# 12 Autoencoder

Systemtechnik BSc
FS 2025

## Aufgaben



### Lernziele

Nach dem Bearbeiten dieser Übungsserie ...

- können Sie den Aufbau und die Fuktionsweise eines *Autoencoders* (AE) ihren Freunden erklären. Sie können mindestens *drei Anwendungen* nennen, wofür die Autoencoder-Architektur gut geeignet ist (embedding, denoising, anomaly detection).
  Aufgaben: 1
- wissen Sie, was ein *Denoising Autoencoder* ist und wie man einen Autoencoder dazu bringt, Rauschen aus Bildern zu entfernen.
  Aufgaben: 1
- können Sie die Architektur (als json-File) und die trainierten Gewichte (als h5-File) Ihrer `PyTorch`-Modelle abspeichern und wieder laden.
  Aufgaben: 1

### Autoencoder - Selbstüberwachte Lerner für kompakte Datenrepäsentation

Autoencoder sind eine spezielle Art von neuronalen Feedforward-Netzen, bei denen die Eingabe die gleiche ist wie die Ausgabe. Sie komprimieren die Eingabe in einen niedrigdimensionalen Code (latent representation, latent space) und rekonstruieren dann die Ausgabe aus dieser Darstellung. Der Code ist eine kompakte Repräsentation oder Komprimierung der Eingabe, auch Latent-Space-Darstellung genannt. Ein Autoencoder

besteht aus 3 Komponenten: Encoder, Code und Decoder. Der Encoder komprimiert die Eingabe und erzeugt den Code, der Decoder rekonstruiert dann die Eingabe nur mit Hilfe dieses Codes.
Um einen Autoencoder zu erstellen, benötigen wir drei Dinge: eine *Encoder*, einen *Decoder* und eine *Verlustfunktion*, um die Ausgabe mit dem Ziel zu vergleichen. Autoencoder sind hauptsächlich ein Algorithmus zur *Dimensionalitätsreduktion* (oder Kompression) mit einigen wichtigen Eigenschaften:

i. **Datenspezifisch**: Autoencoder sind nur in der Lage, Daten sinnvoll zu komprimieren, die denen ähnlich sind, auf denen sie trainiert wurden. Da sie spezifische Merkmale für die gegebenen Trainingsdaten lernen, unterscheiden sie sich von einem Standard-Datenkomprimierungsalgorithmus wie gzip. Wir können also nicht erwarten, dass ein Autoencoder, der auf handgeschriebene Ziffern trainiert wurde, Landschaftsfotos komprimiert.

ii. **Verlustbehaftet**: Die Ausgabe des Autoencoders entspricht nicht genau der Eingabe, sondern ist eine nahe, aber verschlechterte Darstellung. Wenn Sie eine verlustfreie Komprimierung wünschen, sind sie nicht die richtige Wahl.

iii. **Unüberwacht**: Um einen Autoencoder zu trainieren, brauchen wir nichts Ausgefallenes zu tun, sondern können ihm einfach die rohen Eingabedaten vorlegen. Autoencoder werden als unbeaufsichtigte Lerntechnik betrachtet, da sie keine expliziten Lables zum Trainieren benötigen. Genauer gesagt sind sie jedoch self-supervised, da sie ihre eigenen Bezeichnungen aus den Trainingsdaten generieren.

Diese Übungsserie zeigt, wie sich mit Autoencodern effizient Rauschen aus Bildern entfernen lässt. Wir verwenden hier als Datenquelle wiederum den berühmten MNIST-Datensatz aus handgeschriebenen Ziffern, welchem wir künstlich Rauschen hinzufügen.

