Vorlesung 08: Autoencoder

BA-INF 153: Einführung in Deep Learning für Visual Computing

Prof. Dr. Reinhard Klein

Nils Wandel

Informatik II, Universität Bonn

19.06.2024

Wiederholung

Letzte Woche:

1. CNN - Architectures:
   1. AlexNet
   2. VGG
   3. ResNet
2. CNN - Visualisierung und Analyse
   1. Activations
   2. Nearest Neighbors im Feature-Space
   3. Occlusion Experiments
   4. Saliency Maps
   5. Feature Visualisierung
   6. Texture Synthesis
   7. Style Transfer
   8. Adversarial Attacks
3. Transfer-Learning

Today's Lecture:

Autoencoder

1. Definition und Komponenten
2. Variationen von Autoencodern
3. Anwendungen

Weitere Lektüre

- Chapter 14 über Autoencoders aus **Goodfellow-et-al-2016-Book**
- "Tutorial on Variational Autoencoders" von Carl Doersch (`https://arxiv.org/pdf/1606.05908.pdf`)

Part 1

Definition - Autoencoder

Autoencoder

Autoencoder sind spezielle Neuronale Netze, welche trainiert werden, um eine Eingabe zu reproduzieren.

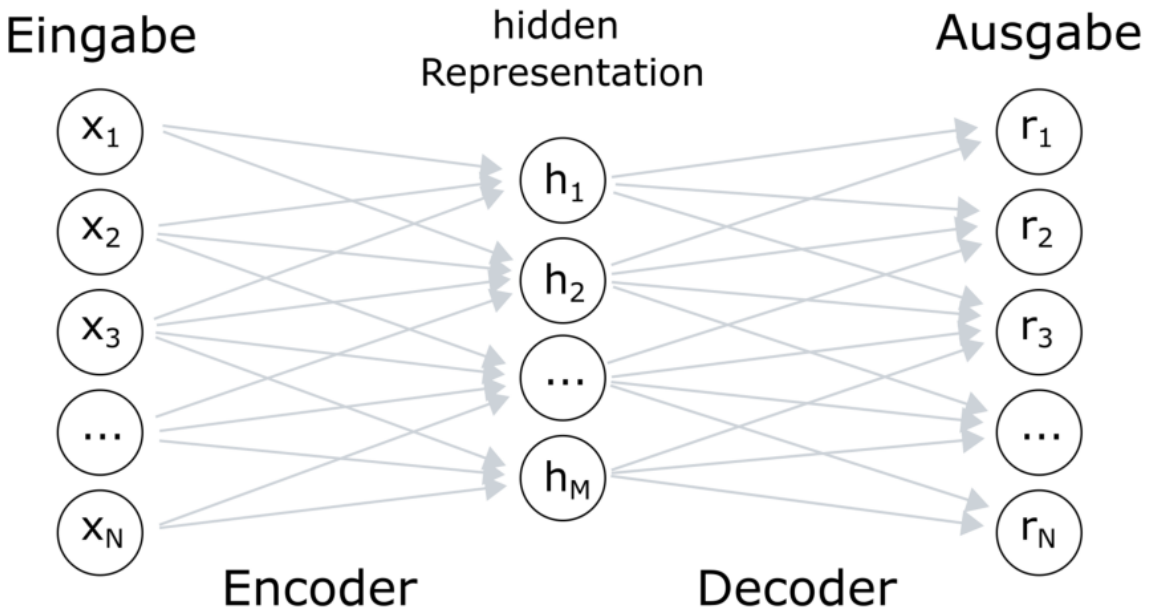


Figure: Autoencoder bestehen aus einem *Encoder*, welcher Eingaben x auf eine hidden Representation h abbildet und einem *Decoder*, welcher h wiederum auf Ausgaben r abbildet, welche die *Eingaben rekonstruieren* sollen.

Autoencoder

Autoencoder sind spezielle Neuronale Netze, welche trainiert werden, um eine Eingabe zu reproduzieren.

In einem standardmässigen Autoencoder gibt es ein hidden Layer $\boldsymbol{h} = f(\boldsymbol{x})$, welches den Input *codiert* bzw repräsentiert und ein Ausgabelayer $\boldsymbol{r} = g(\boldsymbol{h})$, welches h *decodiert* bzw eine Ausgabe aus der Repräsentation des hidden Layers rekonstruiert.

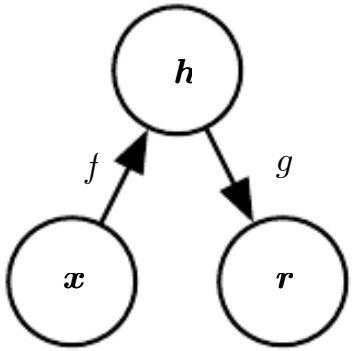


Figure 14.1: The general structure of an autoencoder, mapping an input $\boldsymbol{x}$ to an output (called reconstruction) $\boldsymbol{r}$ through an internal representation or code $\boldsymbol{h}$. The autoencoder has two components: the encoder f (mapping $\boldsymbol{x}$ to $\boldsymbol{h}$) and the decoder g (mapping $\boldsymbol{h}$ to $\boldsymbol{r}$).

Just Another Feedforward Network

Training

Autoencoder sind ein Spezialfall von Feedforward Netzwerken und können auf die selbe Art trainiert werden (z.B. Minibatch SGD + backpropagation). Das Training eines solchen Autoencoders geschieht durch Minimierung einer Lossfunktion zwischen dem ursprünglichen Bild $\boldsymbol{x}$ und dem rekonstruierten Bild $g(f(\boldsymbol{x}))$:

$$L\big(\boldsymbol{x}, g(f(\boldsymbol{x}))\big)$$

Als Fehler-Funktion L kann z.B. der L_2 oder L_1 Loss zwischen der Eingabe x und Rekonstruktion $r = g(f(\boldsymbol{x}))$ verwendet werden.

Just Another Feedforward Network

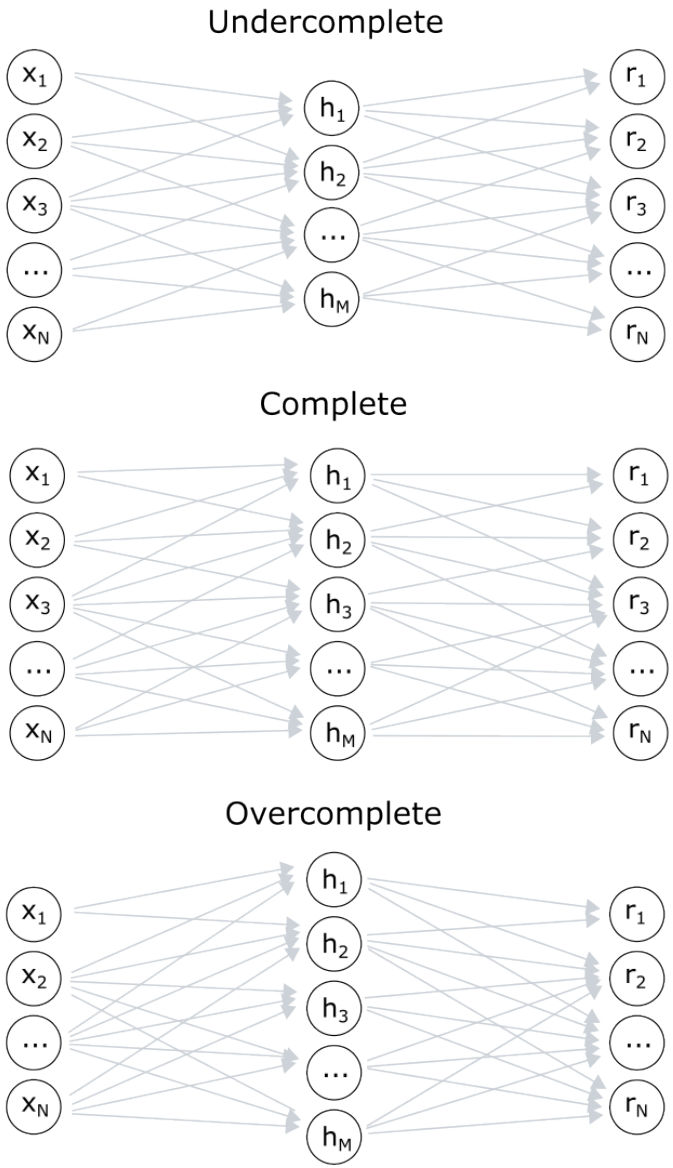
Ziel

Oft ist das Hauptinteresse bei Autoencodern gar nicht der rekonstruierte Output selbst, sonder die gelernten "hidden Representations", bzw der *code* $\boldsymbol{h}$, welcher nützliche Eigenschaften / Informationen besitzt.

