# I Neuronale Netze und Deep Learning

# 1 Einbettungen, latenter Raum und Repräsentationen

**Beim Deep Learning sind Begriffe wie *Einbettungsvektoren*, *Repräsentationen* und *latenter Raum* gebräuchlich. Was haben diese Konzepte gemeinsam und wie unterscheiden sie sich?**

---

Auch wenn diese drei Begriffe oft synonym verwendet werden, können wir zwischen ihnen feine Unterscheidungen treffen:

- Einbettungsvektoren sind Repräsentationen von Eingabedaten, bei denen ähnliche Elemente nahe beieinanderliegen.
- Latente Vektoren sind Zwischenrepräsentationen von Eingabedaten.
- Repräsentationen sind codierte Versionen der ursprünglichen Eingabedaten.

Die folgenden Abschnitte untersuchen die Beziehung zwischen Einbettungen, latenten Vektoren und Repräsentationen. Außerdem erfahren Sie, wie sie jeweils im Kontext des maschinellen Lernens Informationen codieren.

## 1.1 Einbettungen

Einbettungsvektoren – kurz *Einbettungen* – codieren relativ hochdimensionale Daten in relativ niedrigdimensionale Vektoren.

Mithilfe von Einbettungsmethoden können wir einen kontinuierlichen dichten (nicht-dünnbesetzten) Vektor aus einer (dünnbesetzten) 1-aus-n-Codierung (englisch One-hot encoding) erzeugen. Die *1-aus-n-Codierung* ist eine Methode, um kategoriale Daten als binäre Vektoren darzustellen, wobei jede Kategorie auf einen Vektor abgebildet wird, der an der Position, die dem Index der Kategorie entspricht, eine 1, und an allen anderen Positionen eine 0 enthält. Damit ist sichergestellt, dass die kategorialen Werte so dargestellt werden, dass bestimmte Algorithmen für maschinelles Lernen sie verarbeiten können. Wenn wir zum Beispiel eine kategoriale Variable `Farbe` mit den drei Kategorien `Rot`, `Grün` und `Blau` haben, stellt die 1-aus-n-Codierung `Rot` als `[1, 0, 0]`, `Grün` als `[0, 1, 0]` und `Blau` als `[0, 0, 1]` dar. Diese 1-aus-n-codierten kategorialen Variablen lassen sich dann in kon-

tinuierliche Einbettungsvektoren abbilden, indem die gelernte Gewichtsmatrix einer Einbettungsschicht oder eines Moduls verwendet wird.

Einbettungsmethoden eignen sich auch für dichte Daten wie Bilder. Zum Beispiel können die letzten Schichten eines Convolutional Neural Networks (CNN) Einbettungsvektoren liefern, wie Abbildung 1–1 veranschaulicht.

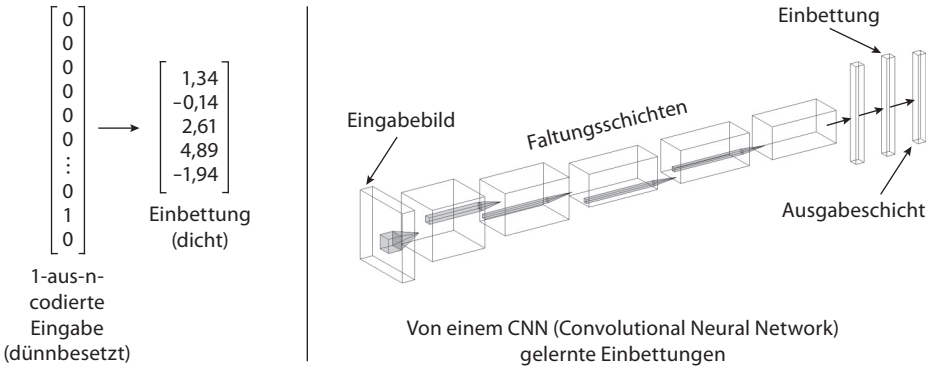


***Abb. 1–1*** *Eine Eingabeeinbettung (links) und eine Einbettung von einem neuronalen Netz (rechts)*

Um technisch korrekt zu sein, könnten alle Ausgaben der Zwischenschicht eines neuronalen Netzes Einbettungsvektoren liefern. Je nach Trainingsziel kann auch die Ausgabeschicht nützliche Einbettungsvektoren erzeugen. Der Einfachheit halber assoziiert das Convolutional Neural Network in Abbildung 1–1 die vorletzte Schicht mit Einbettungen.

