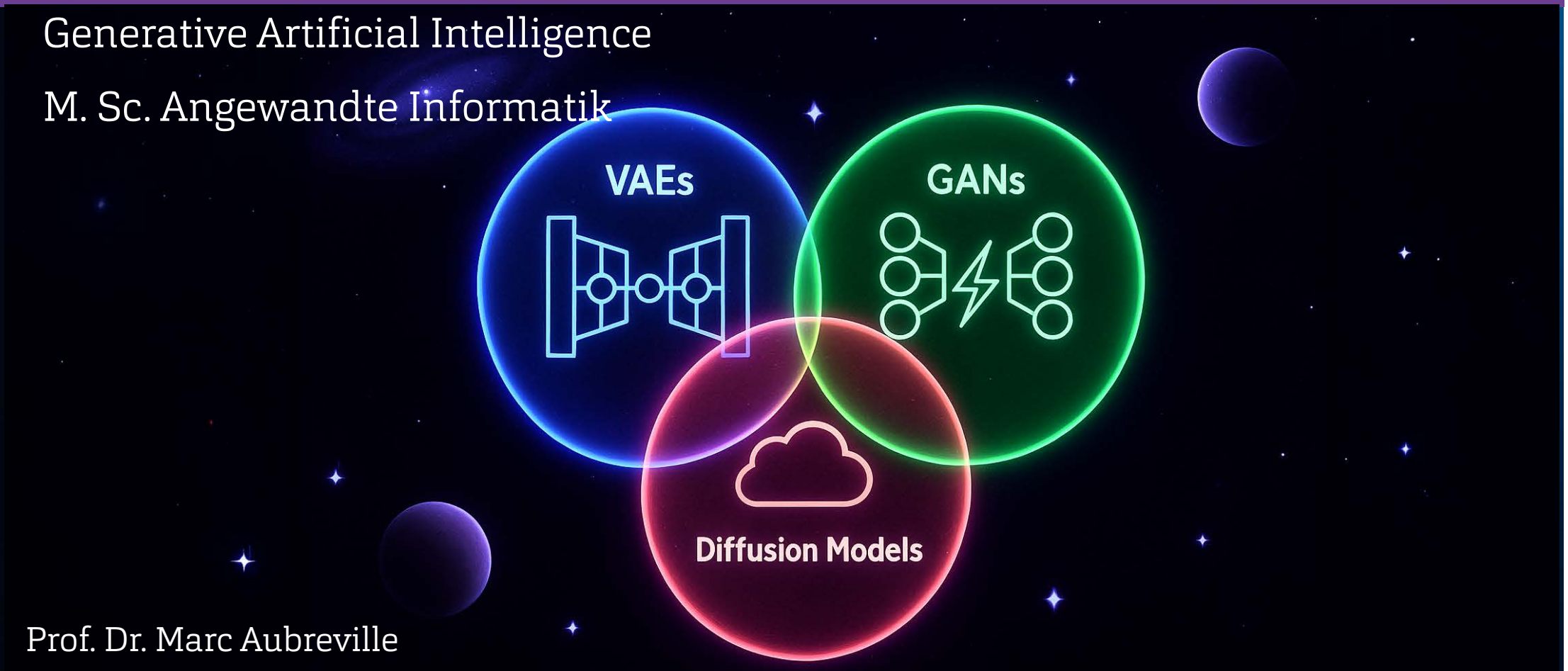
Generative Model Families

Generative Artificial Intelligence

M. Sc. Angewandte Informatik



Prof. Dr. Marc Aubreville

Hochschule
Flensburg
University of
Applied Sciences

Overview

Hochschule Flensburg
University of Applied Sciences

- In the following slide deck, we will be having a look at the main model families used in generative machine learning.
- In particular, these are:
  - Autoregressive models (RNNs, Transformers → GPT)
  - GANs (core idea, conditional GANs)
  - VAEs (concepts, posterior collapse)
  - Diffusion models (denoising, latent diffusion, scaling to large models)
- But first, we recap typical neural network model families.

Before we start

Hochschule Flensburg
University of Applied Sciences

- We will dig into the topics given examples from
  - Image generation
  - Text generation
  - etc.
- We will, however, later see that these ideas have been largely adapted also for the respective other data domain - and more.

Hochschule
Flensburg
University of
Applied Sciences

Model families

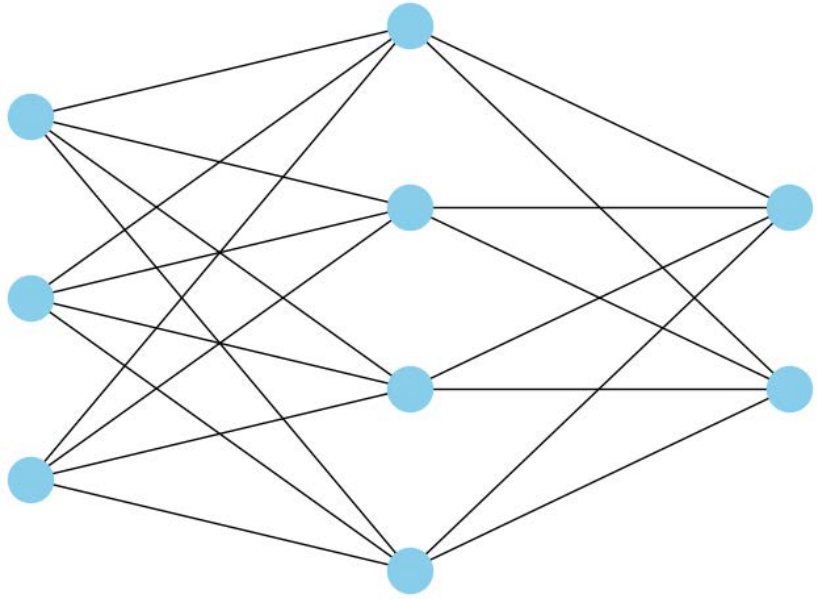
A brief overview

- Before we dig into model families that are utilized for generative AI, let's briefly recap some neural network model families:
  - Feed Forward Networks
  - Recurrent Networks
  - Convolutional Networks
  - Attention-based Networks (Transformers)

Hochschule Flensburg
University of Applied Sciences

Feed-Forward Networks

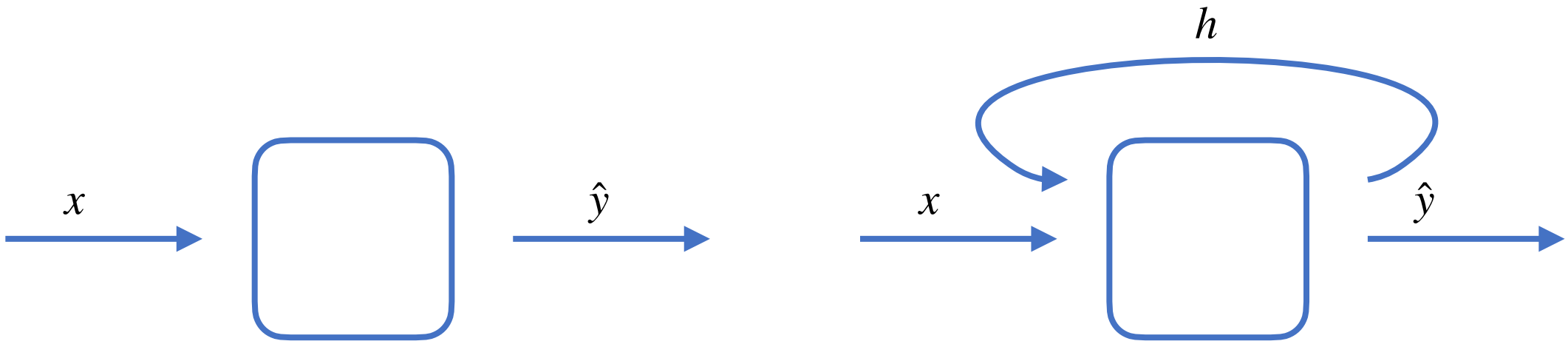
- Simplest type of neural network
- Data flows in one direction (input → hidden layers → output)
- No cycles or loops
- Good for basic classification & regression tasks



Hochschule Flensburg
University of Applied Sciences

Recurrent Neural Networks (RNNs)

- One way of dealing with sequences of varying lengths is to use recurrent structures.
- Recurrent networks contain at least one feedback loop, where the output is fed back to the input.
- This makes it possible to obtain and pass on information.
- This is traditionally represented by a hidden state h.



feed forward

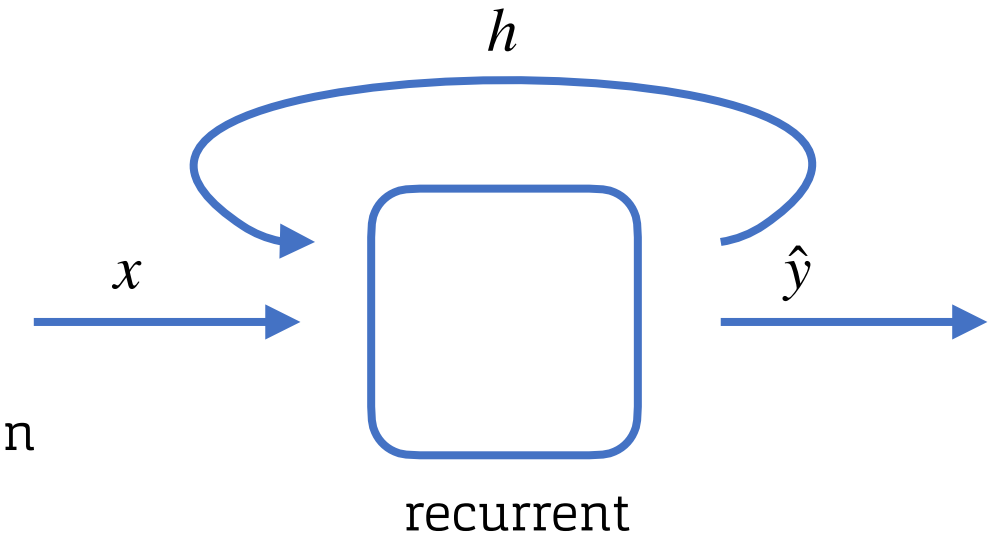
recurrent

RNN Layer

- A simple RNN layer is constructed in such a way that it calculates its output directly from the current hidden state:

$$h_t = \sigma(\mathbf{W}_h x_t + \mathbf{U}_h h_{t-1} + b_h)$$
$$y_t = \sigma(W_y h_t + b_y)$$

($\mathbf{W}, \mathbf{U}$ are weights, b are biases)

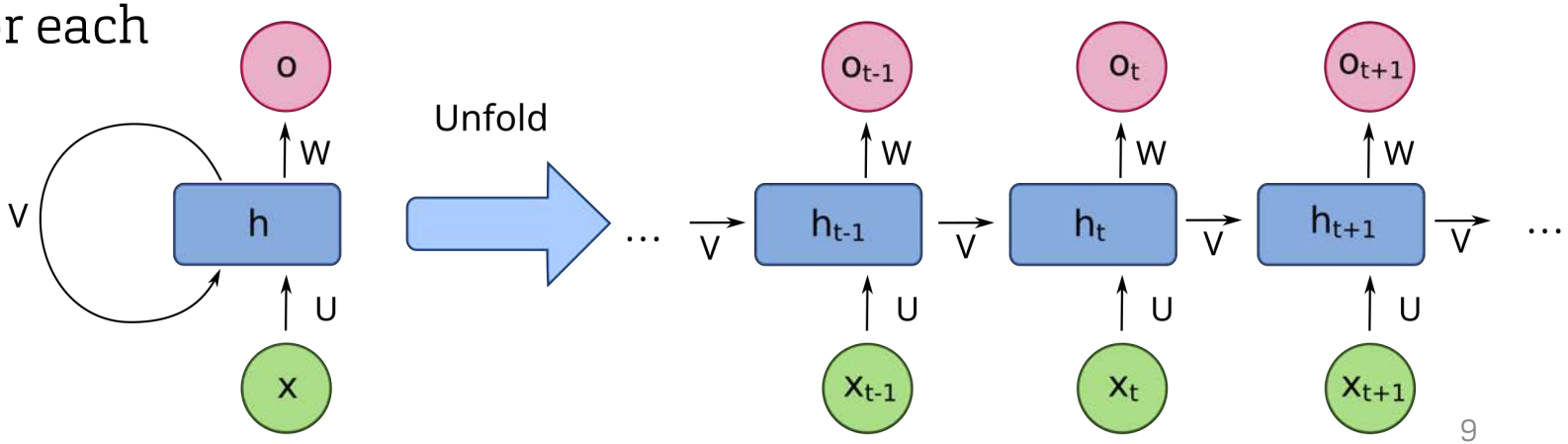


- The hidden state is calculated from the previous hidden states and the input.