Anwendungen

- Dimensionality Reduction
- Data Compression / Sparsity
- Denoising
- Self-Supervised Feature Learning
- Generative Modelle

Undercomplete, Complete und Overcomplete Autoencoder



Undercomplete, Complete und Overcomplete Autoencoder

Undercomplete autoencoders verwenden hidden Repräsentationen mit einer geringeren Dimensionalität als die Eingabedaten. Dadurch werden sie gezwungen, lediglich die wichtigsten Features von $\boldsymbol{x}$ zu extrahieren. In diesem Fall wird das hidden Layer oft auch als *"Bottleneck"-Layer* bezeichnet. Dieses kann hilfreich sein, um Daten zu komprimieren oder aussagekräftige Repräsentationen zu lernen.

Complete autoencoders verwenden hidden Repräsentationen mit der selben Dimensionalität wie die Eingabedaten.

Overcomplete autoencoders verwenden hidden Repräsentationen mit einer grösseren Dimensionalität als die Eingabedaten. In diesem Fall können auch Modelle mit geringer Kapazität (z.B. lineare Encoder / Decoder) die Daten perfekt Rekonstruieren, ohne nützliche Informationen zu extrahieren, indem die Identitätsabbildung gelernt wird. Dieses triviale Setup wäre jedoch ziemlich uninteressant. Deshalb werden Restriktionen z.B. in Form einer Regularisierung von $\boldsymbol{h}$ benötigt, damit der Decoder keine trivialen Kopien der Eingaben mehr erstellt, sondern dazu gezwungen wird, sinnvolle Eigenschaften der Daten zu lernen.

Part 5

Information retrieval

Autoencoders vs Principal Component Analysis (PCA)

Wenn der Decoder g linear ist und L dem MSE entspricht, dann spannt ein undercomplete Autoencoder den selben Unterraum wir eine Principle Component Analysis (PCA) auf.

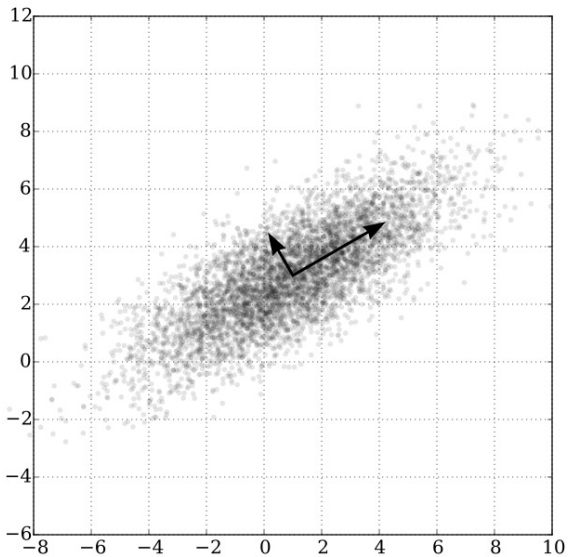


Figure: von [Wikipedia].

Die PCA besteht aus der Eigenwertzerlegung der Kovarianzmatrix. Dabei werden die Eigenvektoren der Kovarianzmatrix nach der Grösse der Eigenwerte sortiert.

Autoencoders vs PCA

Mit nicht-linearen Encodern und Decodern können mächtigere nicht-lineare Repräsentationen gelernt werden im Vergleich zur PCA.
Allerdings muss man einen sorgfältigen Trade-off finden:
Mit zu viel Modellkapazität könnte auch ein Autoencoder mit nur einer Dimension der hidden Representation eine Identitätsabbildung lernen ohne nützliche Informationen über die Daten zu extrahieren (siehe auch universal function approximation theorem). Dabei könnte der Encoder zum Beispiel eine Abbildung der Trainingsdaten $\boldsymbol{x}_i$ auf deren Indizes i lernen und der Decoder die inverse Abbildung der Indizes i zurück auf die Trainingsdaten x_i. Dies ist natürlich unerwünscht, da der Autoencoder somit das Trainingsset overfittet und weder nützlichen Informationen extrahieren noch generalisieren kann.
Mit zu wenig Modellkapazität kann der Autoencoder jedoch die Eingabedaten nicht mehr rekonstruieren.

Dimensionality Reduction und Datenkompression

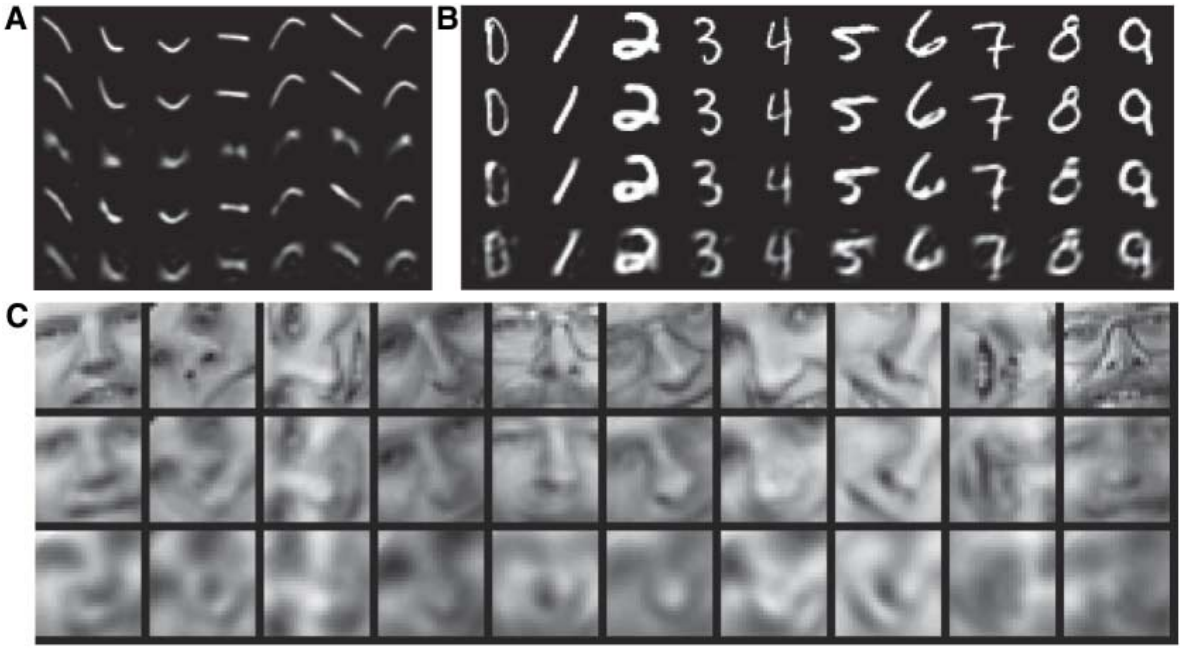


Fig. 2. (**A**) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (*8*) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (**B**) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (**C**) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

Figure: Vergleich von original Daten mit Rekonstruktionen mittels Autoencoder und PCA. Aus Hinton, G. E. und Salakhutdinov, R. (2006). Reduktion der Dimensionalität von Daten mit Hilfe von Neuronalen Netzen. Science, 313(5786).

Mit Hilfe von undercomplete Autoencodern lassen sich Daten sehr effektiv komprimieren. Dabei komprimiert der Encoder die Eingabe zu einer kompakteren hidden representation und der Decoder entpackt daraus wieder mit hoher Genauigkeit die ursprüngliche Eingabe.

Information Retrieval

In Information-Retrieval sollen Einträge einer Datenbank gefunden werden, welche einer Abfrage ähneln; diese Suche kann in dem niedrigdimensionalen Raum der hidden Repräsentationen effizient implementiert werden.

Wenn die Dimensionsreduktion des Autoencoders einen hidden Code produziert, der sowohl niedrigdimensional als auch binär ist, dann können die Daten in einem Hash-Table gespeichert werden, der die Code-Vektoren auf die Eingabebilder abbildet. Leicht unterschiedliche Einträge können ebenfalls effizient gefunden werden, indem einzelne Bits des Query-Codes geflipped werden.

Dieses *Semantic hashing* ermöglicht Information-Retrieval durch Dimensionsreduktion und Binarization und wurde bereits für Text und Bild-Eingaben verwendet.

Part 2

Regularisierte Autoencoder

Regularized Autoencoders

Im Idealfall würde für jede Autoencoderarchitektur die Dimensionalität der hidden Repräsentation derart gewählt werden, sodass sie die Komplexität der Verteilung der Eingabedaten wiederspiegelt.

Zu diesem Zweck können *regularisierte Autoencoder* verwendet werden, welche zusätzliche Regularisierungs-Terme im Loss verwenden um weitere Modelleigenschaften ausser dem Kopieren der Daten zu fordern.

Modelleigenschaften, welche wir fordern möchten, sind:

- Spärlichkeit der Repräsentation
- Robustheit bzgl Rauschen oder fehlenden Inputs

Ein gut regularisierter Autoencoder kann eine grosse Kapazität haben (d.h. nicht-linear / overcomplete hidden representation), aber dennoch nützliche Eigenschaften der zugrundeliegenden Datenverteilung lernen.