### Aufgabe 1. Denoising Autoencoder (DAE)

Denoising Autokodierern (DAE) nehmen eine teilweise verfälschte Eingabe und werden so trainiert, dass sie die ursprüngliche, unverzerrte Eingabe wiederherstellen. In der Praxis besteht das Ziel von Denoising Autokodierern darin, die verfälschte Eingabe zu bereinigen, also zu entrauschen. Diesem Ansatz liegen zwei Annahmen zu Grunde:

- Repräsentationen auf höherer Ebene sind relativ stabil und robust gegenüber der Verfälschung des Eingangsmaterials;

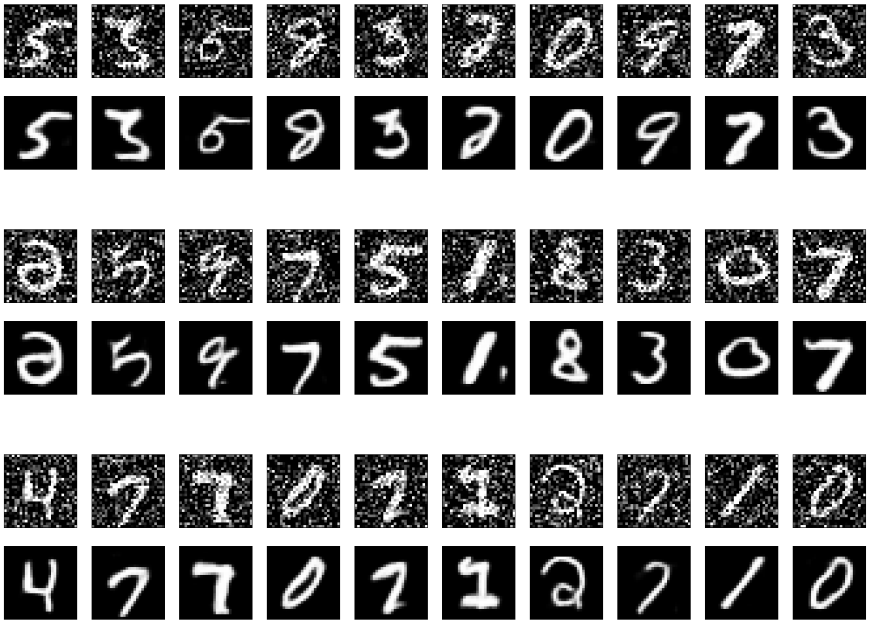

Abbildung 1:

- Um eine gute Rauschunterdrückung durchzuführen, muss das Modell Merkmale extrahieren, die nützliche Strukturen in der Eingangsverteilung erfassen.

Die Rauschunterdrückung dient als Trainingskriterium für das Erlernen der Extraktion nützlicher Merkmale, die bessere Repräsentationen der Eingabe auf höherer Ebene darstellen. Das Training läuft wie folgt ab:

1. Der usprüngliche Input $\mathbf{x}$ wird durch eine stochastische Transformation (hier Rauschen) in einen verrauschten Input $\tilde{\mathbf{x}} \sim q(\tilde{\mathbf{x}} \mid \mathbf{x})$ transformiert.

2. Dieser verrauschte Input $\tilde{\mathbf{x}}$ wird danach durch den Encoder $p_{\mathbf{w}}(\tilde{\mathbf{x}}) = \sigma(\mathbf{w}\tilde{\mathbf{x}} + \mathbf{b})$ in eine latenten Code $\mathbf{z} \sim p_{\mathbf{w}}(\tilde{\mathbf{x}})$ abgebildet.

3. Aus dieser latenten Darstellung $\mathbf{z}$ versucht der der Decoder den ursprünglichen Output zu rekonstruieren: $\mathbf{x}' = q_{\mathbf{w}'}(\mathbf{z}) \approx \mathbf{x}$

Die Parameter des Modells $\mathbf{w}$ und $\mathbf{w}'$ werden so trainiert, dass der durchschnittliche Rekonstruktionsfehler über die Trainingsdaten minimiert wird, d. h. die Differenz zwi-

schen $\mathbf{x}'$ und der ursprünglichen unverfälschten Eingabe $\mathbf{x}$. Das oben beschriebene Trainingsverfahren könnte mit jeder Art von Korruptionsverfahren angewendet werden. Einige Beispiele sind additives isotropes Gass'sches Rauschen, Maskierungsrauschen (ein Teil der für jedes Beispiel zufällig ausgewählten Eingabe wird auf 0 gesetzt) oder Salz-und-Pfeffer-Rauschen (ein Teil der für jedes Beispiel zufällig ausgewählten Eingabe wird mit gleichmässiger Wahrscheinlichkeit auf seinen minimalen oder maximalen Wert gesetzt).