Training of RNNs

Hochschule Flensburg
University of Applied Sciences

- Recurrent models cannot be trained using backpropagation without further ado, as they contain circular dependencies.
- The gradient thus becomes dependent on the previous state of the model or the hidden state.
- Backpropagation Through Time (BPTT):
  - The RNN is converted into an “unfolded” form for the entire sequence, similar to a feedforward network in which each time step model is a copy.
  - Gradients of loss are calculated for each time step, and the chain rule is applied over time.



Problems using Backpropagation Through Time

- Vanishing gradients:
  - If the derivatives are less than 1, the gradient becomes smaller and smaller over many time steps and is effectively lost.
  - Consequence: The model does not learn long-term dependencies.
- Exploding Gradients:
  - If the gradients are greater than 1, the gradient explodes exponentially and leads to numerical instabilities.
- High computational cost:
  - BPTT requires the entire sequence to be stored and processed, which is inefficient for long sequences.

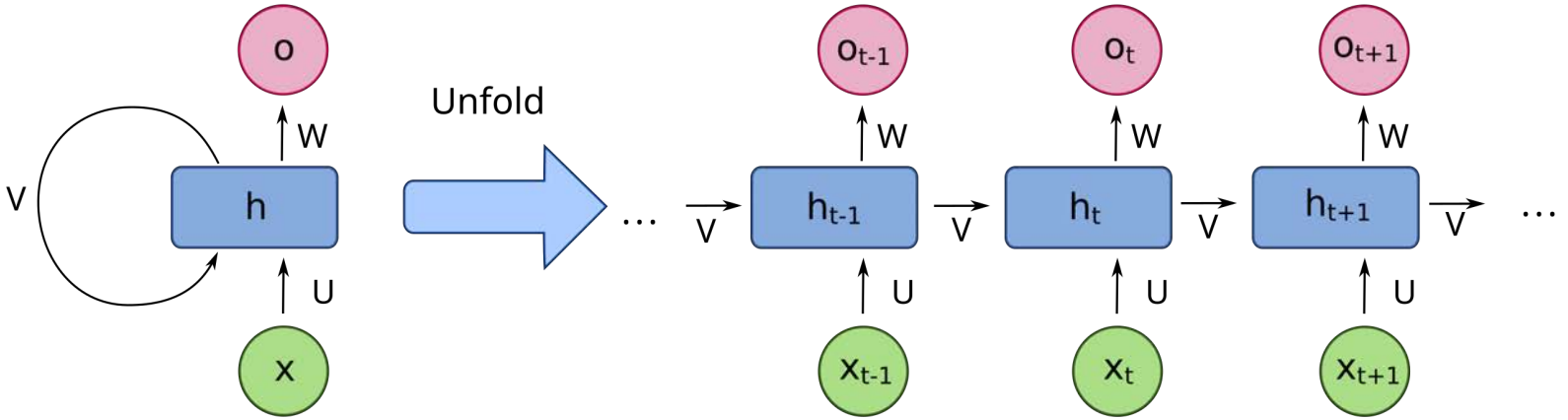
Hochschule Flensburg
University of Applied Sciences

Truncated Backpropagation Through Time

- Instead of propagating backward over the entire sequence, propagation is limited to a fixed number of time steps (e.g., k).
- Advantages:
  - Reduced memory and computing requirements.
  - Mitigation of problems with very long sequences.
- Disadvantage:
  - Loss of information over very long periods of time.

Hochschule Flensburg
University of Applied Sciences

Weaknesses of simple RNNs



- RNNs can only access information from further back in time very indirectly:
- The model processes sequences step by step, which means that information from earlier time steps can be overwritten or displaced by later ones.
- The hidden state stores all information in a single vector representation.
- Distributed and conflicting information (e.g., multiple topics) cannot be represented effectively.

Long Short Term-Memory (Hochreiter, Schmidhuber, 1997)

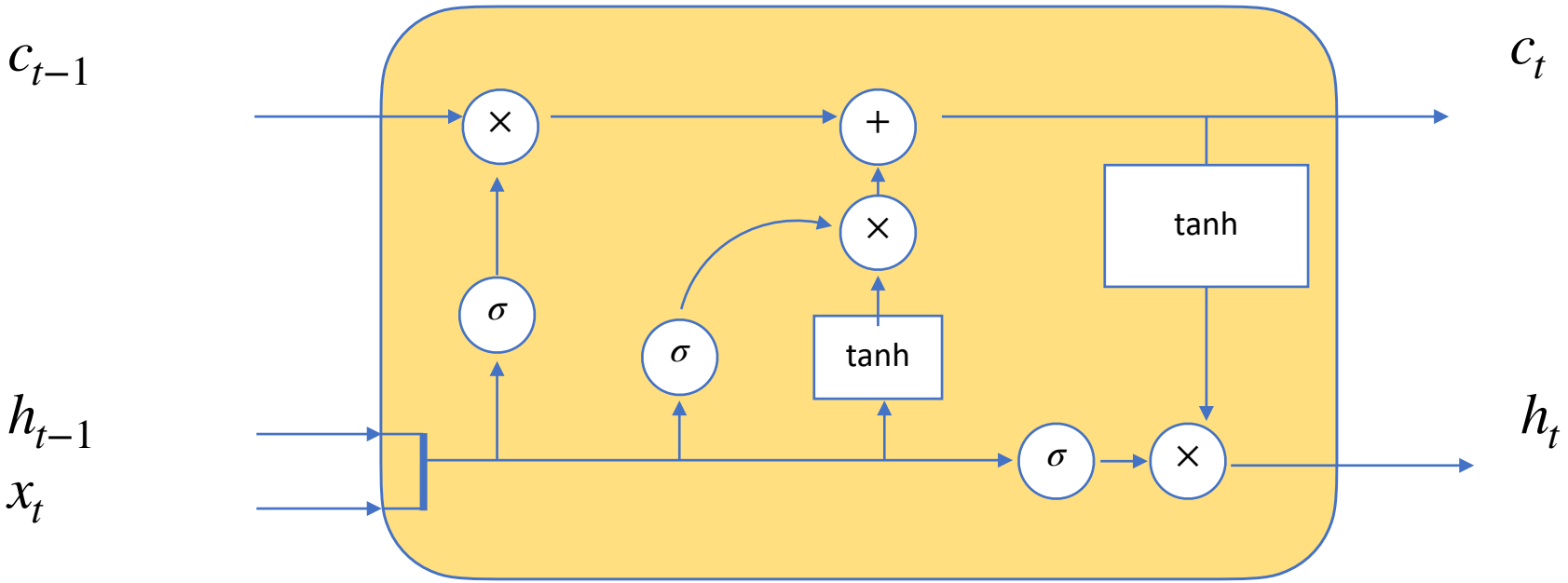
Hochschule Flensburg
University of Applied Sciences

- To overcome these weaknesses, a concept was devised in which the model specifically models which information should be passed on and which should be forgotten.
- The model should therefore have a longer short-term memory (long short-term memory).
- The solution is somewhat reminiscent of residual networks: we have to make it very easy for the network not to forget information. This can be compared to the skip connection.

- In LSTM, a cell state c was introduced for this purpose, which is passed on from model to model.
- In LSTM, information is either specifically added to the cell state (input gate) or specifically removed (forgetting gate).

The LSTM

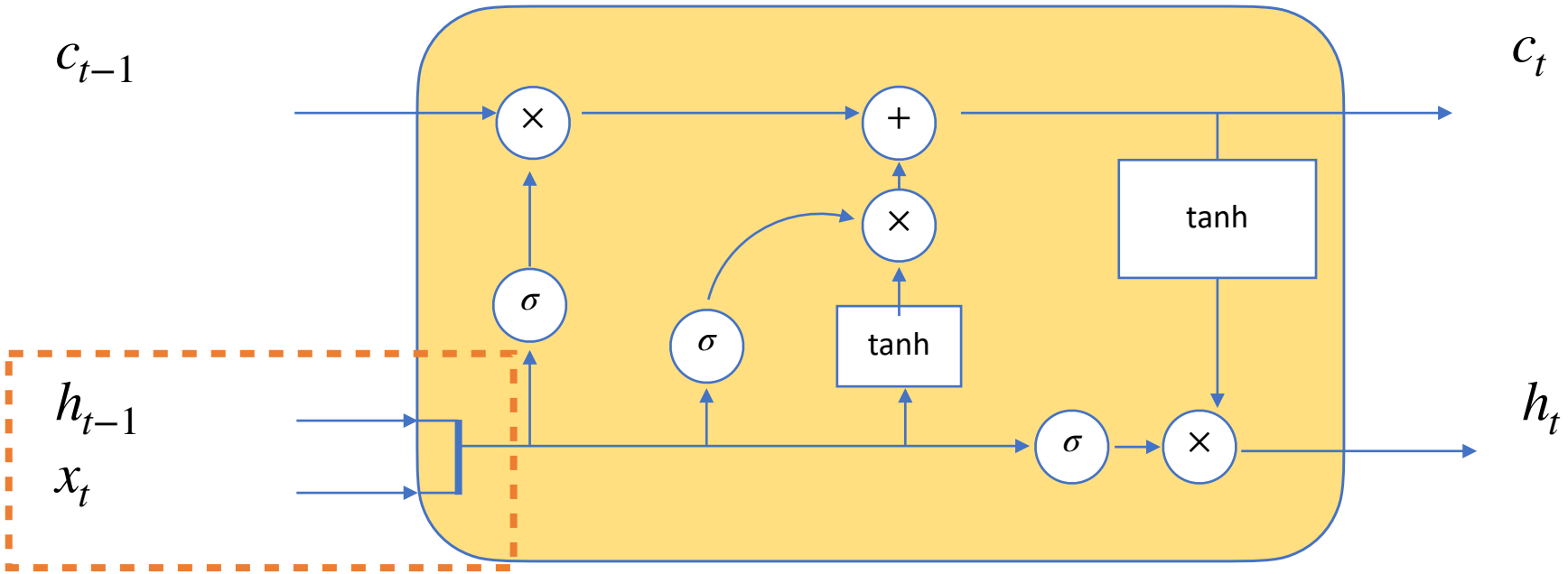
- At first glance, the LSTM seems a little confusing.



- The functions σ and tanh are the activation functions—they are each preceded by a linear layer (not shown).