Es ist möglich, dass Einbettungen eine höhere oder niedrigere Anzahl von Dimensionen haben als die ursprüngliche Eingabe. Mithilfe von Einbettungsmethoden für extreme Ausdrücke lassen sich beispielsweise Daten in zweidimensionale dichte und kontinuierliche Darstellungen für Visualisierungszwecke und Clustering-Analysen codieren, wie Abbildung 1–2 zeigt.

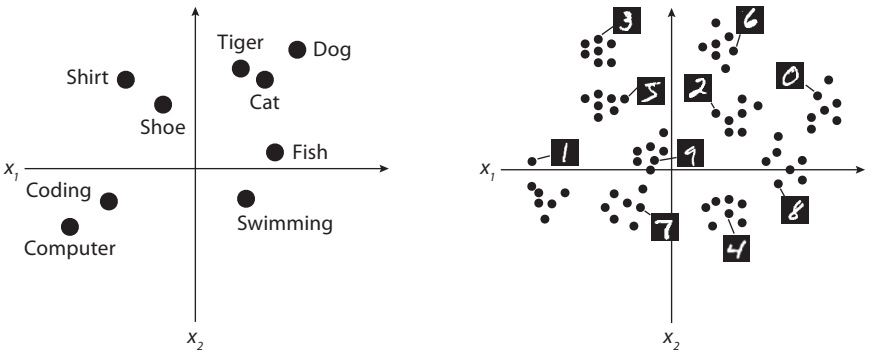


***Abb. 1–2*** *Abbildung von Wörtern (links) und Bildern (rechts) auf einen zweidimensionalen Merkmalsraum*

Zu den grundlegenden Eigenschaften von Einbettungen gehört, dass sie *Abstand* oder *Ähnlichkeit* codieren. Das bedeutet, dass Einbettungen die Semantik der Daten so erfassen, dass ähnliche Eingaben im Einbettungsraum nahe beieinanderliegen.

Für Leser, die an einer formaleren Erklärung mittels mathematischer Terminologie interessiert sind, sei erwähnt, dass eine Einbettung eine injektive und strukturerhaltende Abbildung zwischen einem Eingaberaum X und dem Einbettungsraum Y ist. Dies impliziert, dass ähnliche Eingaben an nahe beieinanderliegenden Punkten innerhalb des Einbettungsraums liegen, was man als »strukturerhaltende« Eigenschaft der Einbettung ansehen kann.

## 1.2 Latenter Raum

*Latenter Raum* wird in der Regel synonym mit *Einbettungsraum* verwendet, das heißt dem Raum, in den Einbettungsvektoren abgebildet werden.

Ähnliche Elemente können im latenten Raum nahe beieinanderliegen, was jedoch keine strikte Voraussetzung ist. Im weiteren Sinne kann man sich den latenten Raum als jeden Merkmalsraum vorstellen, der Features – oftmals komprimierte Versionen der ursprünglichen Eingabefeatures – enthält. Diese latenten Raumfeatures können von einem neuronalen Netzwerk erlernt werden, beispielsweise einem Autoencoder, der Eingabebilder rekonstruiert, wie Abbildung 1–1 zeigt. Diese latenten Eingabefeatures können von einem neuronalen Netz gelernt werden, beispielsweise von einem Autoencoder, der Eingabebilder rekonstruiert, wie Abbildung 1–3 zeigt.

