ

RMSprop bietet Vorteile bei einer Hesse-Matrix mit negativen Eigenwerten (Sattelpunkte) (Dauphin et al. (2015)).

Overfitting

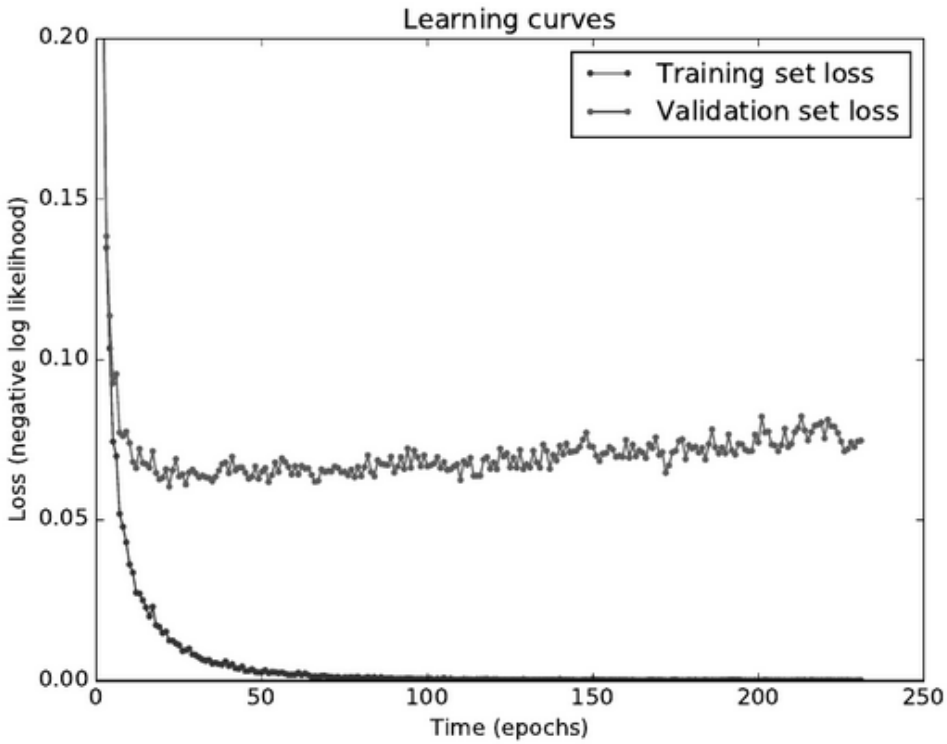


Abbildung : Overfitting am Beispiel des MNIST-Datensatz (Bengio et al. (2015))

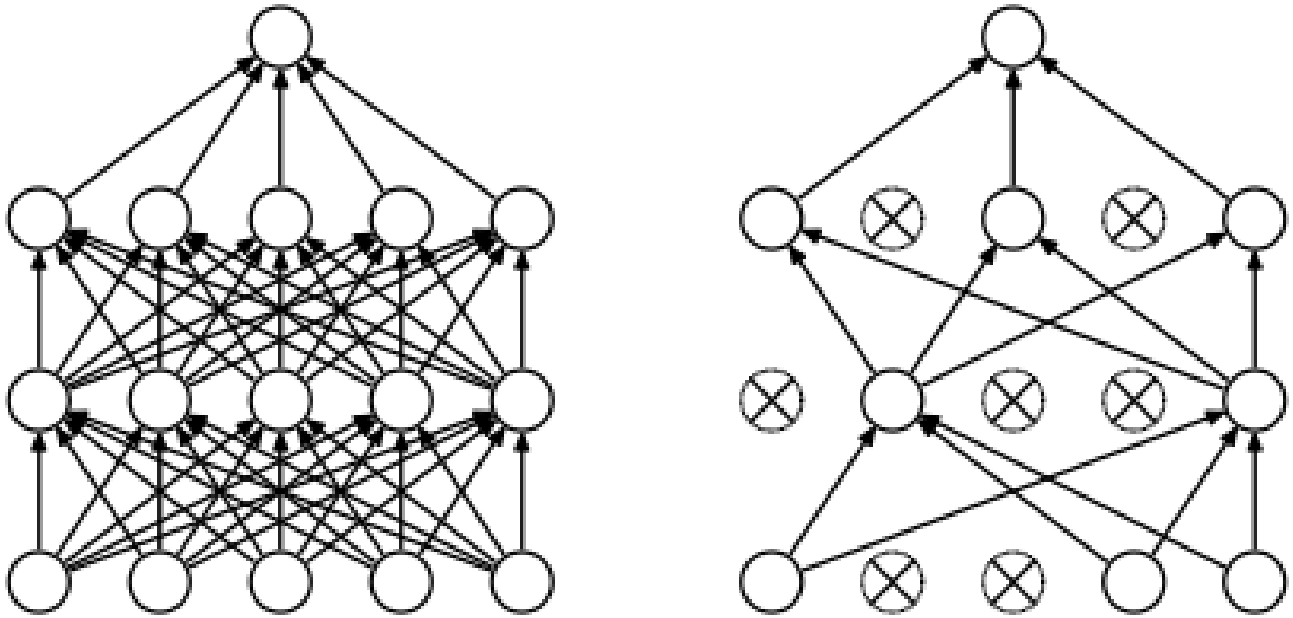
Regularisierung

Neben den bekannten Verfahren wie L1-/L2-Regularisierung (*Weight Penalty*) oder Max-Norm-Regularisierung existieren im Bereich Deep Learning spezialisierte Verfahren:

Spezielle Verfahren

- Modellkapazität durch *Parameter-Sharing* limitieren (CNNs).
- Early-Stopping bricht Training bei Verschlechterung des Validierungsfehlers vorzeitig ab.
- Erweiterung der Trainingsdaten durch additives Rauschen oder affinen oder elastischen Transformationen (*Data Augementation*)

Dropout-Regularisierung



(a) Standard Neural Net (b) After applying dropout.