Sparse Autoencoders

Ein sparse Autoencoder fügt dem Loss einen Regularisierungs-Term $\Omega(\boldsymbol{h})$ für die hidden Representation $\boldsymbol{h}$ hinzu um Spärlichkeit zu fordern:

$$L\big(\boldsymbol{x}, g(f(\boldsymbol{x}))\big) + \Omega(\boldsymbol{h}).$$

Sparse Autoencoder werden typischerweise verwendet um Features für weitere Aufgaben (z.B. Klassifikation) zu lernen.

Im Gegensatz zu weight decay, wo der Regularisierungsterm auf die Modellgewichte angewendet wird und der Loss als MAP-Schätzer interpretiert werden kann, existiert eine solche Interpretation für sparse Autoencoder leider nicht direkt. Dennoch kann der Regularisierungsterm für Autoencoder als eine implizite Präferenz für Funktionen betrachtet werden.

Sparse Autoencoders

Der Sparsity Regularisierungs-Term sollte nicht als ein Penalty für die Rekonstruktion der Eingabe betrachtet werden. Er ist eher eine Annäherung für Maximum Likelihood Training eines generativen Modells mit latenten Variablen.

Betrachte ein Modell mit sichtbaren Variablen $\boldsymbol{x}$ und latenten Variablen $\boldsymbol{h}$ und der multivariaten Verteilung:

$$p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h}) = p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})\, p_{\text{model}}(\boldsymbol{h}).$$

Hier ist $p_{\text{model}}(\boldsymbol{h})$ ein Prior über die latenten Variablen und repräsentiert das Vorwissen des Modells über $\boldsymbol{h}$ bevor $\boldsymbol{x}$ gesehen wurde [1]

[1] Beachte, dass dies nicht dem Prior $p(\boldsymbol{\theta})$ über die Modellparameter im MAP-Schätzer entspricht.

Sparse Autoencoders

Die log-likelihood kann durch Marginalisierung über $\boldsymbol{h}$ der multivariaten Verteilung gefunden werden:

$$\log p_{\text{model}}(\boldsymbol{x}) = \log \sum_{h} p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h}).$$

Somit kann ein Autoencoder auch als eine Annäherung dieses Ausdrucks interpretiert werden, in dem nur ein einzelner, wahrscheinlicher Wert in der Summe für $\boldsymbol{h}$ berücksichtigt wird; Dieses gewählte $\boldsymbol{h}$ maximiert:

$$\log p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h}) = \underbrace{\log p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})}_{-L(x,g(f(x)))} + \underbrace{\log p_{\text{model}}(\boldsymbol{h})}_{-\Omega(\boldsymbol{h})}$$

Sparse Autoencoders

Durch geschickte Wahl von $\log p_{\text{model}}(\boldsymbol{h})$ kann Spärlichkeit induziert werden.

Zum Beispiel kann mit einem Laplace Prior

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} \exp(-\lambda |h_i|),$$

ein sparsity penalty für die Absolutewerte von $\boldsymbol{h}$ eingeführt werden.

Wenn wir Ω definieren als:

$$\Omega(\boldsymbol{h}) = \lambda \sum_i |h_i|, \qquad \text{erhalten wir}$$

$$-\log p_{\text{model}}(h) = \sum_i \left(\lambda |h_i| - \log \frac{\lambda}{2}\right) = \Omega(\boldsymbol{h}) + \text{const.}$$

Depth

Oft sind Autoencoder single-layer Architekturen, mit einem Layer für den Encoder und einem Layer für den Decoder.

Allerdings ist es möglich (und vorteilhaft!), mehrere Layer für den Encoder und Decoder zu verwenden. Dies ist wenig überraschend, da Autoencoder Feedforward Netzwerke sind, deren Mächtigkeit mit mehr Layern üblicherweise zunimmt.

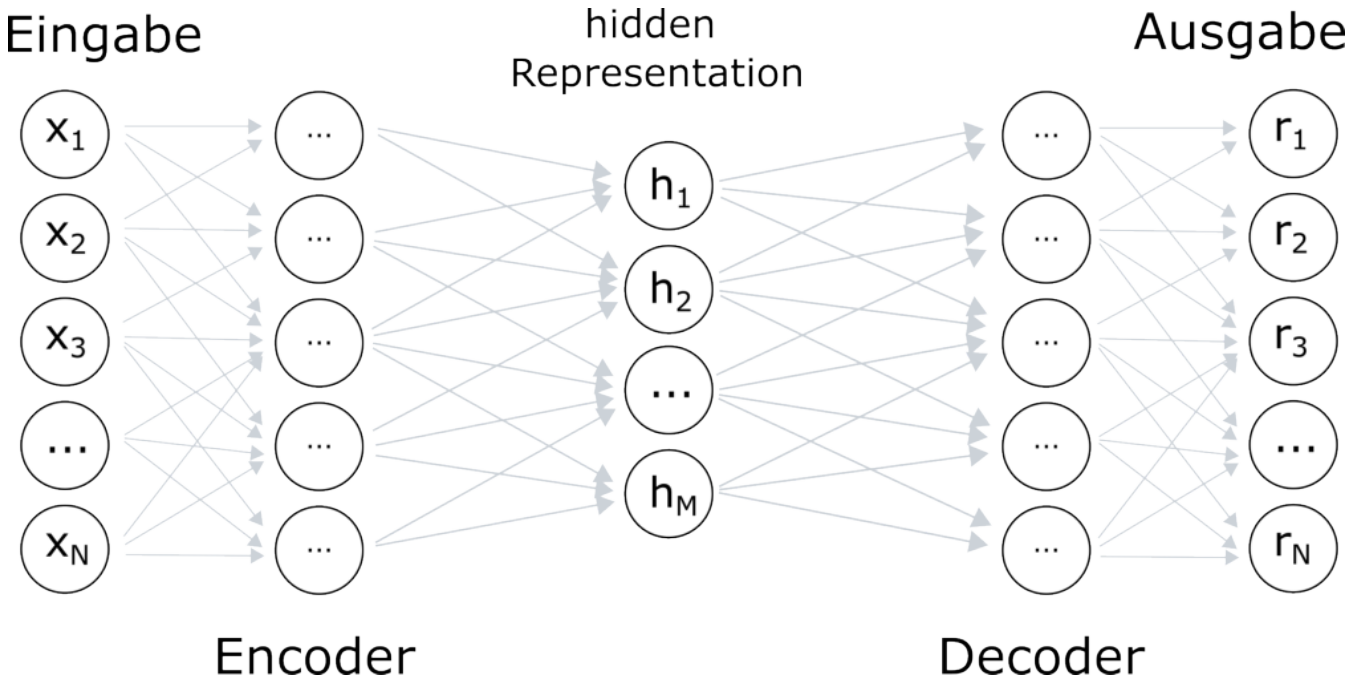


Figure: Autoencoder mit mehreren Layern können abstraktere Repräsentationen lernen.

Representation with Depth

Das universal function approximation theorem besagt, dass feedforward Netzwerke mit mindestens einem hidden Layer und genügend Units (zu einem gewissen Grad) beliebige Funktionen beliebig gut approximieren können. Ein Autoencoder mit einem hidden Layer kann somit die Identitätsabbildung entlang der Datendomäne beliebig gut approximieren.

Ein einziges hidden Layer ist jedoch flach sodass wir nicht beliebige Constraints formulieren können und gleichzeitig genügend Repräsentationsvermögen erhalten können. Hier helfen *tiefe Encoder und Decoder-Architekturen*, da sie sowohl den *Rechenaufwand* zur Repräsentation von Funktionen als auch die benötigte Menge an *Trainingsdaten exponentiell verringern* können.

Part 3

Denoising Autoencoder

Adding Noise

Der Regularisierungseffekt in sparse Autoencodern entsteht durch den Regularisierungs-Term $\Omega(\boldsymbol{h})$.

Eine weitere Möglichkeit der Regularisierung wird in *denoising Autoencodern* verfolgt.

Der Standard-Autoencoder minimiert einen Loss $L(\boldsymbol{x}, g\big(f(\boldsymbol{x})\big))$, welcher Abweichungen zwischen $g(f(\boldsymbol{x}))$ und $\boldsymbol{x}$ bestraft (z.B. durch die L_2-Norm). Mit genügend Modellkapazität lernt $g \cdot f$ jedoch lediglich die Identitäts-Abbildung.

Um dies zu verhindern minimiert der denoising Autoencoder:

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))).$$

wobei $\tilde{\boldsymbol{x}}$ eine verrauschte Version von $\boldsymbol{x}$ ist. Der Autoencoder muss also das Rauschen entfernen anstatt einfach nur die Eingabe auf die Ausgabe zu kopieren und wird somit implizit dazu gezwungen, auch die Struktur von $p_{\text{data}}(\boldsymbol{x})$ zu lernen.