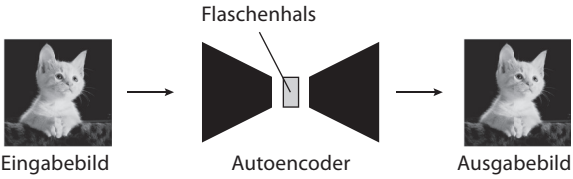


***Abb. 1–3*** *Ein Autoencoder, der das Eingabebild rekonstruiert*

Der Flaschenhals in Abbildung 1–3 repräsentiert eine kleine Zwischenschicht des neuronalen Netzes, die das Eingabebild in eine Repräsentation geringerer Dimensionen codiert. Den Zielraum dieser Abbildung kann man sich als latenten Raum vorstellen. Das Trainingsziel des Autoencoders besteht darin, das Eingabebild zu rekonstruieren, das heißt, den Abstand zwischen den Eingabe- und Ausgabebildern zu minimieren. Um das Trainingsziel zu optimieren, kann der Autoencoder lernen, die codierten Features von ähnlichen Eingaben (zum Beispiel Bilder von Katzen) im latenten Raum nahe nebeneinander zu platzieren, dabei nützliche Einbettungen von Vektoren zu erstellen, wobei ähnliche Eingaben im Einbettungs- (latenten) Raum nahe beieinanderliegen.

## 1.3 Repräsentation

Eine *Repräsentation* ist eine codierte Form und typischerweise eine Zwischenform einer Eingabe. Zum Beispiel ist ein Einbettungsvektor bzw. Vektor im latenten Raum eine Darstellung der Eingabe, wie oben erläutert. Allerdings können Darstellungen auch mit einfacheren Prozeduren erzeugt werden. Zum Beispiel lassen sich 1-aus-n-codierte Vektoren als Repräsentationen einer Eingabe betrachten, wie bereits oben erwähnt. Allerdings lassen sich Repräsentationen auch durch einfachere Verfahren erzeugen. So werden zum Beispiel 1-aus-n-codierte Vektoren als Repräsentationen einer Eingabe betrachtet.

Der Grundgedanke besteht darin, dass die Repräsentation einige wesentliche Features oder Eigenschaften der ursprünglichen Daten erfasst, um sie weiter analysieren zu können.

## 1.4 Übungen

1. Angenommen, wir trainieren ein Netz mit fünf konvolutionalen Schichten, gefolgt von drei vollständig verbundenen Schichten, ähnlich dem AlexNet (*https://en.wikipedia.org/wiki/AlexNet*), wie in Abbildung 1–4 veranschaulicht.

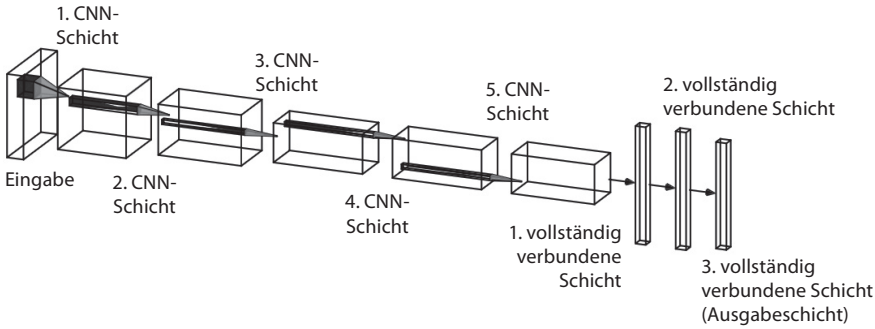


***Abb. 1–4*** *Eine Veranschaulichung von AlexNet*

   Diese vollständig verbundenen Schichten kann man sich als zwei verdeckte Schichten und eine Ausgabeschicht in einem mehrschichtigen Perzeptron vorstellen. Welche Schichten dieses neuronalen Netzes lassen sich nutzen, um nützliche Einbettungen zu erzeugen? Interessierte Leser finden weitere Details zur Architektur und Implementierung von AlexNet in der Originalveröffentlichung von Alex Krizhevsky, Ilya Sutskever und Geoffrey Hinton.

2. Nennen Sie einige Typen von Eingaberepräsentationen, die keine Einbettungen sind.

## 1.5 Referenzen

- Die Architektur und Implementierung von AlexNet wird ursprünglich in folgendem Paper beschrieben: Alex Krizhevsky, Ilya Sutskever und Geoffrey Hinton, »ImageNet Classification with Deep Convolutional Neural Networks« (2012), *https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks*.

# 2 Selbstüberwachtes Lernen

**Was ist selbstüberwachtes Lernen, wann ist es nützlich und wie sehen die Hauptansätze aus, um es zu implementieren?**

*Selbstüberwachtes Lernen* ist eine Vortrainingsprozedur, die es neuronalen Netzen ermöglicht, große, ungelabelte Datensätze in überwachter Art und Weise zu nutzen. Dieses Kapitel vergleicht selbstüberwachtes Lernen mit Transferlernen, einer verwandten Methode zum Vortrainieren von neuronalen Netzen, und erörtert die praktischen Anwendungen von selbstüberwachtem Lernen. Schließlich skizziert es die wichtigsten Kategorien von selbstüberwachtem Lernen.

