# 11 Generative Modelle

Systemtechnik BSc
FS 2025

## Aufgaben

Applied Neural Networks im Modul ANN | WUCH

### Lernziele

Nach dem Bearbeiten dieser Übungsserie ...

- kennen und verstehen Sie den Unterschied zwischen *generativen* und *diskriminativen* Modellen. Sie können für jede Art von Klassifikatoren zwei Beispiele nennen.
  Aufgaben: 1
- sind Sie in der Lage, ein *atomares Sprachmodell* $p(x_{n+1} \mid x_n, x_{n-1}, \ldots x_{n-m})$ (auf Basis von Zeichenketten) auf einem grossen Textkorpus zu trainieren, welches das nächste Zeichen einer vorangehenden Zeichnekette der Länge $m$ voraussagt.
  Aufgaben: 1
- Können Sie mit Hilfe von *Sampling* aus diesem Sprachmodell ein generatives Modell erzeugen, welches Texte im Stil von William Shakespeare (oder anderen Autoren) generiert. Sie kennen die mit einem *Temperaturparameter* $\tau$ erweiterte Softmax-Funktion und können den Effekt dieses Parameters $\tau$ erklären
  Aufgaben: 1

.

### Generative und diskriminative Modelle

- Ein **generatives Modell** ist ein statistisches Modell der kombinierten Wahrscheinlichkeit (*joint probability distribution*) $p(X, y)$ bei gegebener beobachtbarer Variable $X$ und Zielvariable $y$ (target). Im Falle des unüberwachten Lernens, wo kein Label $y$ vorhanden ist, modelliert ein generatives Modell $p(X)$ die Verteilung

der ganzen Datenmenge $X$ und ist somit in der Lage, neue Samples $X' \sim p(X)$ zu erzeugen, z.B. neue Musikstücke im Stil von J.S. Bach oder Bilder von Gesichtern, die es so nicht gibt, wie z.B. this-person-does-not-exist.

- Ein **diskriminatives** Modell ist ein Modell der bedingten Wahrscheinlichkeit $p(y \mid X = x)$ des Labels $y$ bei einer Beobachtung $x$.

Man unterscheidet also zwei Arten von Klassifikatoren:

(a.) **generative Klassifikatoren** modellieren die kombinierten Verteilung $p(X, y)$ und treffen die Entscheidung auf der Basis von Bayes' Theorem (der inverse Wahrscheinlichkeit).

$$p(y \mid X) = \frac{p(X, y)}{p(X)}$$

Beispiele für generative Klassifikatoren sind der Naive Bayes-Klassifikator (NB) und die lineare Diskriminanzanalyse (LDA).

(b.) **diskriminative Klassifikatoren** Klassifikatoren, kommen ohne die Modellierung oder Berechnung der kombinierten Wahrscheinlichkeitsverteilung $p(X, y)$. Beispiele für diskriminative Modelle sind die Logistische Regression, Support Vector Machines (SVM) und Decision Trees oder Random Forests.

Mit dem Aufkommen des Deep Learning entstand durch die Kombination von generativen Modellen und tiefen neuronalen Netzen eine neue Familie von Methoden, die **Deep Generative Models (DGMs)** genannt werden. Zu den beliebten DGMs gehören *Variational Autoencoder* (VAEs) und *generative adversarische Netzwerke* (GANs).

## Dramen schreiben - fast im Stile von William Shakespeare

Diese Übungsserie zeigt, wie wir basierend auf einem Sprachmodell (language model)

$$p(x_{n+1} \mid x_n, x_{n-1}, \ldots x_{n-m})$$

welches auf einem ausreichend grossen Textkorpus erlernt wurde, ein generatives Modell zur Generierung von neuen Texten erzeugen kann. Wir werden Theaterstücke schreiben (fast) wie Wiliam Shakespeare (1564-1616). Unser Sprachmodell baut auf den