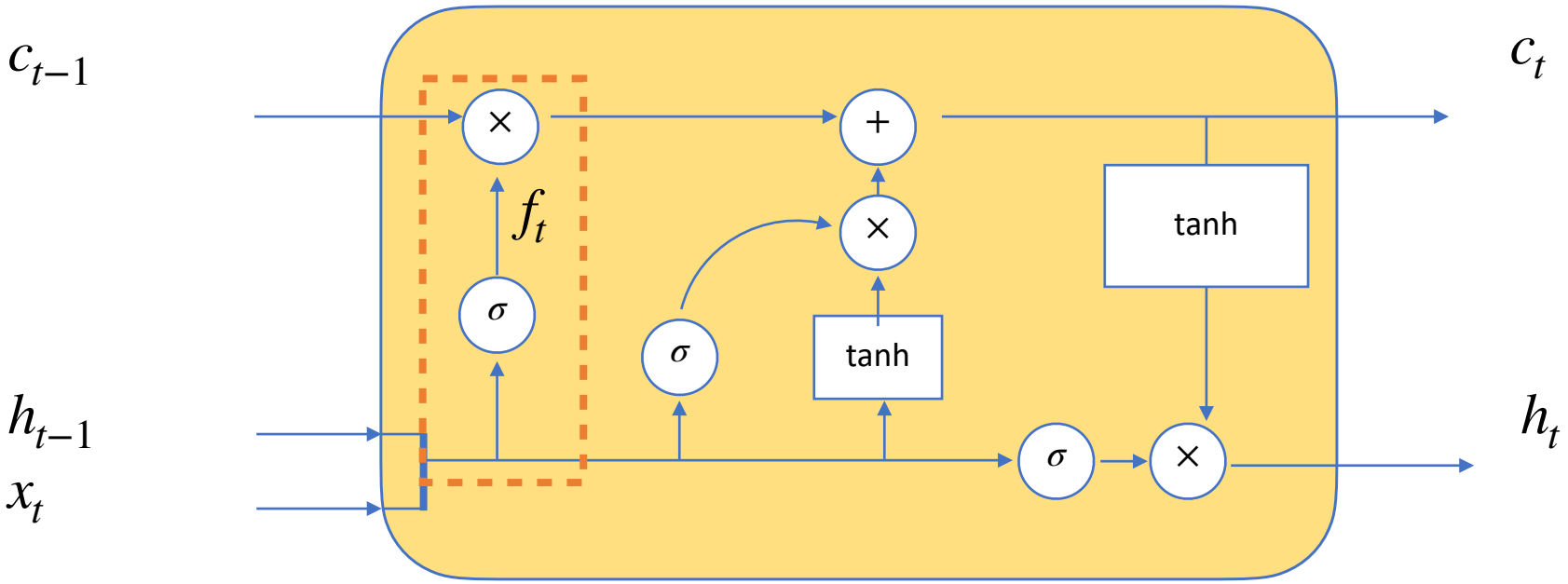
The LSTM: Inputs

- The current input samples x_t are being concatenated to the hidden state h_{t-1}.



The LSTM: Forgetting Gate f

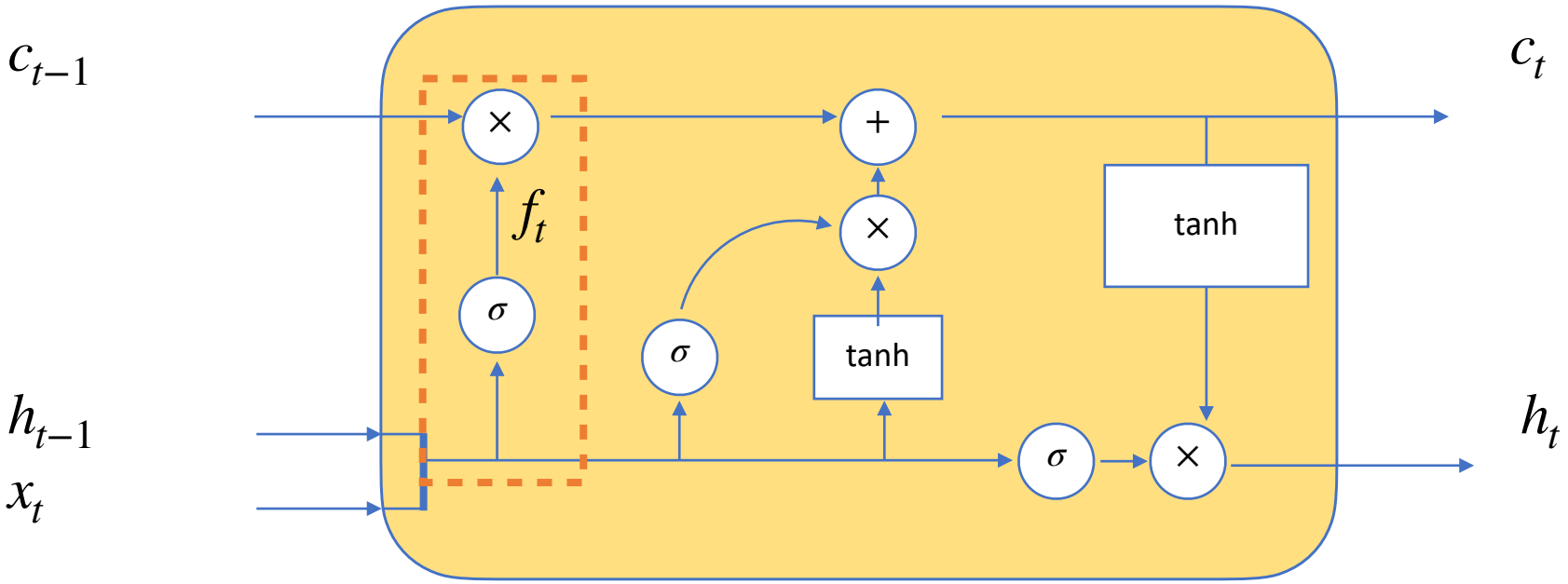
- A linear layer with sigmoid activation decides which part of the cell state information c_{t-1} should be forgotten:



- $f_t = \sigma\left(W_f \cdot \left[x_t, h_{t-1}\right] + b_f\right)$

The LSTM: Forgetting Gate f

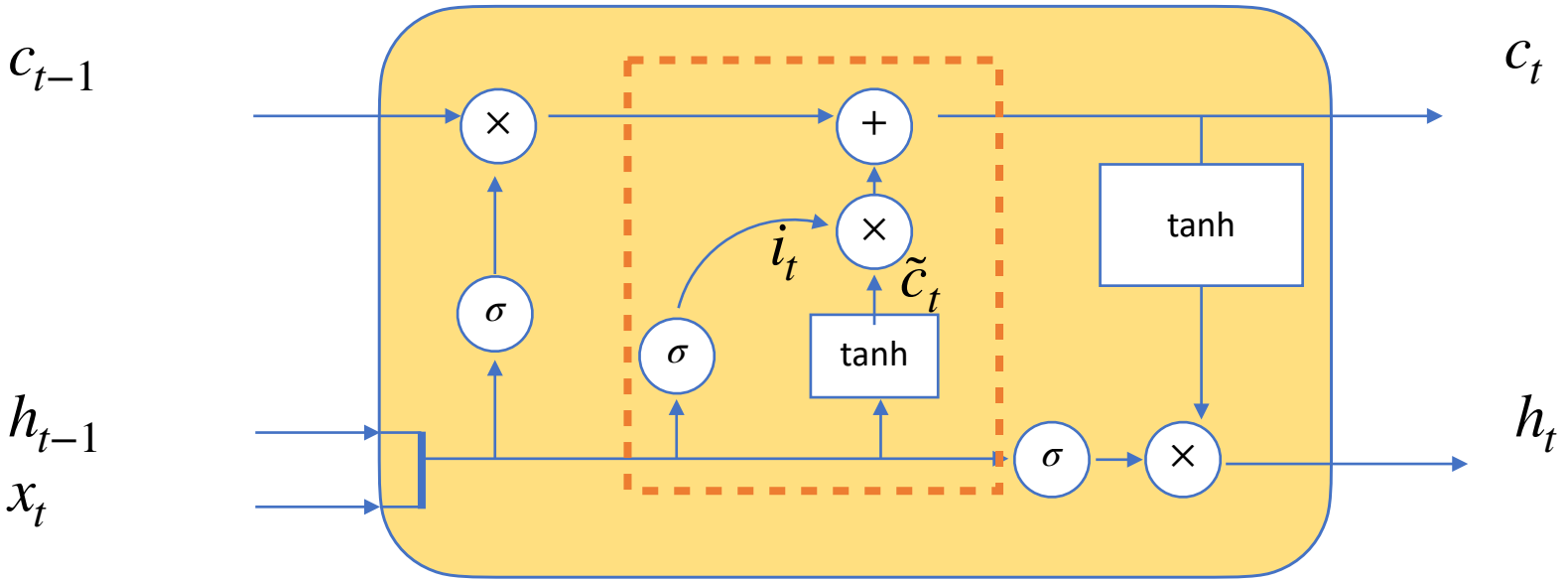
- The sigmoid activation at the forgetting gate can take the values 0 and 1 and can be thought of as a mask for the previous cell state information.



- This "mask" f_t is then multiplied by the cell state.

The LSTM: Input Gate i

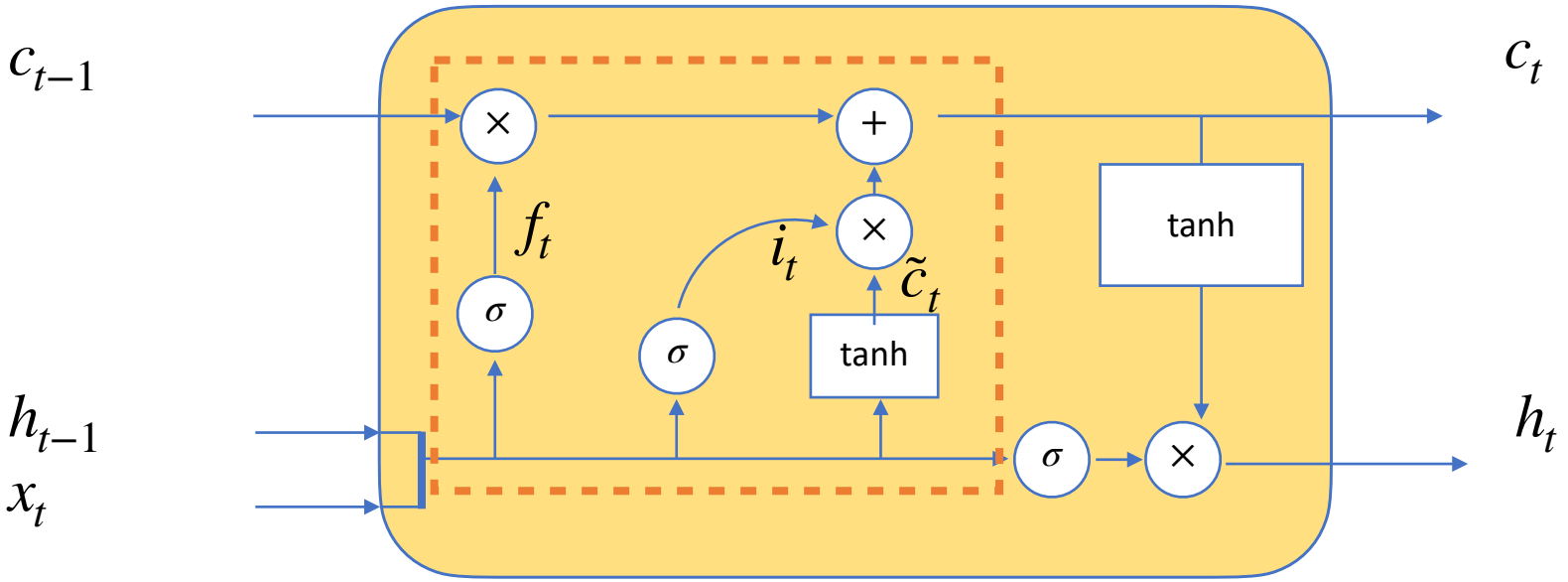
- The input gate selects information from the current hidden state + input that is relevant to the cell state and adds it to it:



- $i_t = \sigma\left(W_i\left[h_{t-1}, x\right] + b_i\right)$
- $\tilde{c} = \tanh\left(W_c\left[h_{t-1}, x\right] + b_c\right)$