Abbildung : Schematische Darstellung eines gewöhnlichen MLP (links) und eines Dropout-MLP (rechts) (Srivastava et al. (2014))

Experimente

Betrachtete Datensätze I

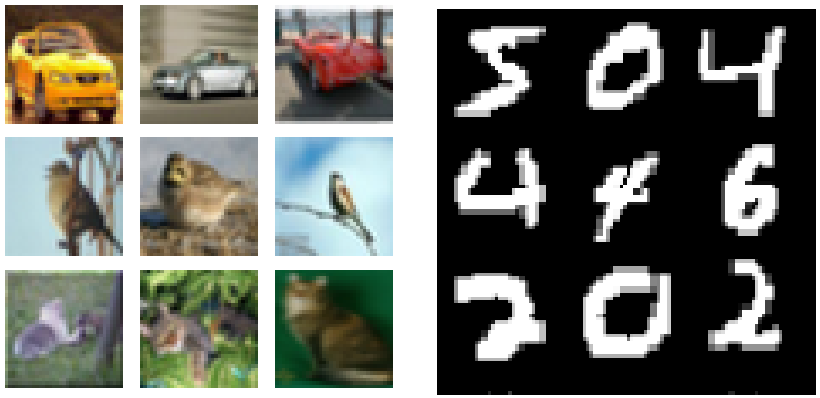


Abbildung : Beispiele aus CIFAR-10 (links) und MNIST (rechts)

Betrachtete Datensätze II

MNIST (Binarisierte handgeschriebene Ziffern)

- 60000 28×28 Pixel große Trainingsbeispiele sowie 10000 Testbeispiele.
- Richtwert für Fehlerrate ohne Vorverarbeitung liegt bei 0.7 % (Ranzato et al. (2006)).
- Beste Ergebnis erreicht das *DropConnect Network* von Wan et al. (2013) mit 0.21 % Fehler.
- Angewandte Vorverarbeitung: Zentrierung

Betrachtete Datensätze III

CIFAR-10-Datensatz (Farbbilder)

- 50000 32×32 Pixel große Trainingsbeispiele und 10000 Testbeispiele.
- Die Bilder sind aufgeteilt in 10 Klassen.
- Richtwert für Fehlerrate liegt bei 21.8 % Fehler (Masci et al. (2011)).
- Bestes Ergebnis mit 8.2 % Fehler erzielt das *Deeply-Supervised Net* von Lee et al. (2014).
- Angewandte Vorverarbeitung: Zentrierung (CIFAR-10A (RGB) / CIFAR-10B (HSV))

Architektur LeNet 5+ (MNIST)

Die Netz ist ein erweitertes *LeNet 5* und orientiert sich an dem von Ranzato et al. (2006) vorgestellten Modell.

Layer-Definition

1. Convolution-Layer: 50 *Feature-Maps* mit 5×5 Filtermasken
2. Pooling-Layer: 2×2 Filtermasken
3. Convolution-Layer: 50 *Feature-Maps* mit 5×5 Filtermasken
4. Pooling-Layer: 2×2 Filtermasken
5. Hidden-Layer: 200 Neuronen
6. Output-Layer: 10 Neuronen mit Cross-Entropy Fehlermaß

Dieses Netz umfasst 63.750 Gewichte in den Convolution-Layern und 162.000 Gewichte im MLP.

Architektur Net-7 (CIFAR-10)

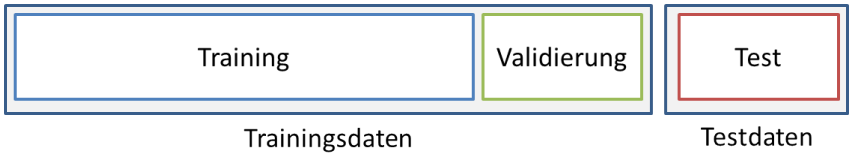
Dieses Netz orientiert sich an den Modellen von Hinton et al. (2012) und Zeiler und Fergus (2013).

Layer-Definition

1. Convolution-Layer: 64 *Feature-Maps* mit 5×5 Filtermasken
2. Pooling-Layer: 2×2 Filtermasken
3. Convolution-Layer: 64 *Feature-Maps* mit 5×5 Filtermasken
4. Pooling-Layer: 2×2 Filtermasken
5. Convolution-Layer: 64 *Feature-Maps* mit 5×5 Filtermasken
6. Hidden-Layer: 64 Neuronen
7. Output-Layer: 10 Neuronen mit Cross-Entropy Fehlermaß

Dieses Netz umfasst 209.600 Gewichte in den Convolution-Layern und 10.496 Gewichte im MLP.