## 2.1 Selbstüberwachtes Lernen vs. Transferlernen

Selbstüberwachtes Lernen ist mit dem Transferlernen verwandt, einer Technik, bei der ein Modell, das für eine bestimmte Aufgabe vortrainiert worden ist, als Ausgangspunkt für ein Modell für eine zweite Aufgabe wiederverwendet wird. Nehmen wir zum Beispiel an, dass wir einen Bildklassifikator trainieren wollen, um Vogelarten zu klassifizieren. Beim Transferlernen würden wir ein CNN (Convolutional Neural Network) auf dem ImageNet-Datensatz trainieren, einem großen, gelabelten Bilddatensatz mit vielen verschiedenen Kategorien, einschließlich verschiedener Objekte und Tiere. Nach dem Vortraining mit dem allgemeinen ImageNet-Datensatz wird das trainierte Modell mit dem kleineren, spezifischeren Zieldatensatz trainiert, der die interessierenden Vogelarten enthält. (Oftmals müssen wir nur die klassenspezifische Ausgabeschicht ändern, können ansonsten aber das vortrainierte Netz unverändert übernehmen.)

Abbildung 2–1 veranschaulicht den Ablauf beim Transferlernen.

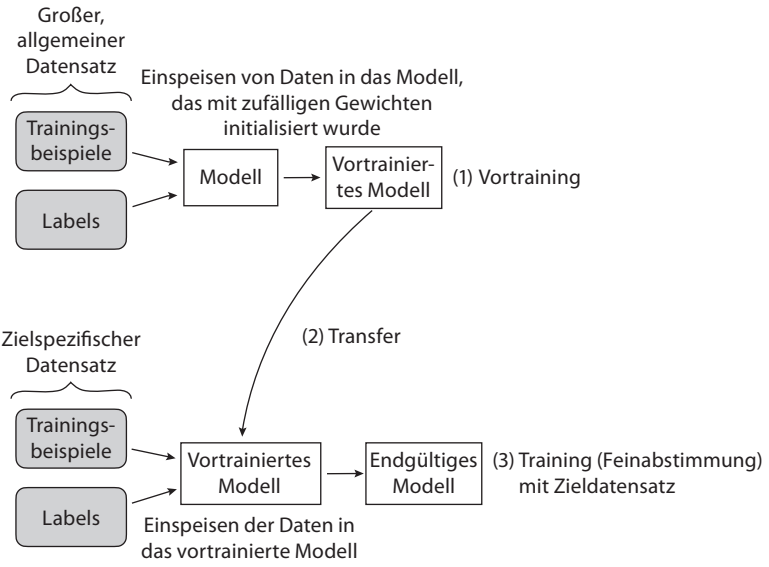


***Abb. 2–1*** *Vortraining mit konventionellem Transferlernen*

Selbstüberwachtes Lernen ist ein alternativer Ansatz zum Transferlernen, bei dem das Modell nicht mit gelabelten Daten, sondern mit ungelabelten Daten vortrainiert wird. Wir betrachten einen ungelabelten Datensatz, für den wir keine Label-Informationen haben, und suchen dann nach einer Möglichkeit, Labels aus der Struktur des Datensatzes zu erhalten, um eine Vorhersageaufgabe für das neuronale Netz zu formulieren, wie Abbildung 2–2 zeigt. Diese Aufgaben für das selbstüberwachte Training nennt man auch *Voraufgaben*.