The LSTM: Updating of the cell state

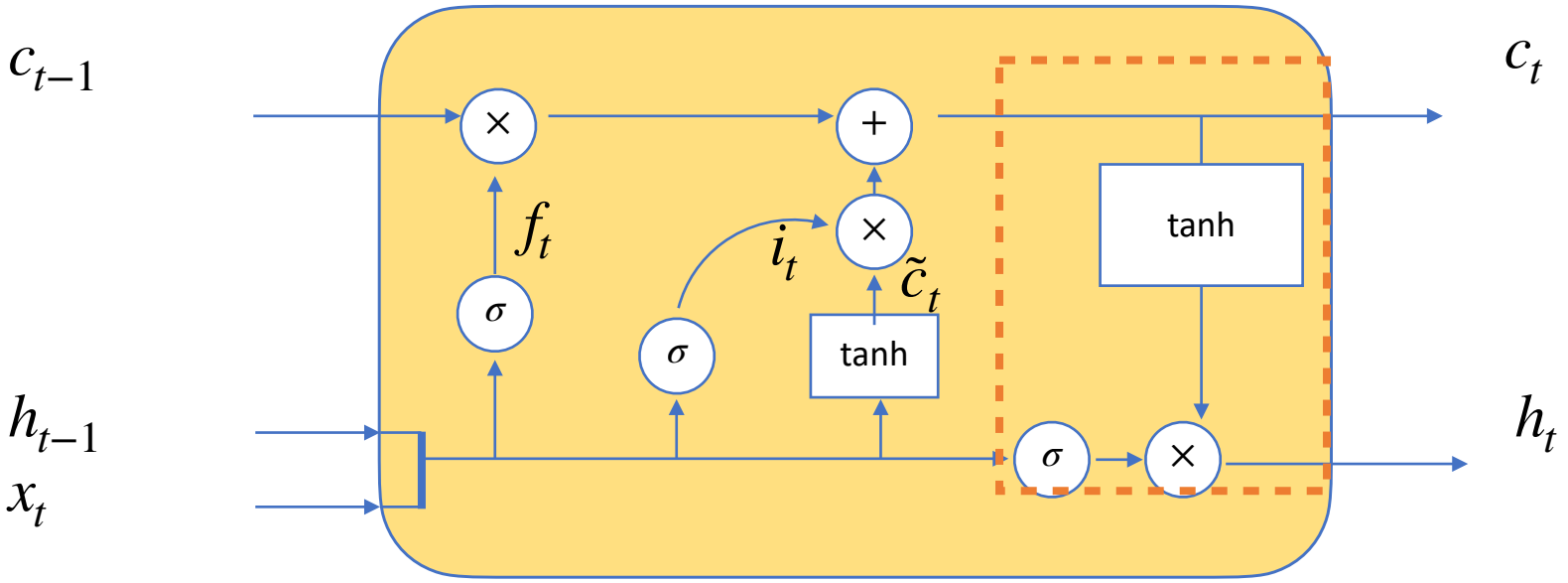
- The cell state is updated at each step via these two parts.



- $c_t = c_{t-1} \cdot f_t + \tilde{c} \cdot i_t$

The LSTM: Updating of the cell state

- The new hidden state (and output state, if applicable) is then calculated from the cell state:



- $h_t = \tanh(c_t) \cdot \sigma\left(W_o\left[h_{t-1}, x\right] + b_o\right)$
- Here, the decision as to which part of the cell state is relevant is again made based on the input and hidden state.

LSTM: Advantages

- Long-term dependencies:
  - Cell State allows relevant information to be retained over many time steps.
- Control via gates:
  - Gating mechanisms regulate which information is deleted, added, or output, which increases learning ability.
- Efficient training:
  - Avoids vanishing gradient problems, as the gradient propagates stably over time.

Hochschule Flensburg
University of Applied Sciences

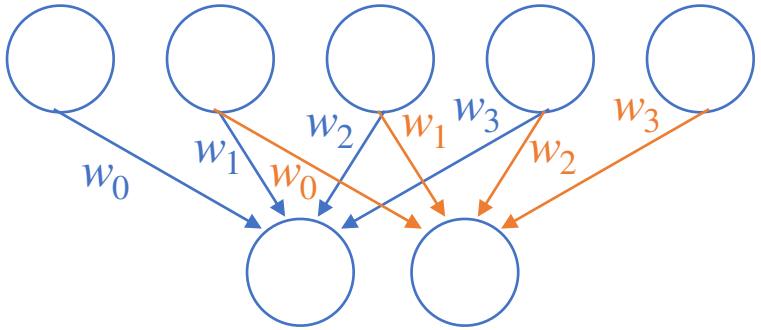
Drawbacks of LSTMs

- Architectural complexity:
  - More parameters:
  - LSTMs have four weight matrices (for forget, input, output gates, and cell state), which results in higher memory and computational requirements.
  - This makes LSTMs slower than simpler architectures such as GRUs or classic RNNs.
- More complex training:
  - The larger number of parameters increases the risk of overfitting, especially with small datasets.
- Sequential processing:
  - No parallelization:
  - LSTMs process sequences step by step, which makes computation time-consuming for long sequences.

Hochschule Flensburg
University of Applied Sciences

Convolutional Neural Networks

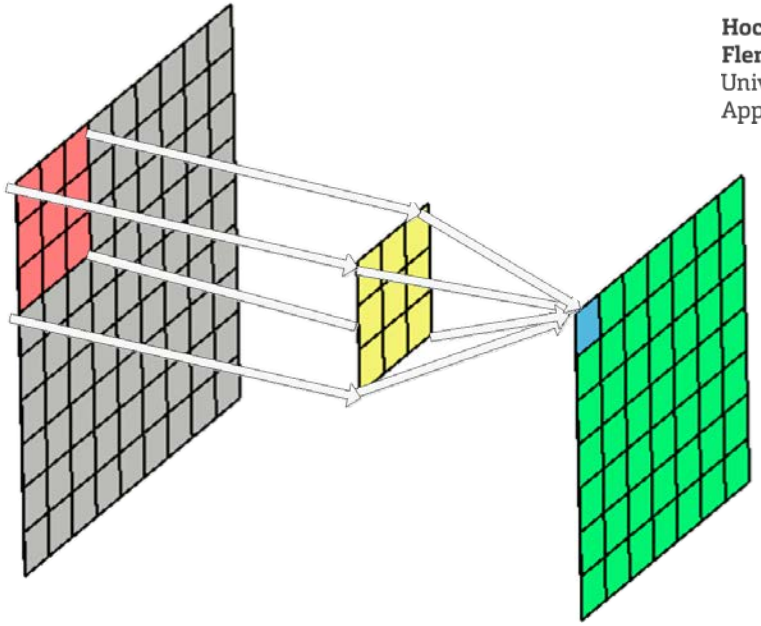
- Networks that utilize convolutions are called Convolutional Neural Networks (CNNs)
- Designed for grid-like data (e.g., images)
- Learn local patterns with shared filters
- Pooling reduces spatial size & builds invariance
- Stacks of convolutional layers → hierarchical features
- Strong for vision; also used in 1D signals (audio, text)



Hochschule Flensburg
University of Applied Sciences

CNNs: Fundamentals

- Fundamentally, CNNs learn kernels that can be convolved with the input signal in search of a pattern.
- The key idea is to exploit the spatial structure of the data through local connectivity and weight sharing.
- Receptive Field:
  Each neuron only "sees" a small region of the input (e.g., a 3×3 patch), capturing local patterns such as edges or textures.
- Shared weights:
  The same set of learned weights (filter) is applied across all spatial locations and hence detects the same feature wherever it appears in the image.

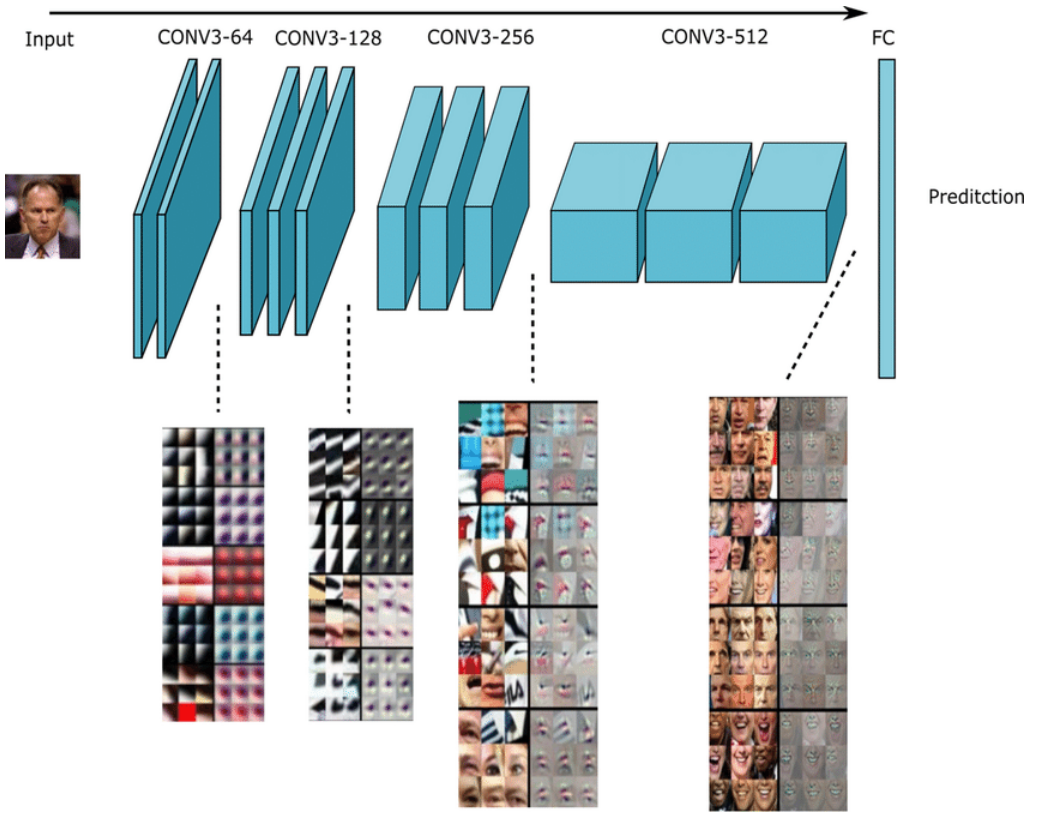


Hochschule Flensburg
University of Applied Sciences

Mei, S.; Yuan, X.; Ji, J.; Zhang, Y.; Wan, S.; Du, Q. Hyperspectral Image Spatial Super-Resolution via 3D Full Convolutional Neural Network. *Remote Sens.* **2017**, *9*, 1139.

CNNs: Hierarchical stacking of filters

- Early layers detect low-level features (edges, colors).
- Deeper layers combine them into higher-level abstractions (objects, semantics).
- This hierarchical composition is key to the success of CNNs in visual tasks.
- In transfer learning settings, the early layers can be reused, as they commonly learn very frequent and generalizing patterns.



10.1007/s43681-021-00108-6

Hochschule Flensburg
University of Applied Sciences

Advantages and disadvantages of CNNs

Hochschule Flensburg
University of Applied Sciences

- CNNs are especially prevalent in the image domain.
- This is because the concept of finding local patterns to combine is especially sensible for images.
- They are also very efficient.
- However, they have also a couple of drawbacks:
  - Limited Receptive Field: CNNs have a hard time learning long-range relationships in the image.
  - If they do learn long-range dependencies, then this is commonly achieved by spatial pooling paradigms, which destroy spatial cues to achieve this.
  - A combination of orientational cues across the image is hardly possible with CNNs.

Transformers (Vaswani et al., 2017)

Hochschule Flensburg
University of Applied Sciences

- Transformers overcome the issues that CNNs face for some tasks by:
  - Replacing local convolutions with attention mechanisms that allow any position to interact with any other.
  - Thus, they apture global dependencies and contextual relationships directly.
- The Attention mechanism enables the model to learn interaction between input variables directly, rather than indirectly.
- As an additional benefit, Transformers are fully parallelizable in training.