Trainingsmethode



- Early-Stopping mit 5 Epochen Wartezeit
- 90% Trainingsdaten und 10 % Validierungsset
- 40 Trainingsbeispiele pro Mini-Batch
- Lernrate $\eta = 0.01$
- Keine *Data-Augmentation*

Unüberwachtes Vortraining

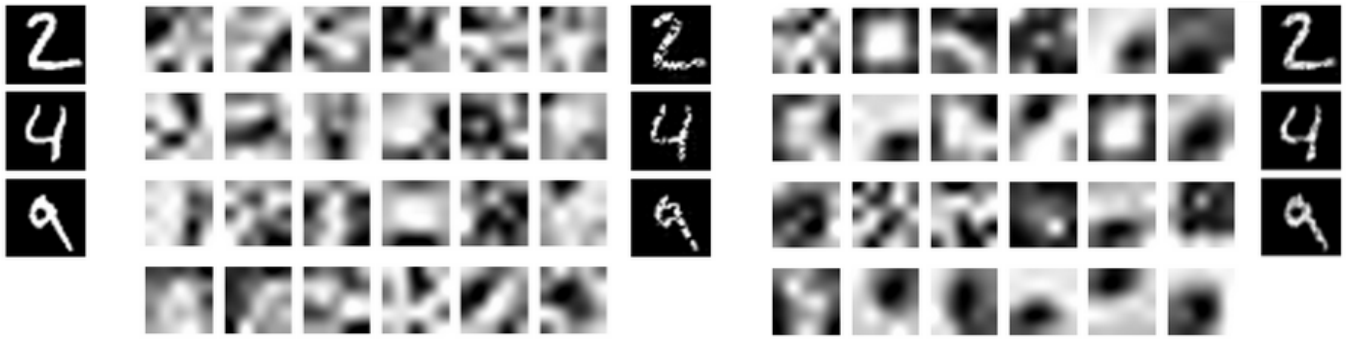


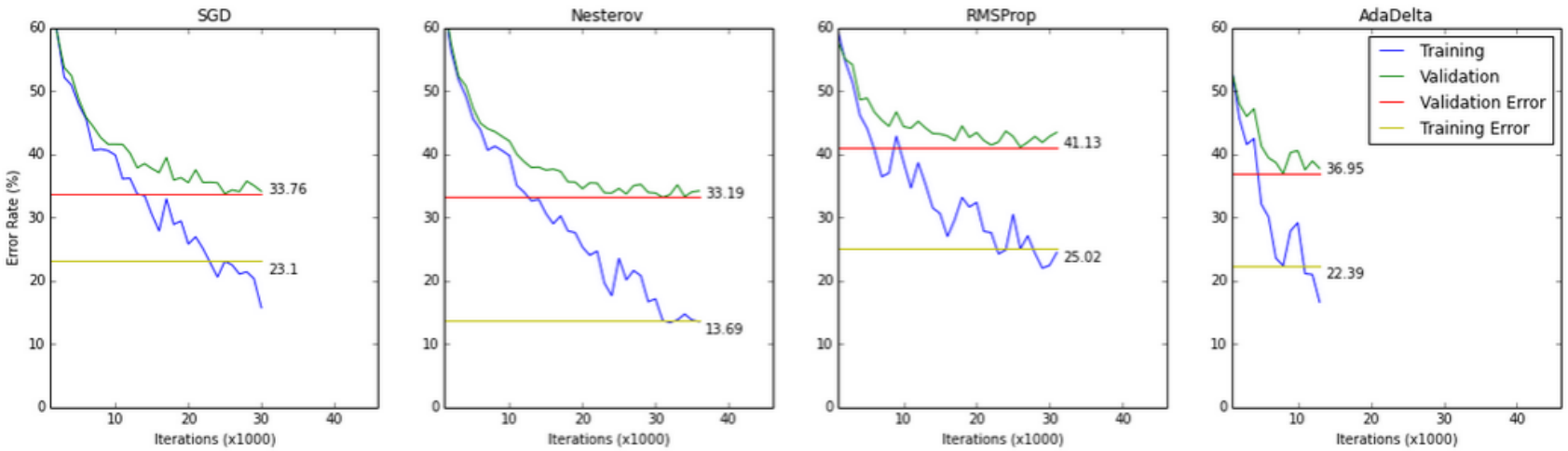
Abbildung : Mittels Convolutional Autoencoder trainierte Filtermasken der ersten Schicht und erzeugte Rekonstruktionen: Ohne Maskierung (links) und mit 30 % Maskierung (rechts).