Denoising Autoencoders

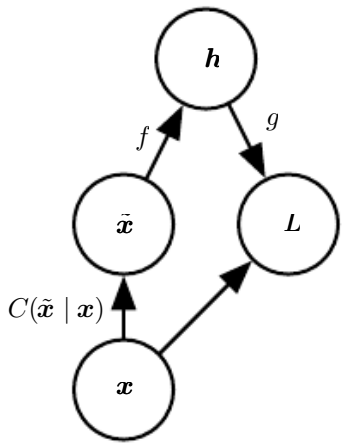


Figure 14.3: The computational graph of the cost function for a denoising autoencoder, which is trained to reconstruct the clean data point $\boldsymbol{x}$ from its corrupted version $\tilde{\boldsymbol{x}}$. This is accomplished by minimizing the loss $L = -\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$, where $\tilde{\boldsymbol{x}}$ is a corrupted version of the data example $\boldsymbol{x}$, obtained through a given corruption process $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$. Typically the distribution p_{decoder} is a factorial distribution whose mean parameters are emitted by a feedforward network g.

Figure: von Goodfellow

Denoising Autoencoders

Das Rauschen wird als ein Prozess $C(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ modelliert. Der Autoencoder lernt dann die Rekonstruktion der Verteilung $p_{\text{reconstruct}}(\boldsymbol{x}|\tilde{\boldsymbol{x}})$ basierend auf Trainings-Paaren $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$. Das Training eines denoising Autoencoders ist wie folgt:

1. Sample ein Trainingsbeispiel $\boldsymbol{x}$ aus den Trainingsdaten.
2. Sample eine verrauschte Version $\tilde{\boldsymbol{x}}$ aus $C(\tilde{\boldsymbol{x}}|\mathbf{x} = \boldsymbol{x})$
3. Trainiere einen Encoder f und einen Decoder g auf dem Paar $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$, wobei $p(\boldsymbol{x}|\tilde{\boldsymbol{x}}) = p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$, mit p_{decoder} durch $g(\boldsymbol{h})$ definiert ist.

Für den dritten Schritt können Gradienten-Abstiegsverfahren verwendet werden um $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ zu minimieren. Hier wurde angenommen, dass der Encoder $h = f(\tilde{\boldsymbol{x}})$ deterministisch ist.

Denoising AE Veranschaulichung

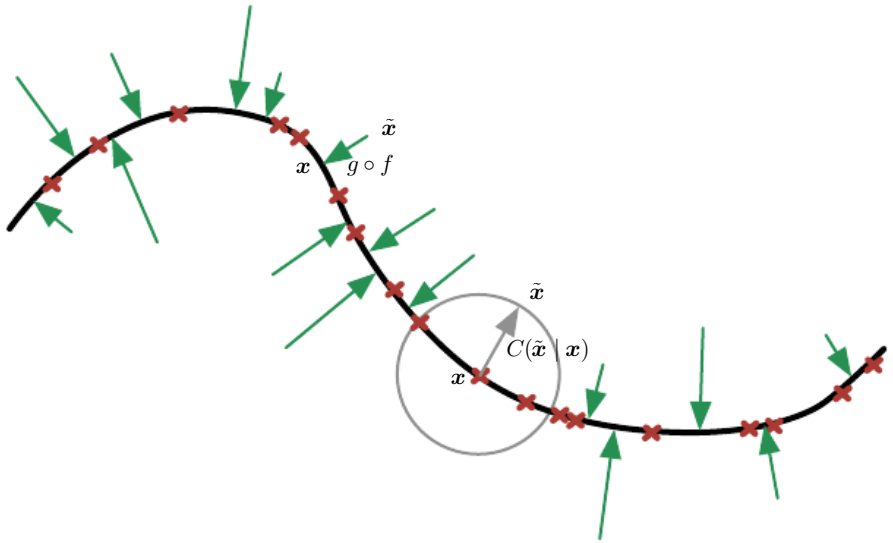
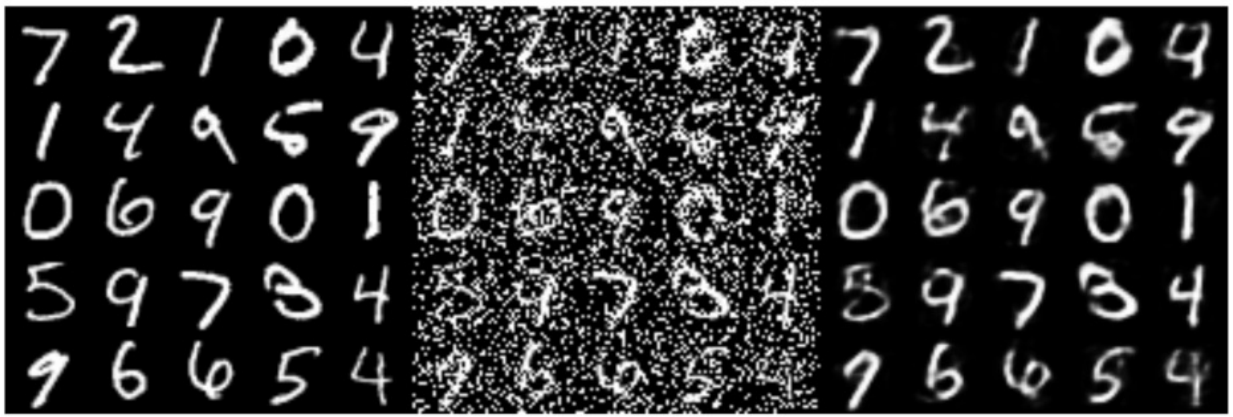


Figure 14.4: A denoising autoencoder is trained to map a corrupted data point $\tilde{\boldsymbol{x}}$ back to the original data point $\boldsymbol{x}$. We illustrate training examples $\boldsymbol{x}$ as red crosses lying near a low-dimensional manifold illustrated with the bold black line. We illustrate the corruption process $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$ with a gray circle of equiprobable corruptions. A gray arrow demonstrates how one training example is transformed into one sample from this corruption process. When the denoising autoencoder is trained to minimize the average of squared errors $||g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}||^2$, the reconstruction $g(f(\tilde{\boldsymbol{x}}))$ estimates $\mathbb{E}_{\mathbf{x},\tilde{\mathbf{x}} \sim p_{\text{data}}(\mathbf{x}) C(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{x} \mid \tilde{\boldsymbol{x}}]$. The vector $g(f(\tilde{\boldsymbol{x}})) - \tilde{\boldsymbol{x}}$ points approximately towards the nearest point on the manifold, since $g(f(\tilde{\boldsymbol{x}}))$ estimates the center of mass of the clean points $\boldsymbol{x}$ which could have given rise to $\tilde{\boldsymbol{x}}$. The autoencoder thus learns a vector field $g(f(\boldsymbol{x})) - \boldsymbol{x}$ indicated by the green arrows. This vector field estimates the score $\nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})$ up to a multiplicative factor that is the average root mean square reconstruction error.

Figure: von Goodfellow

Denoising AE Resultate



Original input, corrupted data and reconstructed data. Copyright by opendeep.org.

Figure: [Quelle]

Netzwerk-Initialisierung mit stacked denoising AE

Autoencoder können auch verwendet werden, um tiefe Netzwerke "vorzutrainieren". Das Training von tiefen Netzwerken ist eine nicht-triviale Aufgabe (siehe Schwierigkeiten, welche wir in der letzten Vorlesung diskutiert haben) und wir können den Trainingsprozess vereinfachen, indem wir einen vortrainings-Schritt hinzufügen.

Durch dieses Vortraining kann das Training eines tiefen Netzwerkes in folgende Phasen unterteilt werden (ähnlich wie bei Transfer-Learning):

- Vortraining: trainiere eine Sequenz von flachen denoising Autoencodern unsupervised und stapel diese Autoencoder aufeinander.
- Fine-tuning Schritt 1: Trainiere das letzte Layer supervised, wobei die Gewichte der Encoder-Layer fixiert bleiben.
- Fine-tuning Schritt 2: Propagiere die Gradienten durch das gesamte Netzwerk und trainiere alle Layer supervised.

Netzwerk-Initialisierung mit stacked denoising AE

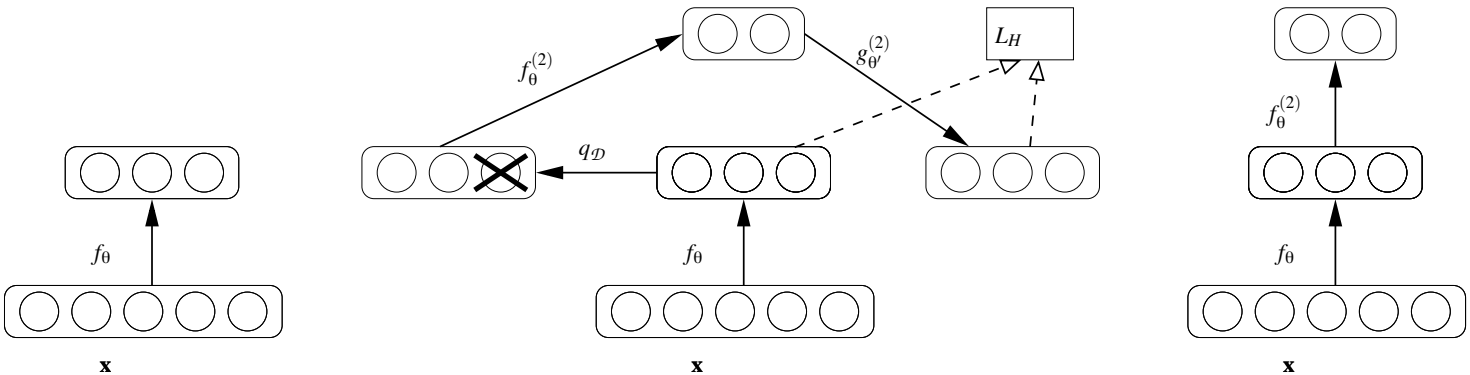


Figure 3: Stacking denoising autoencoders. After training a first level denoising autoencoder (see Figure 1) its learnt encoding function f_θ is used on clean input (left). The resulting representation is used to train a second level denoising autoencoder (middle) to learn a second level encoding function $f_\theta^{(2)}$. From there, the procedure can be repeated (right).