The attention mechanism: Key to transformers

Hochschule Flensburg
University of Applied Sciences

- The attention mechanism tries to answer the question:
  - Which part of the input data, makes sense in relation to which other part of the input data (matching of keys and queries)
  - Which information does this carry (values).
- You can understand it as:
  - Each input data acts as a "query" to our attention mechanism.
  - For each input data word, we thus want to match this against stored "keys".
  - For each stored key, we carry a "value" (the information it carries)

query →

key1	value1
key2	value2
key3	value3
key4	value4
key5	value5

```
Attention is thus in a way
a fancy "lookup" mechanism.
```

Attention

- Attention is commonly written as:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where

  - Q are the queries
  - K are the keys to match against the queries
  - V are the values (carrying the information to each key)
  - d_k is the embedding dimensionality.

Hochschule Flensburg
University of Applied Sciences

One interpretation of attention

- Let's assume we have a single query: $Q \in \mathbb{R}^{1\times E}$, where E is the dimensionality of the embedding (i.e., the vector length).
- We can compare this against a number N of keys $K \in \mathbb{R}^{N\times E}$.
- One way to do a vector comparison is by using a dot product (scalar product).
- If they match, then the scalar product is 1.
- We can split up the formula for attention into:

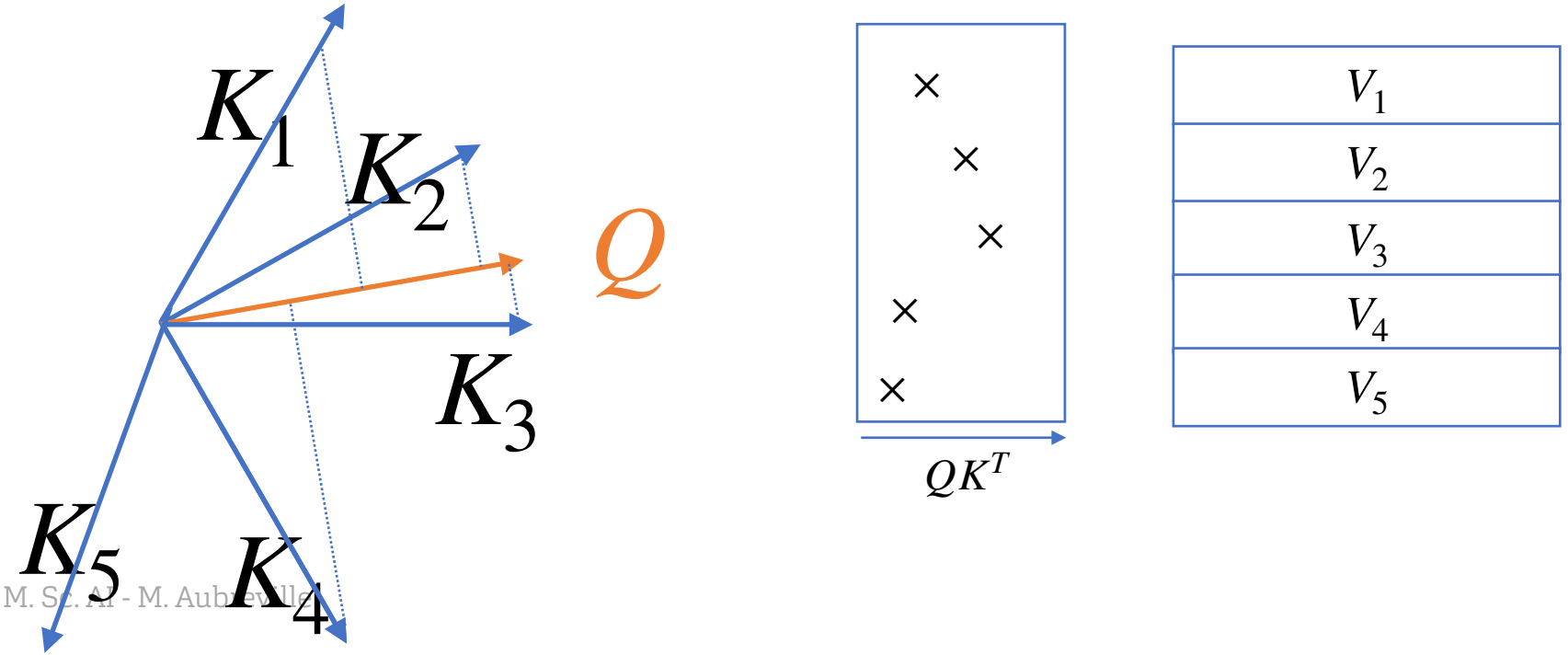
$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V = \text{softmax}\left(\frac{1}{\sqrt{d_k}}\left[QK_1, QK_2, \ldots QK_N\right]\right)V$$

  - This is nothing but a number of dot products between the query and all keys.
  - If one result perfectly matches, we scale the respective value by 1.

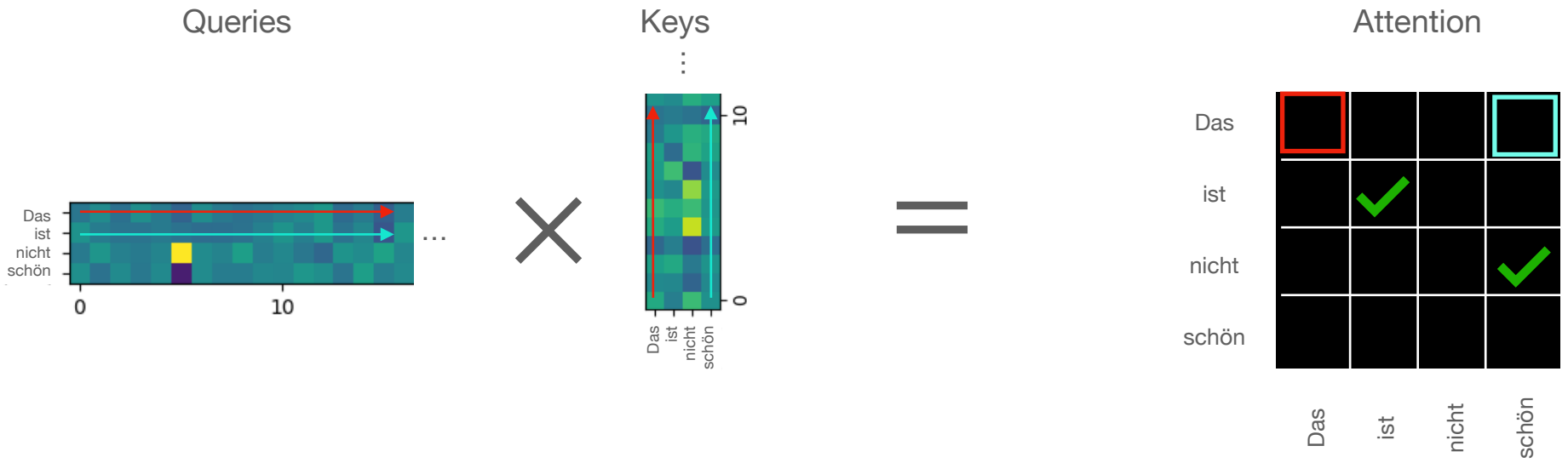
One interpretation of attention

- This process is not binary but soft.
- We can imagine attention to project the queries Q onto the keys K_n:

$$\text{softmax}\left(\frac{1}{\sqrt{d_k}}[QK_1, QK_2, \ldots QK_N]\right)V$$



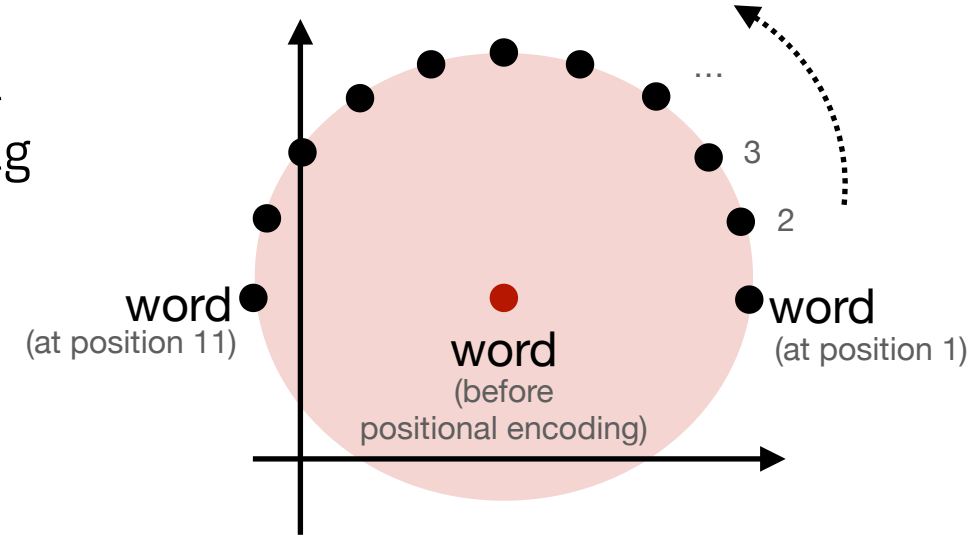
Attention



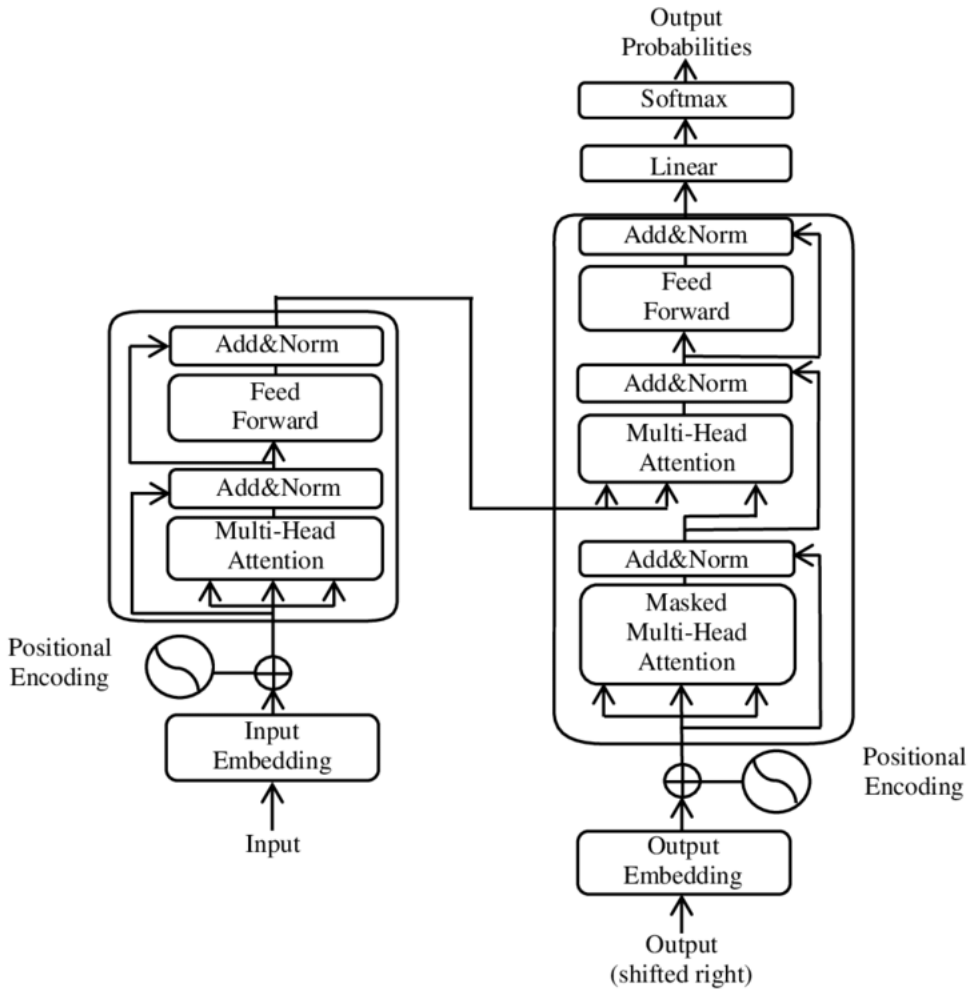
- Each token results in a unique key and a unique query from the projection.
- The attention arises as the result of a matrix multiplication of both.
- BTW: This also means that the attention is independent of the position.