Klassifikation I

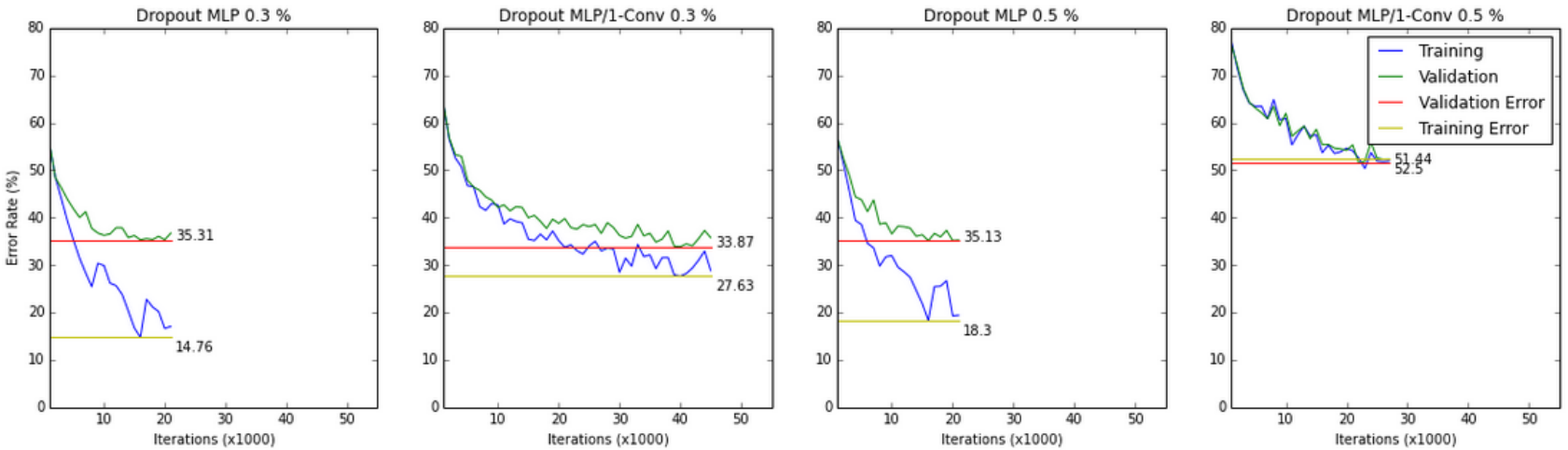
	MNIST	CIFAR-10B
SGD	0.40/0.82/0.93 % (21)	18.87/30.29/33.31 % (27)
Nesterov	0.06/0.12/0.69 % (11)	17.77/28.94/34.01 % (34)
RMSprop	0.22/0.48/0.89 % (10)	29.03/37.74/41.92 % (29)
AdaDelta	0.02/0.18/0.61 % (12)	20.86/21.99/36.65 % (12)

Tabelle : Fehler auf den Trainings-/Validierungs-/Testdaten (Epochen) nach dem gesamten Training mit verschiedenen Methoden zum Gradientenabstieg

Klassifikation II (CIFAR-10B)



Klassifikation mit Dropout (CIFAR-10B)



Architektur Net-7-Small (CIFAR-10B)

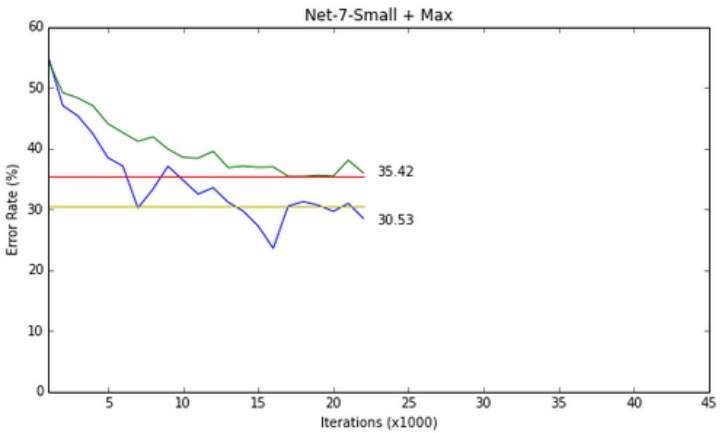
Entfernung von Hidden-Layer und Reduktion der Anzahl der *Feature-Maps* auf 20 reduziert:

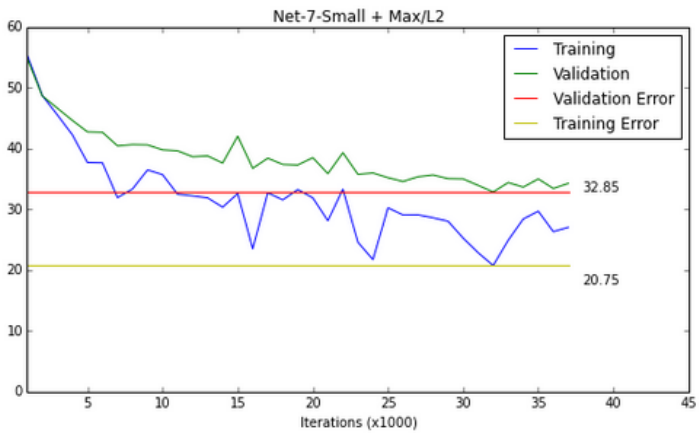
Layer-Definition

1. Convolution-Layer: 20 *Feature-Maps* mit 5×5 Filtermasken
2. Pooling-Layer: 2×2 Filtermasken
3. Convolution-Layer: 20 *Feature-Maps* mit 5×5 Filtermasken
4. Pooling-Layer: 2×2 Filtermasken
5. Convolution-Layer: 20 *Feature-Maps* mit 5×5 Filtermasken
6. Pooling-Layer: 2×2 Filtermasken
7. Output-Layer: 10 Neuronen mit Cross-Entropy Fehlermaß