Figure: von Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." JMLR 11 (2010): 3371-3408. [pdf].

Netzwerk-Initialisierung mit stacked denoising AE

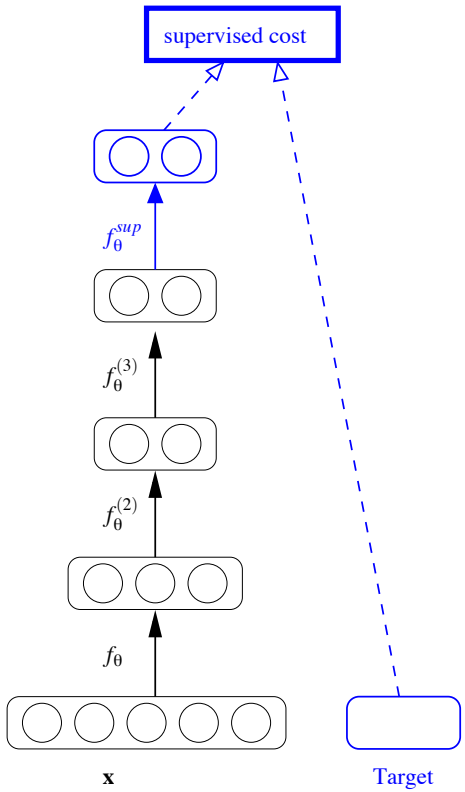


Figure 4: Fine-tuning of a deep network for classification. After training a stack of encoders as explained in the previous figure, an output layer is added on top of the stack. The parameters of the whole system are fine-tuned to minimize the error in predicting the supervised target (e.g., class), by performing gradient descent on a supervised cost.

Figure: von Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." JMLR 11 (2010): 3371-3408. [pdf].

Part 4

Stochastische Autoencoder und Variational Autoencoder

Nebenbemerkung zu Discriminativen vs. Generativen Methoden

Discriminative Modelle

Sei X eine Menge von Beobachtungsdaten und Y zugehörige Labels. In Klassifizierungsproblemen besteht die Inferenz daraus, ein Y^* zu finden, welches $P(Y|X')$ maximiert.
Eine *diskriminative* Methode modelliert $P(Y|X)$ *direkt* und bestimmt das Klassenlabel Y^* wie folgt:

$$Y^* = \underset{Y}{\operatorname{argmax}}\ P(Y|X).$$

Nebenbemerkung zu Discriminativen vs. Generativen Methoden

Generative Modelle

Eine *generative* Methode modelliert die bedingten Wahrscheinlichkeiten für die Klassen $P(X|Y)$ und benutzt Bayes' [2] zusammen mit dem Prior $P(Y)$ um dem Posterior $P(Y|X)$ zu finden.

Der Ausdruck *generativ* bezieht sich darauf, dass der Prior $P(Y)$ und die Likelihood $P(X|Y)$ benutzt werden kann um neue Daten zu "generieren" indem aus $P(X) = \sum_Y P(X|Y) \cdot P(Y)$ gesampled wird. Die Klassenlabel werden bestimmt durch:

$$Y^* = \underset{Y}{\operatorname{argmax}}\ P(Y|X) = \underset{Y}{\operatorname{argmax}}\ \frac{P(X|Y) \cdot P(Y)}{P(X)}.$$

Ein geeignetes $P(X|Y)$ kann üblicherweise mit Hilfe von gelabelten Daten gefunden werden. Allerdings ist die Berechnung von $P(X)$ schwierig.

[2] Hier wird Bayes nicht im Bezug auf die Modellparameter sondern im Bezug auf die hidden Representation verwendet.

Stochastic Autoencoders

Autoencoder funktionieren nicht nur mit deterministischen sondern auch mit probabilistischen Funktionen für den Encoder / Decoder. Dazu kann man die Encoder und Decoder Funktionen als stochastische Abbildungen für $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ und $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ betrachten. Somit kann fast jedes generative Modell mit einer latenten Variablen und der Fähigkeit, den latenten Code aus gegebenen Eingaben zu inferieren als Autoencoder betrachtet werden.

Stochastic Encoders and Decoders

Eine allgemeinere Strategie ist, eine Ausgabeverteilung $p(\boldsymbol{y}|\boldsymbol{x})$ mit Hilfe eines NN zu definieren und dann die negative log-likelihood zu minimieren. Da Autoencoder Feed-forward Netze sind, können wir die selben Ausgabeunits und Lossfunktionen wie vorher verwenden.

Loss

In einem Autoencoder ist der einzige Unterschied, dass $\boldsymbol{x}$ nun sowohl der Eingabe als auch dem Target der Ausgabe entspricht; alle weiteren Eigenschaften bleiben gleich.

Gegeben einem hidden Code $\boldsymbol{h}$, repräsentiert der Decoder eine bedingte Wahrscheinlichkeitsfunktion $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ und das Training besteht aus der Minimierung von $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$.

Stochastic Encoders and Decoders

Output

Die Ausgaben von Autoencodern werden analog zu standard Feed-forward Netzwerken verwendet.

Wenn $\boldsymbol{x}$ einer Normalverteilung folgt, dann werden üblicherweise lineare Output-Units verwendet um den Mean der Normalverteilung für $\boldsymbol{x}$ zu parametrisieren. In diesem Fall entspricht die Minimierung der negative log-likelihood dem mittleren quadrierten Fehler (MSE).

Wenn $\boldsymbol{x}$ einer Bernoulli-Verteilung entspricht, dann können Sigmoid Output-Units verwendet werden und wenn $\boldsymbol{x}$ einer Verteilung über mehrere Klassenlabels entspricht, dann kann Softmax verwendet werden. In diesen Fällen entspricht die Minimierung der negative log-likelihood der Kreuzentropie.

Stochastic Encoders and Decoders

Wir können auch für die Ausgabe des Encoders eine Wahrscheinlichkeitsverteilung $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ über die hidden Repräsentationen $\boldsymbol{h}$ verwenden.

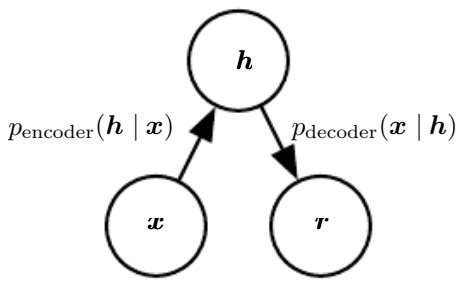


Figure 14.2: The structure of a stochastic autoencoder, in which both the encoder and the decoder are not simple functions but instead involve some noise injection, meaning that their output can be seen as sampled from a distribution, $p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$ for the encoder and $p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$ for the decoder.

Figure: von Goodfellow

Variational Autoencoder

In Variational Autoencodern wird auch die hidden Representation $\boldsymbol{h}$ als stochastisch betrachtet. Dabei gibt der Encoder keine deterministischen Werte vor, sondern die Mittelwerte und Varianzen von Normalverteilungen.