kleinsten Einheiten der Sprache auf, den einzelnen Buchstaben. Basierend auf den $m$ vorangehenden Buchstaben, sagt unter generatives Modell den nachfolgenden Buchstaben $x_{n+1}$ (oder das nachfolgende Zeichen) voraus. Durch wiederholtes, sequenzielles Sampeln entstehen so ganze Sätze oder Textabschnitte, die oft zwar keinen Sinn ergeben, aber doch nach Shakespeare tönen. Für dieses sequenzielle Modell bietet sich eine Architektur eines LSTM-Netzes an, an welches wir noch einen Dense-Layer mit einer Softmax-Aktivierung koppeln.
Anstatt nur immer das wahrscheinlichste Zeichen des Softwmax-Outputs zu wählen (`argmax`), können wir den Softmax-Output mit einer (verallgemeinerten) Temperaturparameter $\tau$ skalieren.

$$p_\tau(y = j \mid X) = \frac{\exp(\mathbf{W}^{(j)}\mathbf{X}/\tau)}{\sum_{k=1}^{K} \exp(\mathbf{W}^{(k)}\mathbf{X}/\tau)}$$

Bei hohen Temperaturen $\tau \to \infty$ haben alle Klassen (Buchstaben) nahezu die gleiche Wahrscheinlichkeit. Bei einer niedrigen Temperatur $\tau \to 0^+$ tendiert die Wahrscheinlichkeit Klasse mit der höchsten Wahrscheinlichkeit gegen 1. Sampelt man aus dieser Multinomialen Verteilung den nächstfolgenden Buchstaben, so nimmt mit steigendem Temperaturparameter die Wahrscheinlichkeit zu, dass mal ein anderer, aber vielleicht auch passender Buchstabe gezogen wird. Dies erhöht die Variation und sorgt dafür, dass immer wieder neue Texte generiert werden, auch wenn mit der gleichen Anfangssequenz gestartet wird.

NB: Falls Sie nur wenig Zeit zur Verfügung haben, laden Sie das Jupyter-Notebook `ANN11_LSTM_Shakespeare_SOLUTION.ipynb` herunter und versuchen Sie, Schritt für Schritt die Lösungen nachzuvollziehen. Alternativ können Sie die Lösungen auch auf Google-Colab ausführen.

### Aufgabe 1. Shakespeare Dramen generieren

Sie finden das Jupyter Notebook `ANN11_LSTM_Shakespeare_TEMPLATE_pl.ipynb` und die Daten auf Moodle. Gehen Sie von Anfang bis zur ersten Aufgabe (a) durch und lesen Sie die Einführung.

(a) **Textdatei laden**: Laden Sie das Textfile `shakespeare.txt` entweder direkt vom ANN github Verzeichnis oder lokal von Ihrem Computer. Lesen Sie den Datensatz

ein, der in `shakespeare.txt` gespeichert ist und speichern Sie den Text in einer Variablen `text`.

(b) **Zeichenbasierte Kodierung des Texts:** Generieren Sie ein `dictionary`, das jedem Character einen Index zuweist und umgekehrt.

(c) Stellen Sie eine beliebige Texststelle aus dem Text dar, z.B. die ersten 500 Zeichen.

(d) **Dataset Klasse**: Generieren Sie mit Hilfe eines LLM (ChatGPT, Gemini, ...) ein PyToch-Dataset, das folgende Aufgaben erfüllt:

- Generierung eines Datensatzes für die Sprachmodellierung auf Zeichenebene bei Shakespeare-Texten.
- Erstellen eines Vokabulars der Zeichen und Codierung der Zeichenketten in Indizes.
- Rückgabe von Sequenzen von Zeichenindizes und Index des nächsten Zeichens (label).
- Aufteilen des Input-Korpus in Zeichenketten der maximalen Länge `maxlen=40`. Verwenden Sie einen Stride von 3 Zeichen, sodass eine semi-redundante Repräsentation der Input-Sequenz entsteht.
- Erstellen einer eine Liste `next_chars`, welche das nächste, folgende Zeichen nach dieser Zeichenkette speichert. Dies sind die zugehörigen Labels.