Insgesamt besitzt dieses Netz lediglich 24.700 Gewichte

Klassifikation Net-7-Small (CIFAR-10)





Visualisierung der Filtermasken

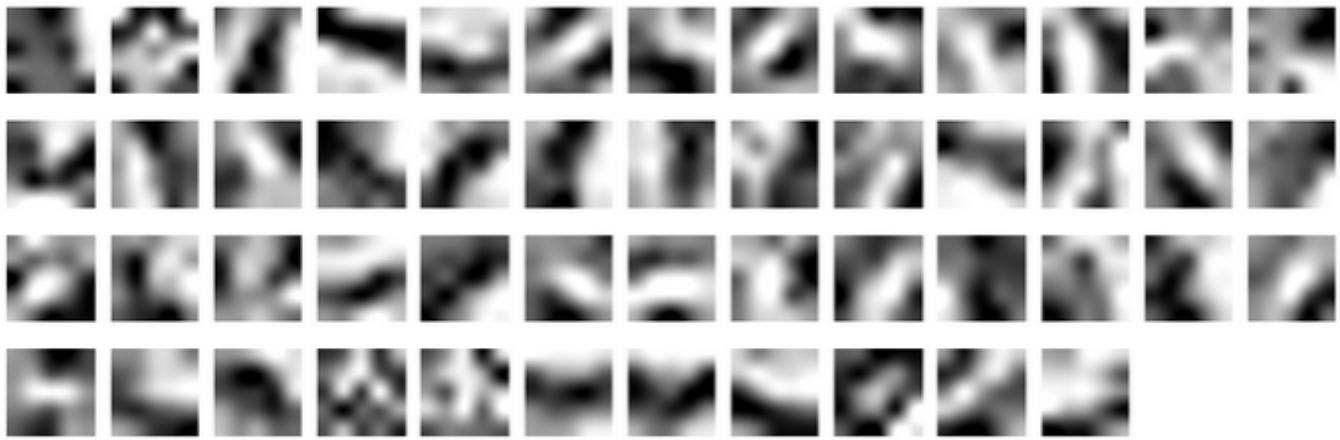


Abbildung : Filtermasken der ersten Schicht eines trainierten LeNet 5+

Neuronen-Visualisierung

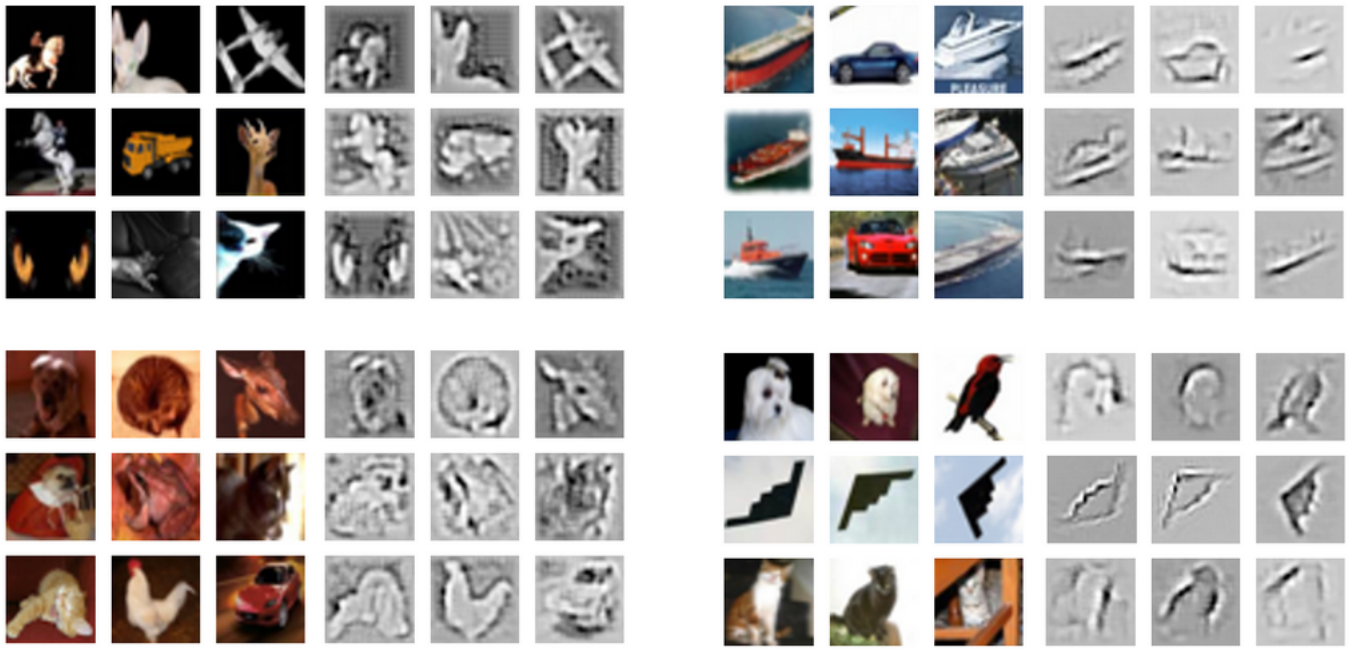


Abbildung : Visualisierung verschiedener Neuronen in der ersten Schicht (links) und in der zweiten Schicht (rechts) mittels der Methode von Zeiler und Fergus (2014)