NB: Falls Sie nur wenig Zeit zur Verfügung haben, laden Sie das Jupyter-Notebook `ANN12_Denoising_Auotencoder_TEMPLATE_pl.ipynb` herunter und versuchen Sie, Schritt für Schritt die Lösungen nachzuvollziehen. Alternativ können Sie die Lösungen auch auf Google-Colab ausführen.

### Aufgaben

(a) **Hilfsfunktionen**: Öffnen sie das Jupyter-Notebook `ANN12_Denoising_Auotencoder_TEMPLATE_pl.ipynb`, welches Sie auf Moodle finden und führen Sie die ersten Zellen aus, welche die Hilfsfunktionen für das Hinzufügen von Rauschen zu definieren (preprocessing) und die Daten zu visualisieren. Studieren Sie einzelnen Funktionen und vollziehen Sie nach, was diese Funktionen tun.

(b) **Datenvorbereitung**: Dieser Abschnitt stellt das MNIST-Dataset und die Dataloader zur Verfügung, die später für das Trainieren des Denoising Autoencoders (DAE) verwendet werden.

(c) **Autoencoder:** Generieren Sie eine PyTorch-Lightning-Modellarchitektur für den DAE mit folgendem *Encoder*:

```
self.encoder = nn.Sequential(
  nn.Conv2d(1, 32, 3, padding=1),
  nn.ReLU(),
  nn.MaxPool2d(2),
  nn.Conv2d(32, 32, 3, padding=1),
  nn.ReLU(),
  nn.MaxPool2d(2),    )

```

und folgendem *Decoder*:

```
self.decoder = nn.Sequential(
  nn.ConvTranspose2d(32, 32, 3, stride=2,
    padding=1, output_padding=1),
  nn.ReLU(),
  nn.ConvTranspose2d(32, 32, 3, stride=2,
    padding=1, output_padding=1),
  nn.ReLU(),
  nn.Conv2d(32, 1, 3, padding=1),
  nn.Sigmoid(),   )

```

Definieren Sie die `forward`-Methode, den `training_step` und den `validation_step`. Verwenden Sie zur Optimierung den `adam`-Optimizer mit einer Lernrate $\lambda = 10^{-3}$ und als Verlustfunktion die `binary_crossentropy` und als Metrik die `accuracy`.

(d) **Training** Trainieren Sie den DAE über 20 Epochen. Definieren einen `EarlyStopping`-Callback und verwenden Sie den `tensoboard`-Logger

```
logger = TensorBoardLogger("tb_logs", name="denoising_ae")

trainer = pl.Trainer(
  max_epochs=20,
  logger=logger,
  accelerator="auto",
  devices=1,
  callbacks=[EarlyStopping(monitor="train_loss", patience=5)],)

```

(e) **Speichern:** Speichern Sie die Modellarchitektur (Graph) als `json`-File und die trainierten Gewichte als `pt`-File ab.

```
torch.save(model.state_dict(), "AE_weights.pt")

```

(f) **Lernkurven**: Plotten Sie die Lernkurven (für `loss` und `accuracy`) in `tensorboard`.

(g) **Prediction, Denoising:** Verwenden Sie das trainierte Modell verwendet, um verrauschte Testbilder zu rekonstruieren und visualisieren Sie anschliessend original vs. rekonstruktuierter (entrauschter) Daten.

(h) **Trainieren eines Denoising-AE (DAE):** Dieser Schritt ist *optional* und dient dazu, die Qualität der Rekonstruktionen weiter zu verbessern, indem das Modell

länger trainiert wird. Trainieren Sie den Autoencoder für 100 Epochen und einer Sapelgrösse von 128, indem sie als Input-Daten nun die verrauschten Daten und aus Output-Daten die sauberen Daten verwenden.

(i) **Noise filtern mit dem DAE**: Machen Sie nun einen Test, um zu prüfen, wie gut der DAE das Rauschen aus den Bildern entfernt. Machen Sie hierfür eine Prädiktion auf den verrauschten Testdaten und vergleichen Sie diese mit ein sauberen Inputdaten.

## Lösungen

**Lösung 1.**
Sie finden die Lösung auf Moodle unter `ANN12_Denoising_Autoencoder_SOLUTION_pl.ipynb`.