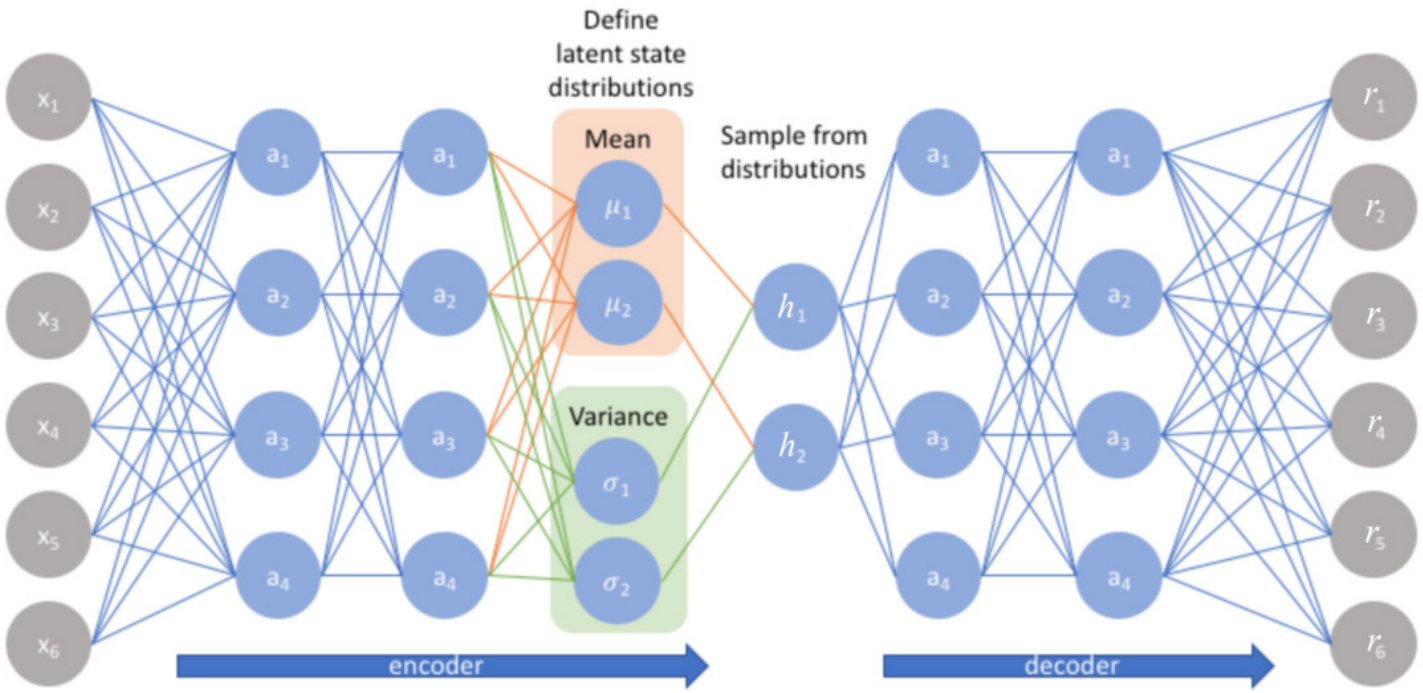


Figure: Architektur eines variational Autoencoders. [Quelle]. Der Encoder berechnet $q(h|x)$ und der Decoder berechnet $p(x|h)$. Die ausgegebenen r_i des Decoders könnten hierbei z.B. als Mittelwerte $\mu_{dec}(h)$ von Normalverteilungen mit konstanter Varianz σ^2_{dec} betrachtet werden.

Variational Autoencoder

Da Gradienten nicht ohne weiteres durch einen Sampling-Prozess propagiert werden können, muss ein Reparametrisierungs-Trick angewendet werden. Dabei skaliert und shiftet man Samples aus einer normierten Normalvertielung $\mathcal{N}(0, 1)$ mit den Ausgaben des Encoders σ und μ.

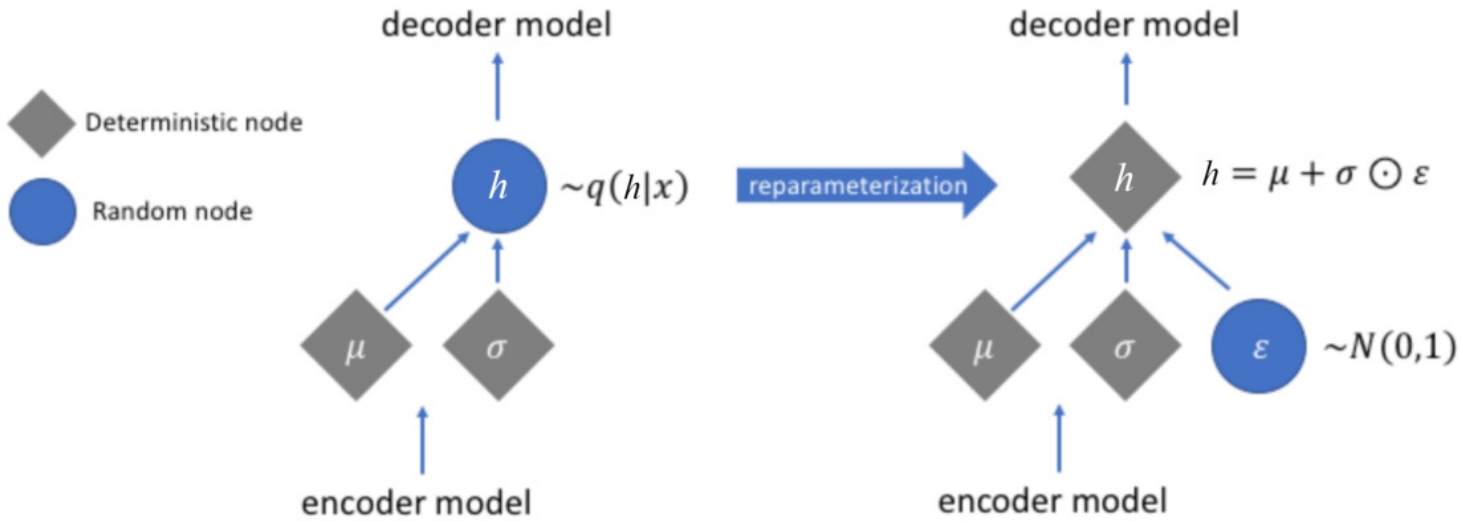


Figure: Reparametrisierungs-Trick in VAEs. [Quelle]

Variational Autoencoder

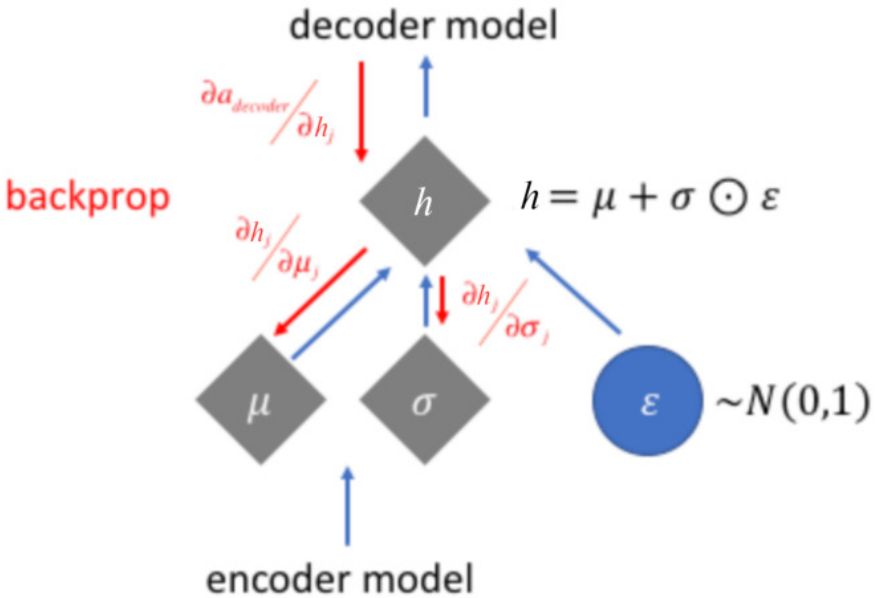


Figure: Durch den Reparametrisierungs-Trick können Gradienten bezüglich den Encoder-Ausgaben μ und σ berechnet werden. [Quelle]

Variational Autoencoder Loss

Der Loss eines Variational Autoencoders ergibt sich aus 2 Komponenten:

$$\mathcal{L} = \lambda \mathcal{L}(\boldsymbol{x}, \boldsymbol{r}) + D_{KL}(q(\boldsymbol{h}|\boldsymbol{x}), p(h))$$

Die *erste Komponente* ist der *Rekonstruktionsloss* und sorgt dafür, dass die Eingabedaten $\boldsymbol{x}$ durch die Rekonstruktionen $\boldsymbol{r}$ des Autoencoders korrekt reproduziert werden können. Dieser kann z.B. einem L_2 oder L_1 loss entsprechen. λ ist ein optionaler Hyperparameter.
Die *zweite Komponente* entspricht der Kullback-Leibler-Divergenz zwischen der Wahrscheinlichkeitsfunktion der hidden Representation $q(\boldsymbol{h}|\boldsymbol{x})$ und $p(h)$, welche eine normierten Normalverteilung $\mathcal{N}(h|0,1)$ ist. Dadurch wird dafür gesorgt, dass die Verteilung der *hidden Representations einer normierten Normalverteilung folgen*.
Da $q(\boldsymbol{h}|\boldsymbol{x})$ einer Normalverteilung $\mathcal{N}(\boldsymbol{h}|\mu_i, \sigma_i)$ mit den Netzwerkausgaben μ_i, σ_i entspricht, kann D_{KL} analytisch berechnet werden:

$$\begin{aligned} D_{KL}(q(\boldsymbol{h}|\boldsymbol{x}), \mathcal{N}(\boldsymbol{h}|0,1)) &= \int_{\boldsymbol{h}} \mathcal{N}(\boldsymbol{h}|\mu_i, \sigma_i^2) \log\left(\frac{\mathcal{N}(\boldsymbol{h}|\mu_i, \sigma_i^2)}{\mathcal{N}(\boldsymbol{h}|0,1)}\right) \\ &= \frac{1}{2}(-\sum_i(\log(\sigma_i^2)+1) + \sum_i \sigma_i^2 + \sum_i \mu_i^2) \end{aligned}$$