Autoencoder-Visualisierung

Abbildung : Visualisierung kombinierter Merkmalsvektoren mittels Autoencoder

t-SNE-Transformation MNIST

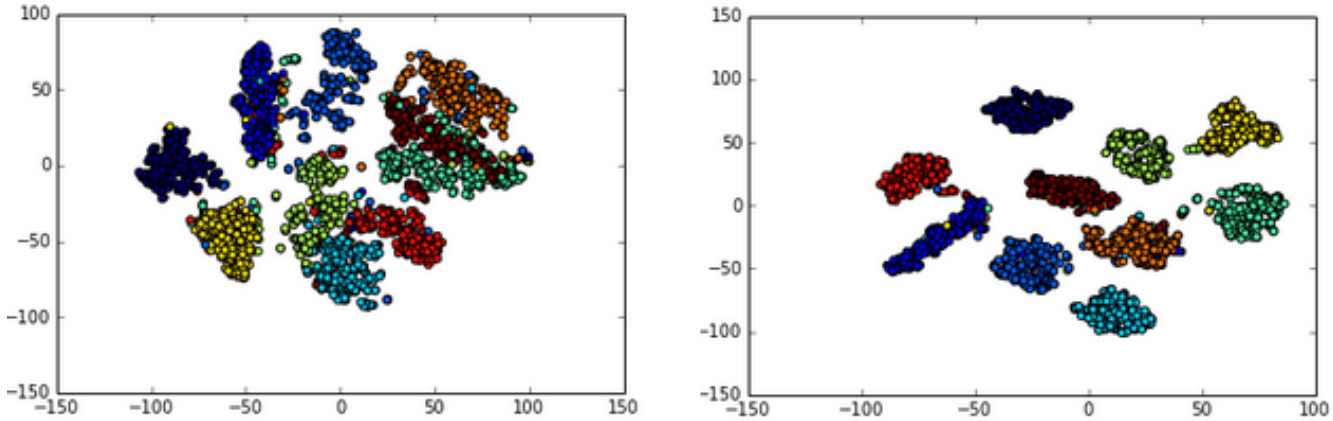


Abbildung : t-SNE-Encoding mit Originale (links) und mit CNN-Codes (rechts)

Ergebnisse und Ausblick

Ergebnisse I

Fehlerraten mit ConvNetCPP-Prototyp (AdaDelta-Methode)

- 0.61 % mit MNIST-Datensatz (12 Epochen)
- 25.02 % mit CIFAR-10-Datensatz (Padding) (72 Epochen)

- Padding erhöht die Kapazität und verbessert die Performanz ohne markant größeren Rechenaufwand.
- Empfehlenswerte Optimierungsverfahren sind Nesterov-Momentum und AdaDelta-Methode.
- Dropout und L2-Regularisierung sind gute Verfahren zur Verbesserung der Generalisierung.

Ergebnisse II

- Die Anwendung einer starken Regularisierung verlängert die Trainingszeit immens.
- Ein kleiner dimensioniertes Netz kann gleichwertige Ergebnisse wie ein großes, nicht regularisiertes Netz erzielen.
- Methoden zur Visualisierung sind gute Werkzeuge, um erlernte Merkmale sichtbar zu machen und die Funktionsweise von CNNs zu erklären.

Code Profiling

Gemessene Laufzeiten des *ConvNetCPP*-Prototyps:

- LeNet 5+ (MNIST) $\approx 1h$ CPU-Zeit pro Epoche
- Net-7 (CIFAR-10) $\approx 3h$ CPU-Zeit pro Epoche

Laufzeitverhalten

GEMM	52 %
`img2col(·)`	18 %
{=}-Operator	12 %
Speicherallokation	8 %
`krnl2col(·)`	5 %
Sonstiges	5 %

General Matrix Multiply (GEMM)