Positional Encoding

- Since the attention mechanism itself is independent of positions, we have to provide it with the information of where which input was located.
- We achieve this, for example, by rotating the encodings around their original point in the latent space.
- This does a little modification to the input data, just enough so the model can distinguish it from occuring at another position.
- The words/token remain distinguishable.



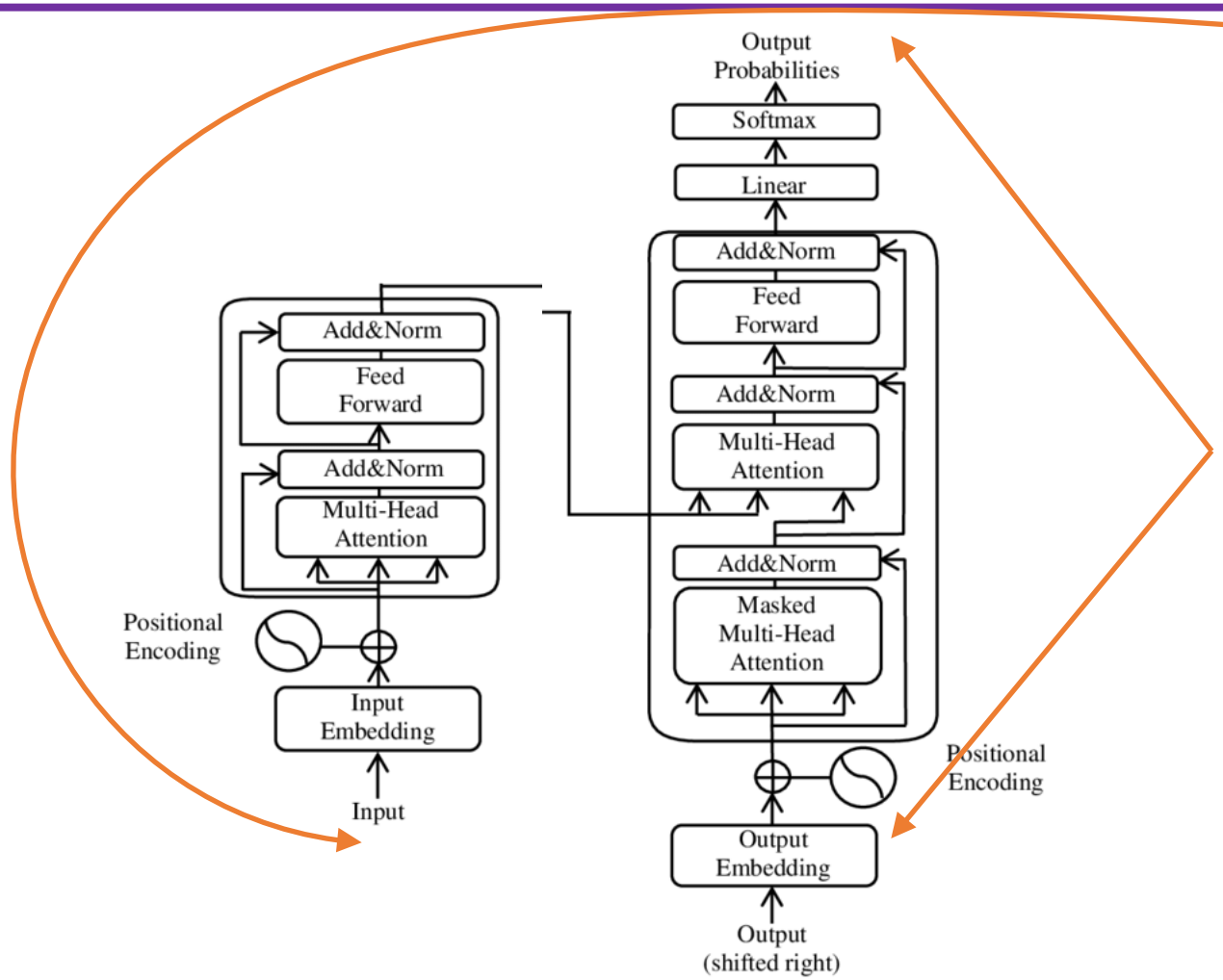
The transformer architecture



- The original Transformer architecture was developed for a Seq2seq (sequence-to-sequence) task.

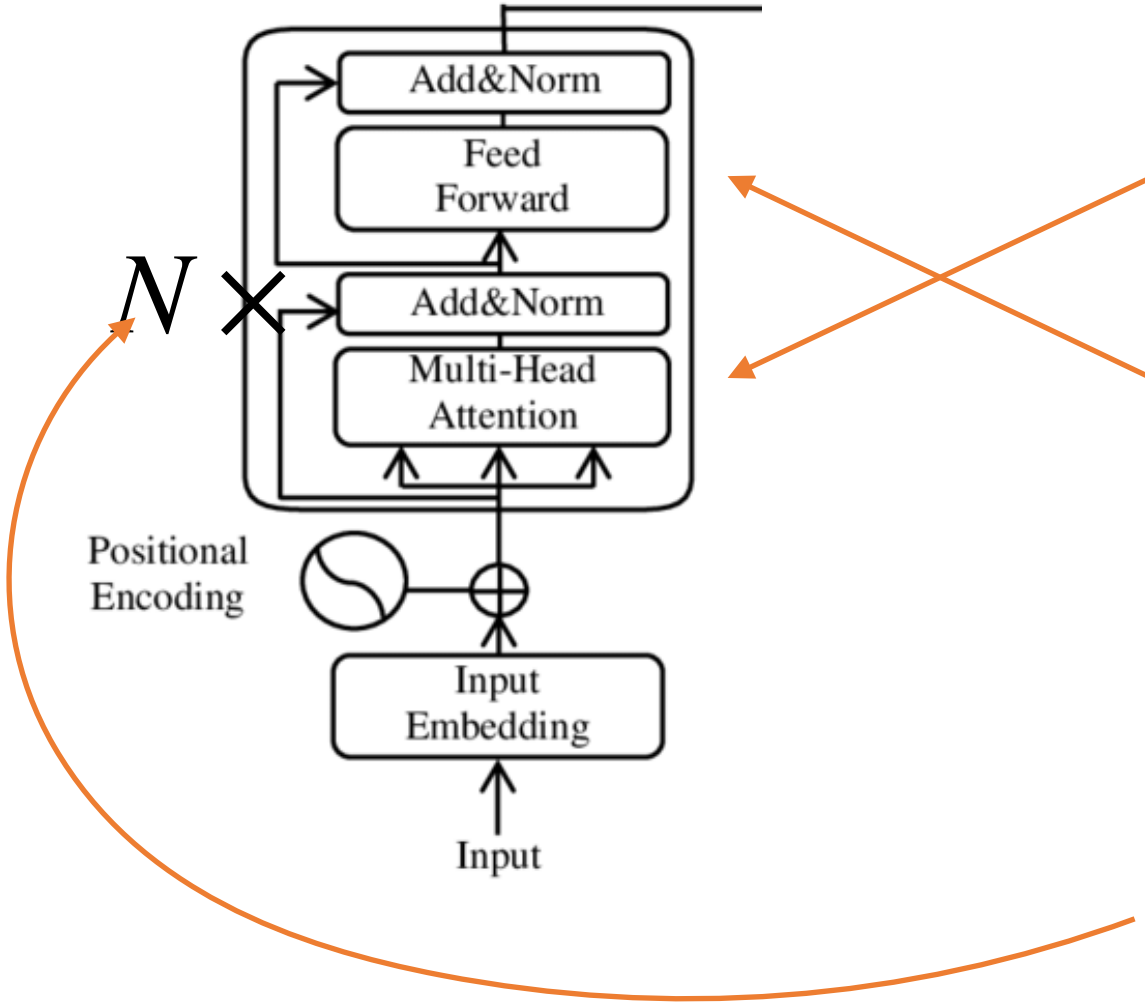
- Seq2seq tasks include:
  - Translation
  - Text summarization
  - Image captioning
  - Text-to-image

Der Transformer



- The token embeddings with positional encoding form the input.
- The sequences are generated in an autoregressive manner, i.e.
  - The previously produced output tokens are fed into the network
  - This temporal relationship enables consistent output.

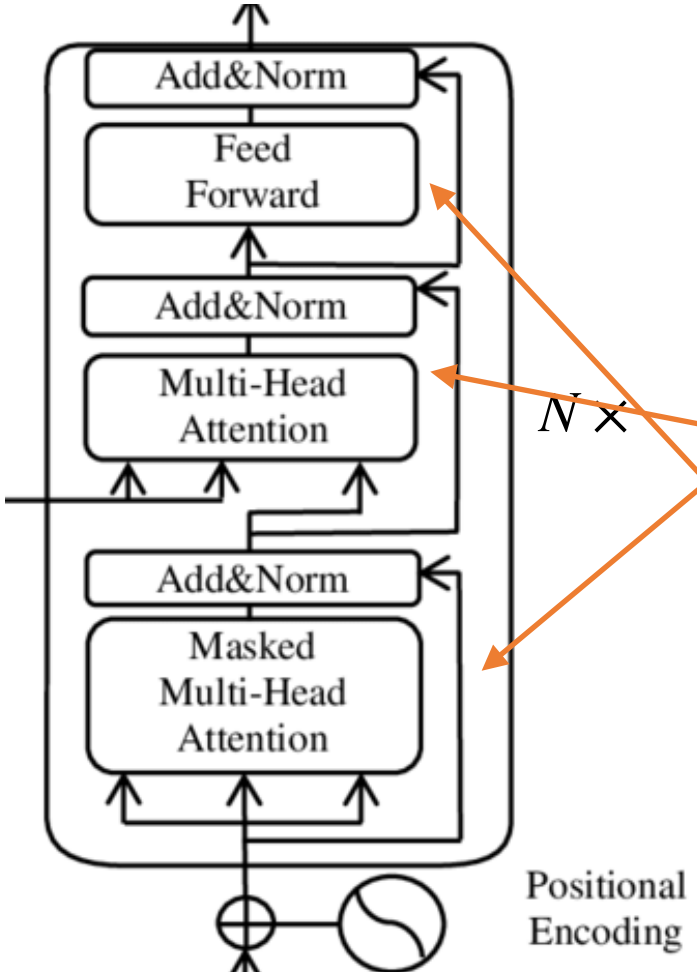
Der Transformer: MHSA



- The transformer uses multi-head self-attention (MHSA) to model relationships within the input sequence.
- Each MHSA block is followed by a feed-forward block to model the relationship between self-attention and the desired semantic latent representation, which we need for the translation task, for example.
- This Transformer block is used multiple times in a residual setup with normalization.