Variational Autoencoder Loss

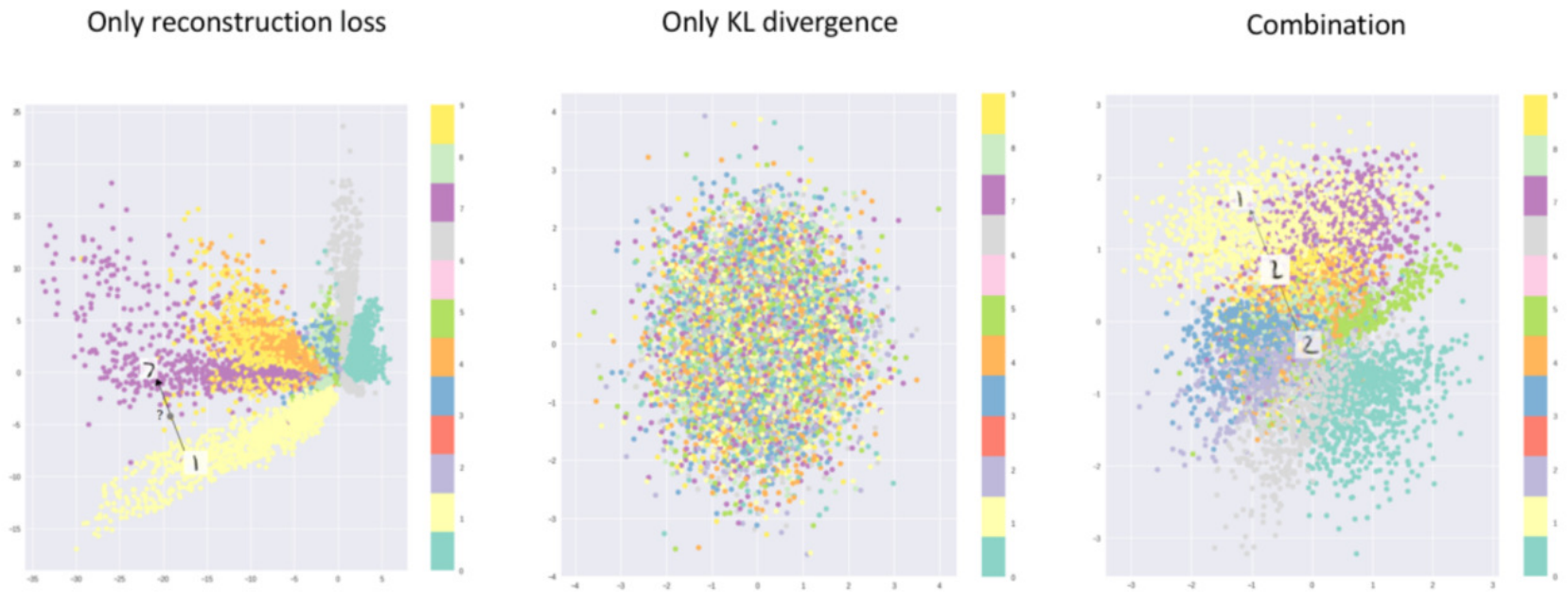


Figure: Einfluss der Loss-Komponenten auf die hidden Representation eines VAE für das MNIST datenset. [Quelle]

Variational Autoencoder: Resultat auf MNIST

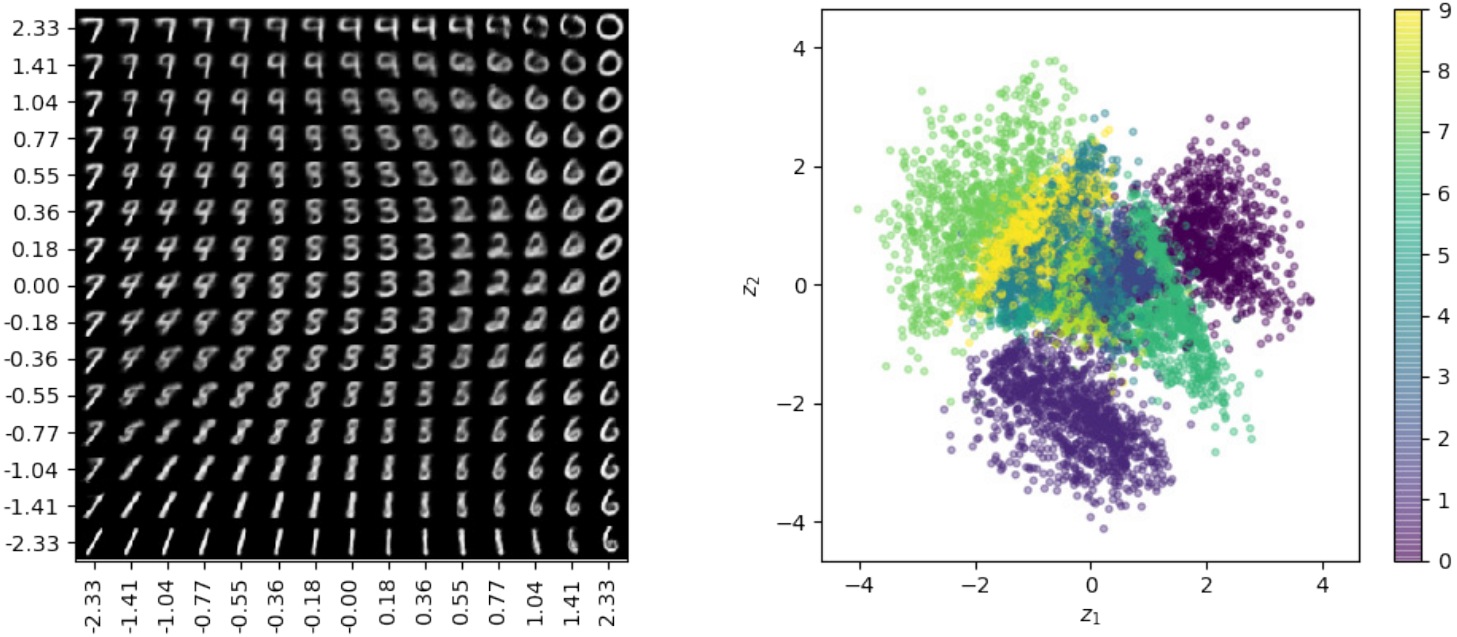


Figure: Resultat eines VAE mit einer 2 dimensionalen hidden Representation auf dem MNIST datenset. [Quelle]

Variational Autoencoder Loss

Der Loss für Variational Autoencoder lässt sich mit Hilfe der "Evidence Lower-Bound" (ELBO) formal herleiten.

Schritt 1

Zunächst betrachten wir nur den Decoder. Dieser soll einer bedingten Wahrscheinlichkeitsfunktion $p(x|h)$ entsprechen, welche für eine gegebene hidden Representation h eine Wahrscheinlichkeit über die Ausgabedaten (z.B. Bilder) zurückgeben soll. $p(x|h)$ könnte z.B. durch eine Normalverteilung gegeben sein, dessen Mittelwert von einem neuronalen Netz $\mu_{dec}(h)$ berechnet wird:

$$p(x|h) = \mathcal{N}(x|\mu_{dec}(h), \sigma^2_{dec})$$

σ_{dec} kann hier als konstant angenommen werden. Die Verteilung über die hidden Representations soll einer normierten Normalverteilung entsprechen:

$$p(h) = \mathcal{N}(h|0, 1)$$

Damit könnten wir neue Samples aus $p(x)$ generieren, indem wir über h marginalisieren:

$$p(x) = \int p(x|h)p(h)dh$$

Variational Autoencoder Loss

Der Loss für Variational Autoencoder lässt sich mit Hilfe der "Evidence Lower-Bound" (ELBO) formal herleiten.

Schritt 2

Damit die neu generierten Samples möglichst gut der originalen Verteilung über die Daten $X = \{x_1, ..., x_n\}$ entspricht, möchten wir $p(X) = \prod_{x \in X} p(x)$ maximieren. Das heisst, wir müssen die Evidenz:

$$p(X) = \prod_{x \in X} \int p(x|h)p(h)dh$$

maximieren. Problem: Dieser Ausdruck ist nur sehr schwer zu berechnen, weil das Integral $p(x) = \int p(x|h)p(h)dh$ nur schwer zu berechnen ist (Wenn wir das Integral durch Sampling nach der Verteilung $p(h)$ berechnen möchten, dann ist $p(x|h)$ für die meisten Werte $= 0$).
$\Rightarrow$ Idee: könnten wir vielleicht h derart samplen, dass $p(x|h)$ nicht so häufig kleine Werte liefert? (D.h. statt $p(h)$ würden wir lieber $p(h|x)$ samplen).
$\Rightarrow$ Mit ein paar Tricks ist das möglich...