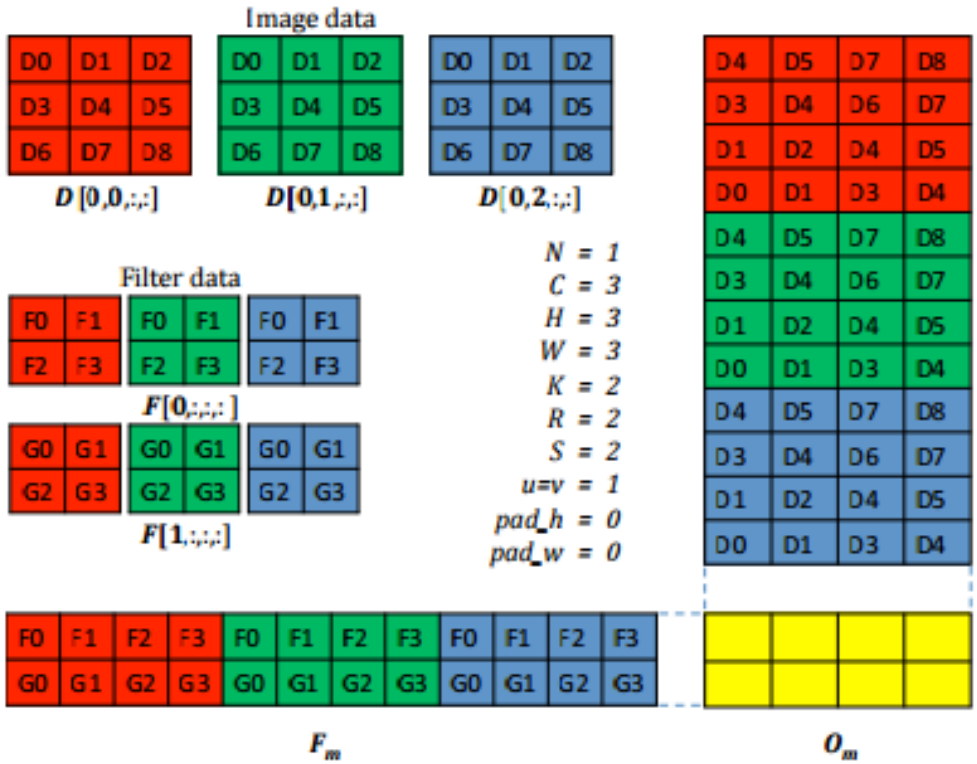


Abbildung : Reduktion der Faltung auf eine Matrix-Matrix-Multiplikation (GEMM) (Chetlur et al. (2014))

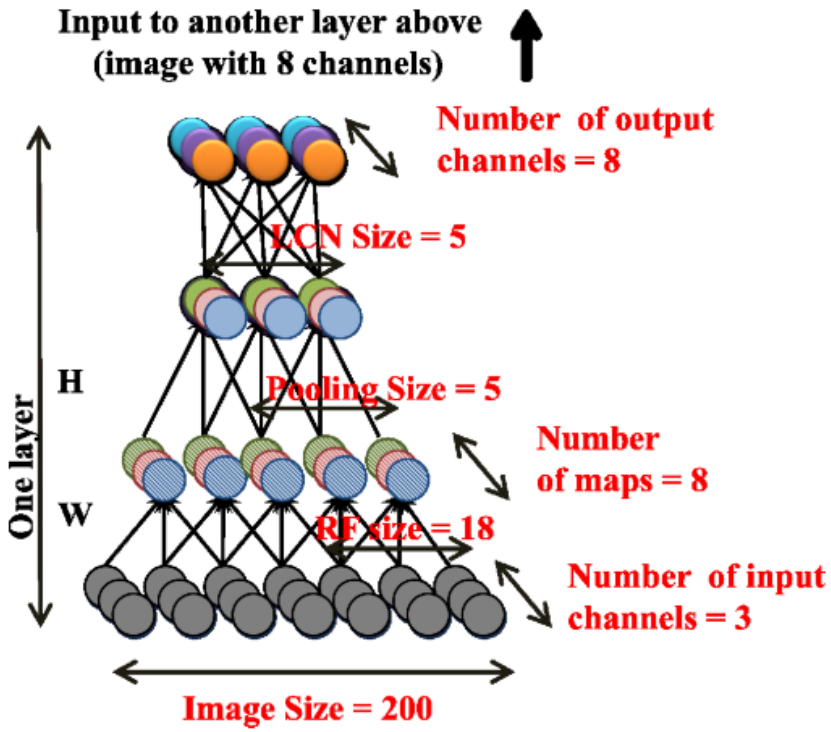
Empfehlung für den Einsatz

Empfohlene Vorgehensweise

1. Mit möglichst kleinem Netz starten
2. Sukzessive Erweiterung des Netzes bis der Validierungsfehler nicht weiter sinkt
3. Angleichung von Trainings- und Validierungsfehler mittels Techniken der Regularisierung für bestmögliche Generalisierung

Ausblick Prototyp

- GPU-Implementierung
- LC-Layer (Abb. rechts) (Le et al. (2012))
- Max-Out-Netze (Goodfellow et al. (2013))
- Stacked Denoising Autoencoder (Vincent et al. (2010))



Ausblick Forschungsfeld

- Untersuchung von Transferlernen
  (*Bell und Bala (2015) verwenden als Basis ein AlexNet*)
- Überwachung des Trainings
  (*Lee et al. (2014) verwenden SVMs pro Schicht*)
- Effizientere Verfahren zur Bestimmung von Hyperparametern
- Entwicklung anderer Verfahren für das Lernen verteilter Repräsentationen

Diskussionsrunde

Literatur I

Abdel-Hamid, O., Deng, L., und Yu, D. (2013). Exploring convolutional neural network structures and optimization techniques for speech recognition. In Bimbot, F., Cerisara, C., Fougeron, C., Gravier, G., Lamel, L., Pellegrino, F., und Perrier, P., Hrsg., *Procceedings of the International Conference of the 14th International Speech Communication Association, Interspeech*. ISCA.

Bell, S. und Bala, K. (2015). Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics*, 34(4):98:1–98:10.

Literatur II

Bengio, Y., Goodfellow, I. J., und Courville, A. (2015). Deep learning. Buch in Vorbereitung für MIT Press. URL: `http://www.iro.umontreal.ca/~bengioy/dlbook/` (15.08.2015).