Hochschule Flensburg
University of Applied Sciences

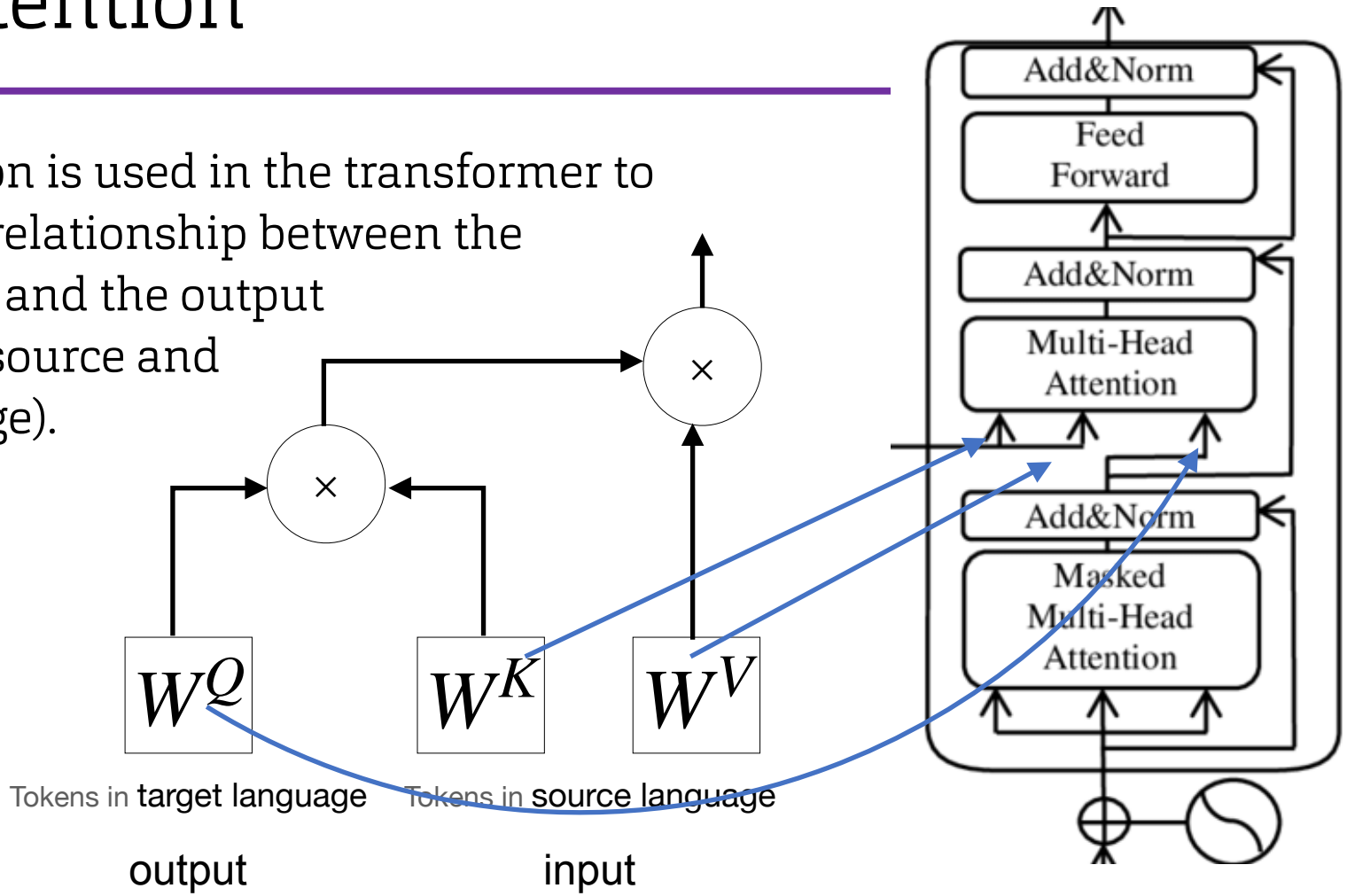
Der Transformer: Ausgabeblock



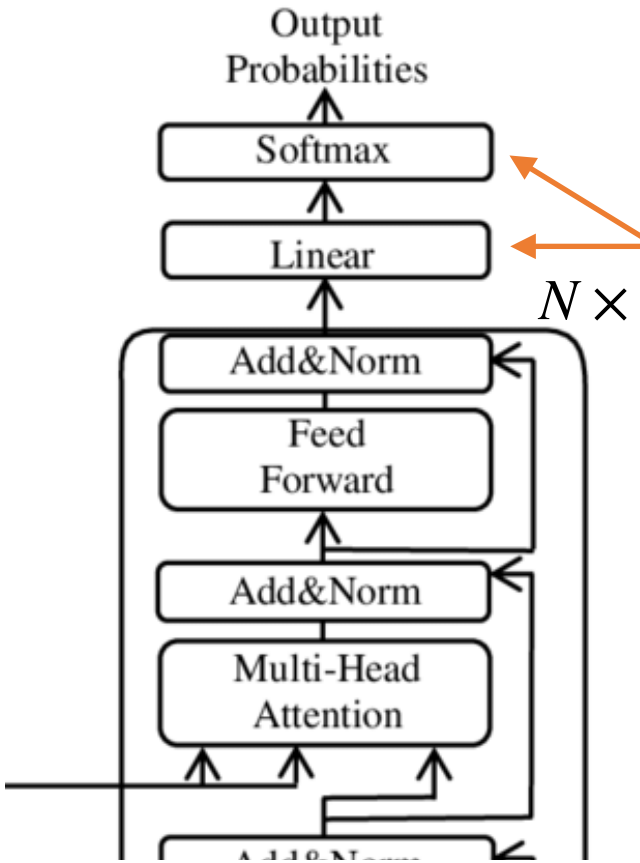
- The output branch initially uses the same blocks for self-attention, but masked: This ensures causality, i.e., that the current prediction always depends only on previous tokens.
- Then, the cross-attention of the input encoder block is used to inform the model about the input sequence (i.e., previous outputs and new inputs are combined).
- Finally, both are combined again with a linear forward projection with GELU activation.

Cross-Attention

- Cross-attention is used in the transformer to establish the relationship between the input domain and the output domain (e.g., source and target language).



Transformer: Projektionen



- Finally, we use a linear projection to convert back to the input space.
- The output tokens are simply generated using the softmax() operator (and finally argmax to select a single token).

Summary: Model architectures

- RNNs: Process sequences step-by-step; capture short-term dependencies.
- LSTMs: Add gating to store long-term information; fix vanishing gradients.
- CNNs: Use local convolutions and shared filters to detect spatial patterns.
- Transformers: Use self-attention for global, parallel context modeling.

RNN	LSTM	CNN	Transformers
sequential processing	Long-term memory retention	Local spatial patterns	Global context and relationships

Hochschule
Flensburg
University of
Applied Sciences

Autoregressive Models

Recap: Generative Models

- We are given a set of examples, e.g., images of cats.

- We want to learn a model distribution $p(x)$ over the images x_i, such that:
  - We can sample a new image $x_{\text{new}} \sim p(x)$ that actually looks like a cat image.
  - Ideally, we want to be able to control how the image looks like.

$x_i \sim p_{\text{data}}$ $\quad i \in [0,1,\ldots,n]$

Hochschule Flensburg
University of Applied Sciences

Example: MNIST

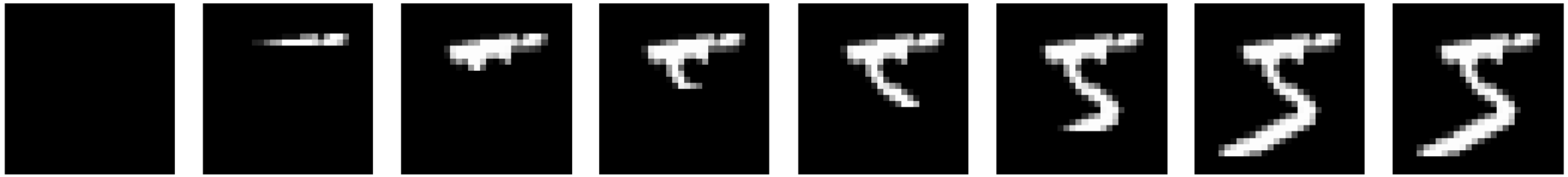
- Let's take the example of hand-written digits in the MNIST dataset:

- Each image has 28x28 pixels, so we can represent it by a 784-sized vector.
- We have the goal to learn a probability distribution vector of size 784:
  $\mathbf{p}(x) = [p_1(x), p_2(x), \ldots, p_{784}(x)]$
  over $x \in \{0,1\}^{784}$ such that when we sample from it, it looks like a digit.
- We will be using a model for this.

Autoregressive sampling

Hochschule Flensburg
University of Applied Sciences

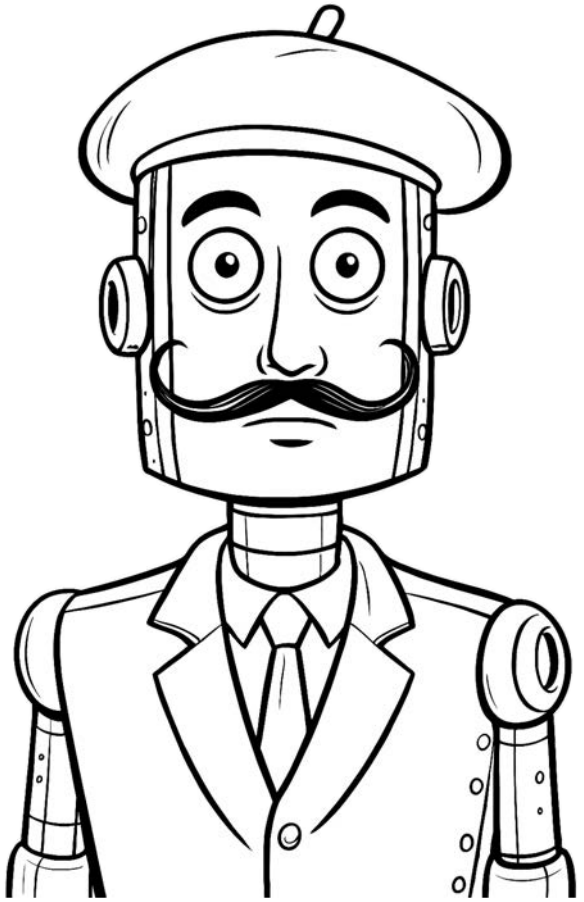
- When we sample from the distribution, we can start by sampling the first pixel x_1.
- Given the value of the first pixel, we can move on and predict the second pixel $x_2 \sim p(x_2 \mid x_1)$.



- The third pixel already depends on the first two pixels: $x_3 \sim p(x_3 \mid x_1, x_2)$
- And we go on until we have all pixels predicted.
- This is called autoregressive sampling.