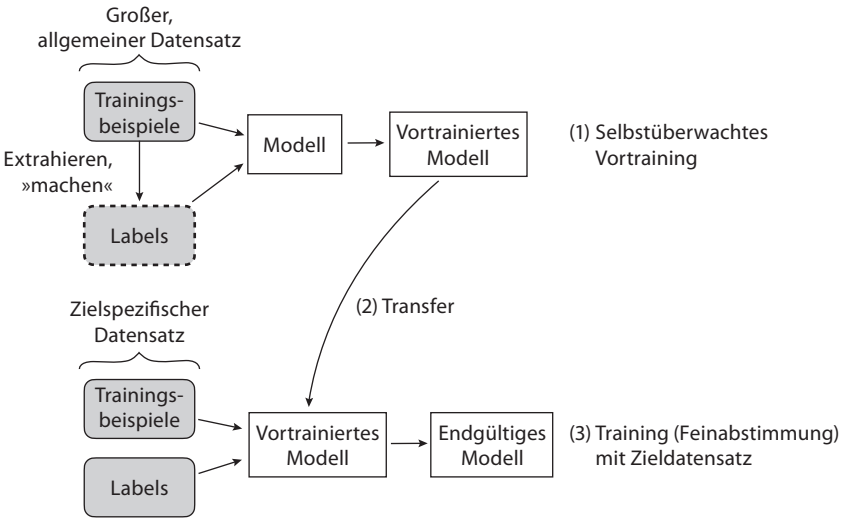


***Abb. 2–2*** *Vortraining mit selbstüberwachtem Lernen*

Der Hauptunterschied zwischen Transferlernen und selbstüberwachtem Lernen liegt in der Art und Weise, wie wir die Labels in Schritt 1 der Darstellungen von Abbildung 2–1 und Abbildung 2–2 erhalten. Beim Transferlernen gehen wir davon aus, dass die Labels zusammen mit dem Datensatz bereitgestellt werden. In der Regel werden die Labels manuell von Label-Experten den Daten zugeordnet. Beim selbstüberwachten Lernen lassen sich die Labels direkt aus den Trainingsbeispielen ableiten.

Eine Aufgabe des selbstüberwachten Lernens könnte eine Vorhersage fehlender Wörter im Kontext der Verarbeitung natürlicher Sprache sein. Zum Beispiel können wir bei dem Satz »Draußen ist es schön und sonnig« das Wort »sonnig« ausmaskieren, dem Netz die Eingabe »Draußen ist es schön und [MASKE]« einspeisen und das Netz das fehlende Wort an der Position »[MASKE]« vorhersagen lassen. In ähnlicher Weise könnten wir in einem Computer-Vision-Kontext Bildausschnitte entfernen und das neuronale Netz die Lücken füllen lassen. Dies sind lediglich zwei Beispiele für Aufgaben des selbstüberwachten Lernens; es gibt viele weitere Methoden und Paradigmen für diese Art des Lernens.

Zusammenfassend lässt sich sagen, dass man selbstüberwachtes Lernen bei der Voraufgabe als Repräsentationslernen betrachten kann. Wir können das vorab trainierte Modell verwenden, um es für die Zielaufgabe (auch bekannt als die *Downstream*-Aufgabe) zu optimieren.

## 2.2 Ungelabelte Daten nutzen

Große Architekturen neuronaler Netze benötigen große Mengen an gelabelten Daten, um eine gute Leistung und Generalisierung zu erzielen. Für viele Problembereiche haben wir jedoch keinen Zugang zu großen, gelabelten Datensätzen. Beim selbstüberwachten Lernen können wir ungelabelte Daten nutzen. Daher ist selbstüberwachtes Lernen beim Arbeiten mit großen neuronalen Netzen und mit einer begrenzten Menge von gelabelten Trainingsdaten höchstwahrscheinlich nützlich.

Transformer-basierte Architekturen, die die Grundlage der LLMs und Vision Transformer bilden, erfordern bekanntermaßen selbstüberwachtes Lernen für das Vortraining, um gute Leistungen zu erzielen.

Für kleine Modelle von neuronalen Netzen wie zum Beispiel mehrschichtige Perzeptrons mit zwei oder drei Schichten wird selbstüberwachtes Lernen in der Regel weder als nützlich noch als notwendig erachtet.

Selbstüberwachtes Lernen ist auch beim herkömmlichen maschinellen Lernen mit nichtparametrischen Modellen wie baumbasierten Random Forests oder Gradient Boosting nicht sinnvoll. Konventionelle baumbasierte Methoden besitzen keine feste Parameterstruktur (im Gegensatz zum Beispiel zu den Gewichtsmatrizen). Daher sind konventionelle baumbasierte Methoden nicht zum Transferlernen fähig und mit selbstüberwachtem Lernen nicht kompatibel.