Bengio, Y., Lamblin, P., Popovici, D., und Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., und Hoffman, T., Hrsg., *Proceedings of the 20th International Conference on Advances in Neural Information Processing Systems, NIPS*, Seiten 153–160. MIT Press. ISBN: 9781622760381.

Literatur III

Bengio, Y. und LeCun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., und Weston, J., Hrsg., *Large Scale Kernel Machines*. MIT Press. ISBN: 9780262026253.

Bengio, Y. und LeCun, Y. (2009). Learning deep architecture. In *Proceedings of the ICML Workshop on Learning Feature Hierarchies*. ICML.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., und Shelhamer, E. (2014). cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759.

Literatur IV

Dauphin, Y., Pascanu, R., Gülçehre, Ç., Cho, K., Ganguli, S., und Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572.

Dauphin, Y. N., de Vries, H., Chung, J., und Bengio, Y. (2015). Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390.

Fei-Fei, L. und Karpathy, A. (2014). CS231n Convolutional Neural Networks for Visual Recognition, University of Stanford. Course Material. URL: `http://cs231n.github.io/` (14.08.2015).

Literatur V

Glorot, X. und Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. und Titterington, M., Hrsg., *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, AISTATS*, Seiten 249–256. JMLR W&CP.

Glorot, X., Bordes, A., und Bengio, Y. (2011). Deep sparse rectifier neural networks. In Teh, Y. W. und Titterington, M., Hrsg., *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, AISTATS*, Seiten 315–323. JMLR W&CP.

Literatur VI

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., und Bengio, Y. (2013). Maxout networks. In Lawrence, N. und Reid, M., Hrsg., *Proceedings of the 30th International Conference on Machine Learning, ICML*, Seiten 1319–1327. JMLR W&CP.

Hinton, G. E., Osindero, S., und Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., und Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

Literatur VII

Hinton, G. E., Srivastava, N., und Swersky, K. (2015). CSC321 Neural Networks for Machine Learning, University of Toronto. Course Material. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (14.08.2015).

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. Thesis. URL: http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf (15.08.2015).

Hochreiter, S. und Schmidhuber, J. (1997). Long short-term memory. *Neural Computing*, 9(8):1735–1780.

Literatur VIII

Hubel, D. H. und Wiesel, T. N. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.

Krizhevsky, A., Sutskever, I., und Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., und Weinberger, K., Hrsg., *Proceedings of the 25th International Conference on Advances in neural information processing systems, NIPS*, Seiten 1097–1105. MIT Press. ISBN: 9781627480031.

Literatur IX

Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., und Ng, A. (2012). Building high-level features using large scale unsupervised learning. In Langford, J. und Pineau, J., Hrsg., *Proceedings of the 29th International Conference on Machine Learning, ICML*. ACM. ISBN: 9781450312851.

LeCun, Y., Bottou, L., Bengio, Y., und Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y. A., Bottou, L., Orr, G. B., und Müller, K.-R. (1998b). Efficient backprop. In Montavon, G., Orr, G. B., und Müller, K.-R., Hrsg., *Neural networks: Tricks of the trade*, Seiten 9–48. Springer. ISBN: 9783642352898.

Literatur X

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., und Tu, Z. (2014). Deeply-supervised nets. *CoRR*, abs:1409.5185v2.

Martens, J. (2010). Deep learning via hessian-free optimization. In Fürnkranz, J. und Joachims, T., Hrsg., *Proceedings of the 27th International Conference on Machine Learning, ICML*. Omnipress.

Masci, J., Meier, U., Cireşan, D., und Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the International Conference on Artificial Neural Networks and Machine Learning, ICANN*, Seiten 52–59. Springer. ISBN: 9783642217357.

Literatur XI

McCulloch, W. und Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., und Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Nagi, J., Ducatelle, F., Caro, G. A. D., Cires, D., Meier, U., Giusti, A., Nagi, F., Schmidhuber, J., und Gambardella, L. M. (2011). Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *Proceedings of the IEEE International Conference on Signal and Image Processing Applications, ICSIPA*. IEEE.

Literatur XII

Ranzato, M., Poultney, C. S., Chopra, S., und Lecun, Y. (2006). Efficient Learning of Sparse Representations with an Energy-Based Model. In Bernhard Schölkopf, J. P. und Hofmann, T., Hrsg., *Proceedings of the 19th Conference on Neural Information Processing Systems, NIPS*, Seiten 1137–1144. MIT Press.

Riedmiller, M. und Braun, H. (1992). Rprop-a fast adaptive learning algorithm. In *Procceedings of the 7th International Symposium on Computer and Information Science, ISCIS*.

Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer. ISBN: 9783540605058.

Literatur XIII

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., und Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Vincent, P., Larochelle, H., Bengio, Y., und Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning, ICML*, Seiten 1096–1103. JMLR W&CP.

Literatur XIV

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., und Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11(Dec):3371–3408.

Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y. A., und Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML*. JMLR W&CP.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

Literatur XV

Zeiler, M. D. und Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557.

Zeiler, M. D. und Fergus, R. (2014). Visualizing and understanding convolutional networks. In Fleet, D. J., Pajdla, T., Schiele, B., und Tuytelaars, T., Hrsg., *Proceedings of the 13th European Conference on Computer Vision, ECCV*, Seiten 818–833. Springer. ISBN: 9783319105901.