Variational Autoencoder Loss

Der Loss für Variational Autoencoder lässt sich mit Hilfe der "Evidence Lower-Bound" (ELBO) formal herleiten.

Schritt 3

Anstatt $p(X)$ zu maximieren, verwenden wir wieder den log-Trick und maximieren stattdessen $\log(p(X))$.

$$\log(p(X)) = \log(\prod_{x \in X} p(x)) = \sum_{x \in X} \log(p(x))$$

Dies ist möglich, das der log monoton steigend ist. Anstatt nun $p(x) = \int p(x|h)p(h)dh$ zu setzen, fügen wir einen weiteren Term hinzu und maximieren stattdessen:

$$\sum_{x \in X} \underbrace{\log(p(x)) - \mathcal{D}_{KL}(q(h|x), p(h|x))}_{\text{ELBO = evidence lower bound}}$$

Da die Kullback-Leibler Divergenz $\mathcal{D}_{KL}$ immer ≥ 0 ist, und $= 0$, falls $q(h|x) = p(h|x)$ gilt, stellt das eine untere Schranke für den logarithmus der Evidenz dar. Wenn wir diesen Ausdruck maximieren und sich $q(h|x)$ dann $p(h|x)$ annähert, geht der $\mathcal{D}_{KL}$-Term gegen 0 und wir maximieren wieder den originalen $\log(p(x))$ Term. Aber was gewinnen wir aus dieser scheinbaren Verkomplifizierung?

Variational Autoencoder Loss

Der Loss für Variational Autoencoder lässt sich mit Hilfe der "Evidence Lower-Bound" (ELBO) formal herleiten.

Schritt 4

... wir können den ELBO wie folgt vereinfachen:

$$\begin{aligned}\text{ELBO} &= \log(p(x)) - \mathcal{D}_{KL}(q(h|x), p(h|x)) \\ &= \log(p(x)) - \int q(h|x)\left(\log(q(h|x)) - \log(p(h|x))\right) dh\end{aligned}$$

Verwende Satz von Bayes: $p(h|x) = \frac{p(x|h)p(h)}{p(x)}$

$$= \underbrace{\log(p(x))}_{\text{cancel}} - \int q(h|x)\left(\log(q(h|x)) - \log(p(x|h)) - \log(p(h)) + \underbrace{\log(p(x))}_{\text{cancel}}\right) dh$$

$$= \int q(h|x)\log(p(x|h))dh - \underbrace{\int q(h|x)\left(\log(q(h|x)) - \log(p(h))\right) dh}_{\mathcal{D}_{KL}(q(h|x),p(h))\Rightarrow\text{analytische Lösung s.o.}}$$

Variational Autoencoder Loss

Der Loss für Variational Autoencoder lässt sich mit Hilfe der "Evidence Lower-Bound" (ELBO) formal herleiten.

Schritt 5

Anstatt den ELBO zu maximieren, wird üblicherweise der negative ELBO minimiert, wodurch wir bei der ursprünglichen Loss-Funktion angelangt wären:

$$\mathcal{L} = \underbrace{-\int q(h|x)\log(p(x|h))dh}_{\lambda\mathcal{L}(\boldsymbol{x},\boldsymbol{r})} + D_{KL}(q(\boldsymbol{h}|\boldsymbol{x}), p(h))$$

Im Vergleich zum ursprünglichen Integral ($p(x) = \int p(x|h)p(h)dh$), kann $\int q(h|x)\log(p(x|h))dh$ effizient gesamplet werden, weil sich $q(h|x)$ an $p(h|x)$ annähert. Dazu wird für ein gegebenes Eingabebild x mittels des Encoder-Netzwerkes zunächst μ_i und σ_i für $q(h|x) = \mathcal{N}(h|\mu_i, \sigma_i)$ berechnet. Aus dieser Verteilung können dann mit Hilfe des Reparametrisierungs-Tricks differenzierbar Werte für h gezogen werden. Diese werden dann im Decoder verwendet, um $p(x|h)$ zu berechnen. Wenn wir für $p(x|h)$ eine Normalverteilung verwenden (siehe Schritt 1), dann kommen wir - ganz ähnlich wie bei unseren früheren Berechnungen zur NLL - auf einen L_2 loss für $\mathcal{L}(x, r)$ mit $r = \mu_{dec}(h)$.

Variational Autoencoder

Vorteile

- Variational Autoencoder sorgen dafür, dass sich Punkte, die in der hidden Representation räumlich nahe liegen auch im Ausgaberaum ähnlich sind
- Mit VAE können neue Samples der Datenverteilung generiert werden, indem hidden Representations aus der Normalverteilung $\mathcal{N}(0, 1)$ gezogen werden $\Rightarrow$ somit sind VAEs generative Modelle.

Nachteile

- Die Ausgabebilder von VAEs sind üblicherweise etwas unschärfer als von Standard-Autoencodern

Part 6

U-Net

U-Net

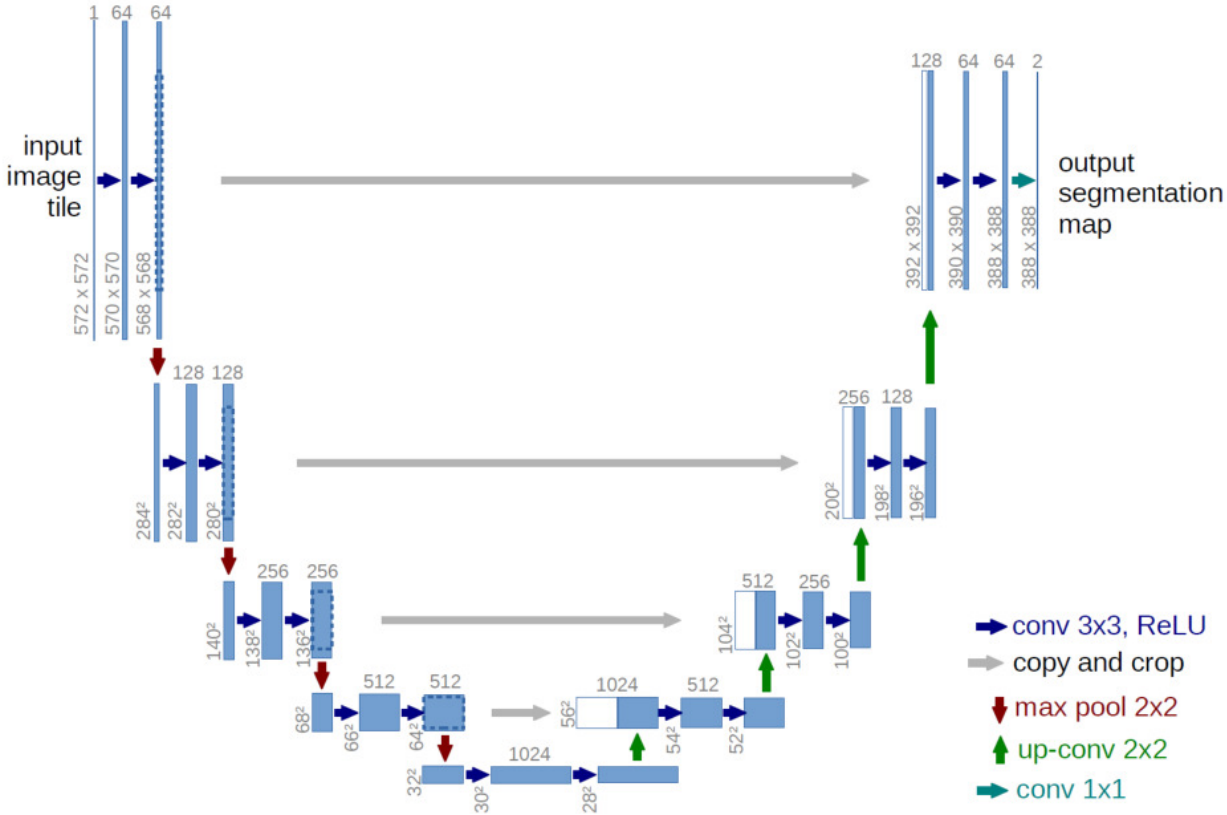


Figure: Das U-Net besitzt eine ähnliche Architektur wie ein Autoencoder. Zunächst erstellt ein Encoder schrittweise eine immer abstraktere Repräsentation des Eingabebildes, welche dann durch einen Decoder auf die Ausgabe abgebildet wird. Allerdings erlauben zusätzliche Shortcut-Verbindungen zwischen dem Decoder und dem Encoder auf verschiedenen Auflösungs-Ebenen, dass auch hochfrequente Details des Eingabebildes erhalten bleiben. [Quelle]

../deeplearn