Verwenden Sie folgende Struktur:

```
class CharDataset(Dataset):
"""
Dataset for character-level language modeling on Shakespeare texts.
Returns sequences of character indices and next-character labels.
"""
  def __init__(self, data_dir: str, maxlen: int = 40, step: int = 3)
:
    #enter your code here:
  def __len__(self):
    return len(self.sequences)

  def __getitem__(self, idx):
    seq = self.sequences[idx]
```

```
        # Return sequence of indices
        x = torch.tensor([self.char2idx[ch] for ch in seq],
         dtype=torch.long)
        # Target index
        y = torch.tensor(self.char2idx[self.next_chars[idx]],
         dtype=torch.long)
        return x, y

```

(e) **Modellarchitektur**: Implementieren Sie (mit Hilfe von ChatGPT o.ä.) ein mehrschichtiges LSTM-Modell, das Zeichenfolgen generieren kann, indem es lernt, die Wahrscheinlichkeit für das jeweils nächste Zeichen vorherzusagen. Zu Beginn werden die Eingabeindizes über eine Embedding-Schicht in dichte Vektoren überführt, was gegenüber grossen, spärlichen One-Hot-Repräsentationen deutlich speichereffizienter ist und es dem Modell erlaubt, semantische Ähnlichkeiten zwischen Zeichen zu lernen.

In `PyTorch` (und `PyTorch Lightning`) müssen Sie Ihre Ziele fast nie manuell mit einer One-Hot-Codierung versehen, und Sie vermeiden häufig eine One-Hot-Codierung Ihrer Eingaben durch Verwendung einer Einbettungsschicht. Verlustfunktionen in `PyTorch` akzeptieren ganzzahlige Klassenlabels. Modelleingaben können Indizes + Embedding sein

Nach dem embedding-Layer verarbeitet ein Long Short-Term Memory (LSTM) diese eingebetteten Sequenzen, um zeitliche Abhängigkeiten im Text zu erfassen und langfristige Kontextinformationen im verborgenen Zustand zu speichern.

Der letzte verborgene Zustand des LSTM dient als komprimierte Repräsentation der gesamten Eingabesequenz. Eine Dropout-Schicht verhindert Überanpassung. Ein dichtes Zwischennetzwerk mit nichtlinearer Aktivierung (ReLU) sorgt für zusätzliche Modellkapazität und stellt sicher, dass komplexe Zusammenhänge zwischen den Merkmalen gelernt werden können. Ein erneutes Dropout verbessert die Generalisierungsfähigkeit weiter. Abschließend projiziert eine finale vollverknüpfte Schicht dieses Merkmal auf so viele Ausgabeneinheiten, wie Zeichen im Vokabular vorhanden sind, um für jedes mögliche Folgezeichen einen Rohwert (Logit) zu erzeugen.

Verwenden Sie folgende Parameter:

```
    class LitCharRNN(pl.LightningModule):
      def __init__(self,
      vocab_size: int,
      embed_dim: int = 64,
      hidden_size: int = 128,
      dropout: float = 0.2,
      lr: float = 5e-3):

```

(f) Generieren Sie eine Funktion `tempered_softmax(preds, T)`, welche aus beliebigen Zufallwerten eine temperierte Softmax-Verteilung generiert. Plotten Sie diese Verteilung für unterschiedliche Temperaturwerte.

(g) **LSTM-Trainingsloop:** Trainieren Sie das Modell mit Early Stopping und einer Patience=5.

(h) Generieren Sie Texte mit verschiedenen Temperaturen. Was beobachten Sie? Ab welcher Temperatur sind Wörter nicht mehr erkennbar?

Verwenden Sie ein anderes Input-Textfile z.B. aus der Gutenberg-Datenbank.

## Lösungen

**Lösung 1.**
Sie finden die Lösung auf Moodle unter `ANN11_LSTM_Shakespeare_SOLUTION.ipynb`.