Autoregressive Models

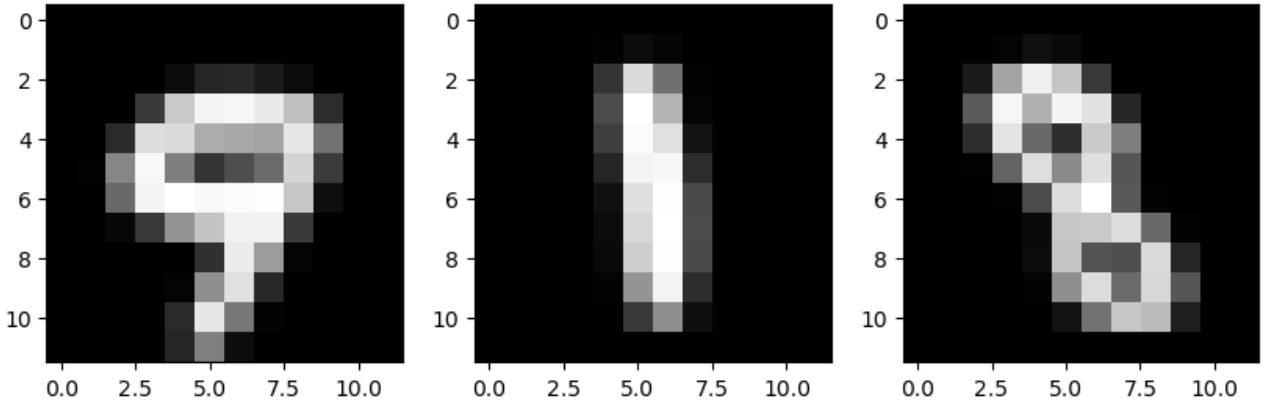
- Autoregressive models are generative models that model the distribution in an autoregressive way.
- We can use those to generate new samples - be it text, images, or whatever.
- We could use a range of models for this:
  - Recurrent Neural Networks (RNNs), originally used in sequence modeling tasks like language modeling
  - (Masked) Convolutional Neural Networks
  - Transformers (more recently)



Salvatore AI

Example: Using an RNN to generate an image

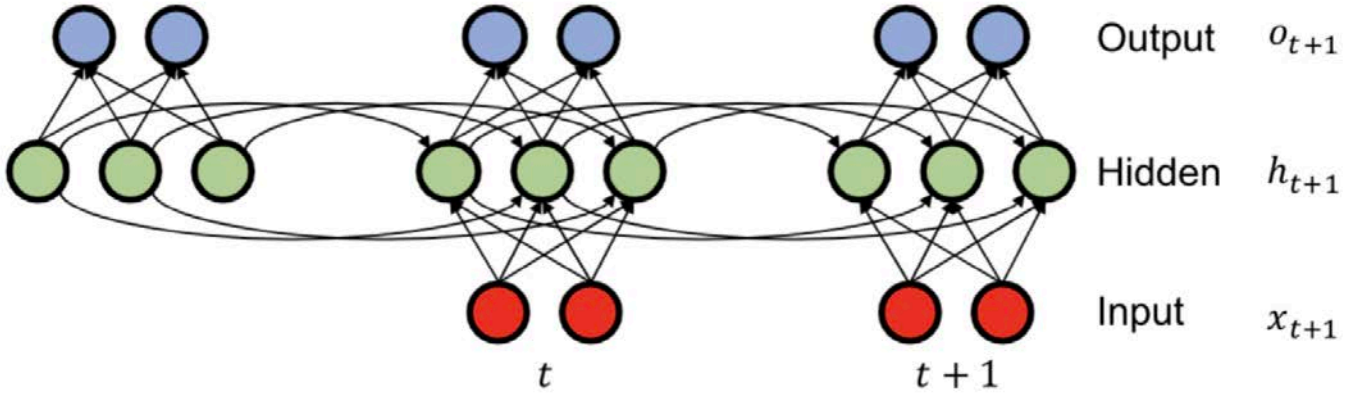
- In the following, I want to generate images using a RNN (in this case: an LSTM)



- To make things a bit simpler, we use a 12x12-version of MNIST.

Recurrent Neural Networks (RNNs)

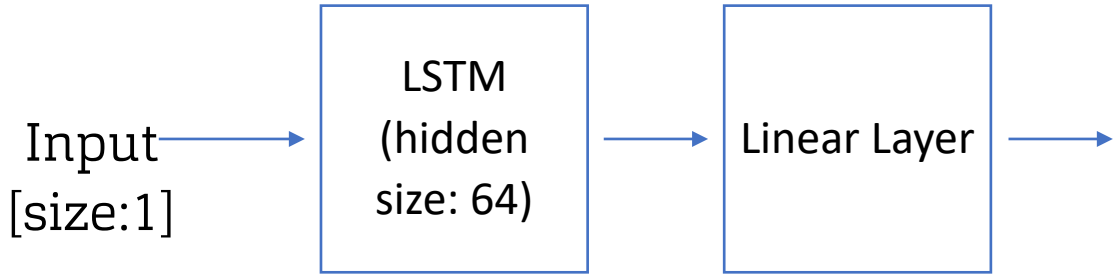
- Since in our prediction task, the history is
  - not of constant length
  - getting longer with every prediction
- It is not well-suited for typical feed-forward networks, which expect a fixed-size input.
- Instead, we can utilize RNNs, which are made for sequence modeling tasks.



Let's try to code this

Hochschule Flensburg
University of Applied Sciences

- We want to have a very simple model:



- The model shall always predict the next pixel.

DataLoader

```python
# MNIST laden, auf 12x12 skalieren
transform = transforms.Compose([transforms.Resize((12, 12)), transforms.ToTensor()])
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)


class MNISTVectorDataset(torch.utils.data.Dataset):
    def __init__(self, mnist_dataset):
        self.mnist_dataset = mnist_dataset
    def __len__(self):
        return len(self.mnist_dataset)
    def __getitem__(self, idx):
        img, _ = self.mnist_dataset[idx]
        img = img.squeeze()  # [12, 12]
        vec = img.view(-1)  # [144]
        return vec

train_loader = DataLoader(MNISTVectorDataset(train_dataset), batch_size=batch_size, shuffle=True)
```

- The data loader will resize MNIST images to 12x12, and then reshape to 144.

Model

```python
import torch
import torch.nn as nn

# LSTM-Modell für Vektor
class VectorLSTM(nn.Module):
    def __init__(self, input_size=1, hidden_size=64, num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, 1)
    def forward(self, x):
        # x: [batch, seq_len, 1]
        out, _ = self.lstm(x)
        out = self.fc(out)
        return out.squeeze(-1)  # [batch, seq_len]
```

- This simple model uses an LSTM with hidden state size of 64 and a single layer.
- Finally, it has a linear layer to project the hidden state into the output.

Training Script

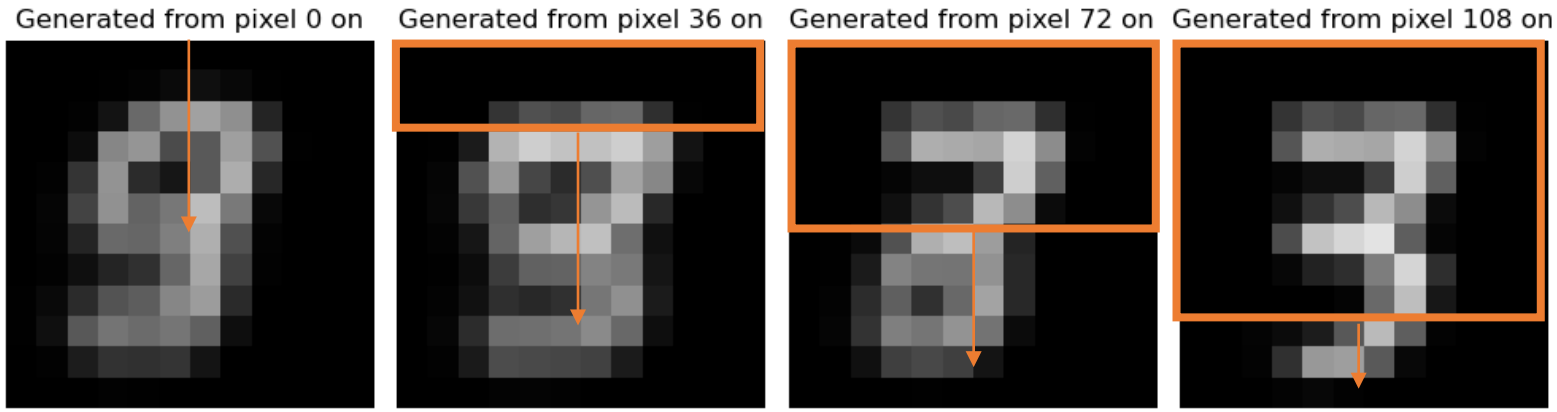
```
device = torch.device('mps') # mps is on Macs, use cuda if available, or cpu else.
model = VectorLSTM().to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=lr)

# Training
for epoch in range(epochs):
    for vecs in tqdm(train_loader):
        vecs = vecs.to(device)  # [batch, 100]
        optimizer.zero_grad()
        # Input: alle Pixel außer letztem
        input_seq = vecs[:, :-1].unsqueeze(-1)  # [batch, 99, 1]
        # Target: alle Pixel ab zweitem
        target_seq = vecs[:, 1:]  # [batch, 99]
        output = model(input_seq)  # [batch, 99]
        loss = criterion(output, target_seq)
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f}")
```

- We train the script using BCE, note that L1 loss is just as well possible.

The result

- To play around with the model, we generate using a sample (depicting the number 9).
- We either generate from Pixel 0, or from pixel 36 (row 3), or from pixel 72 (row 6), or from pixel 108 (row 9).



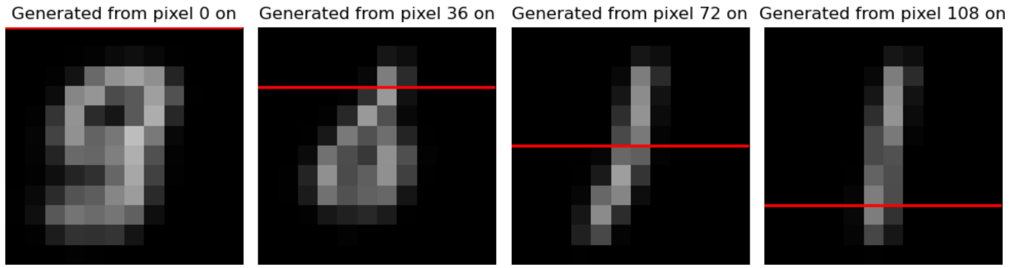
- Note how much the model benefits from having a well-defined "starting point".

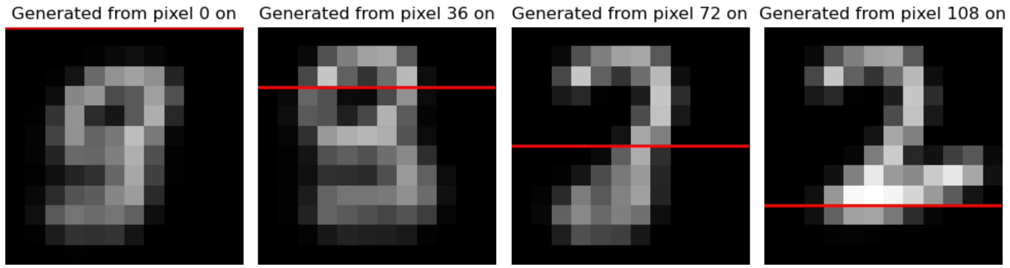
OK, now let's try to enlarge this ...

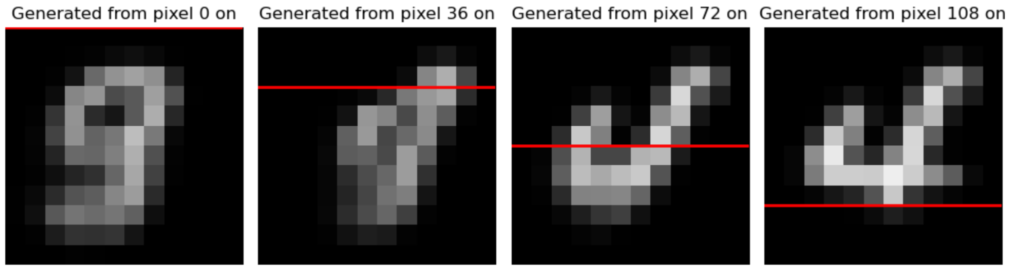
- Let's go from 12x12 to 24x24 (vector size: 144 to 576)
- How well do we expect our model to do?

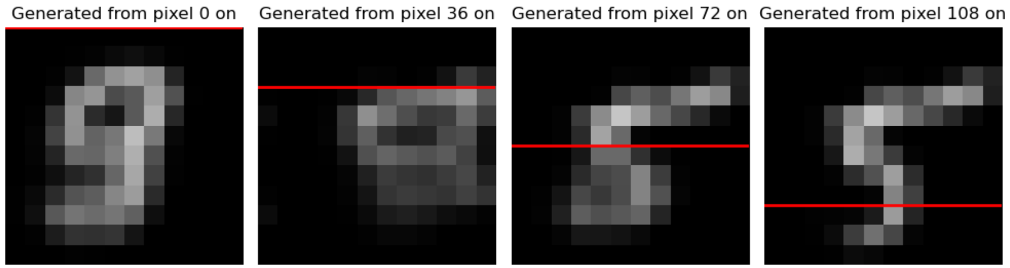
How does it continue other digits?

- Let's try this out:

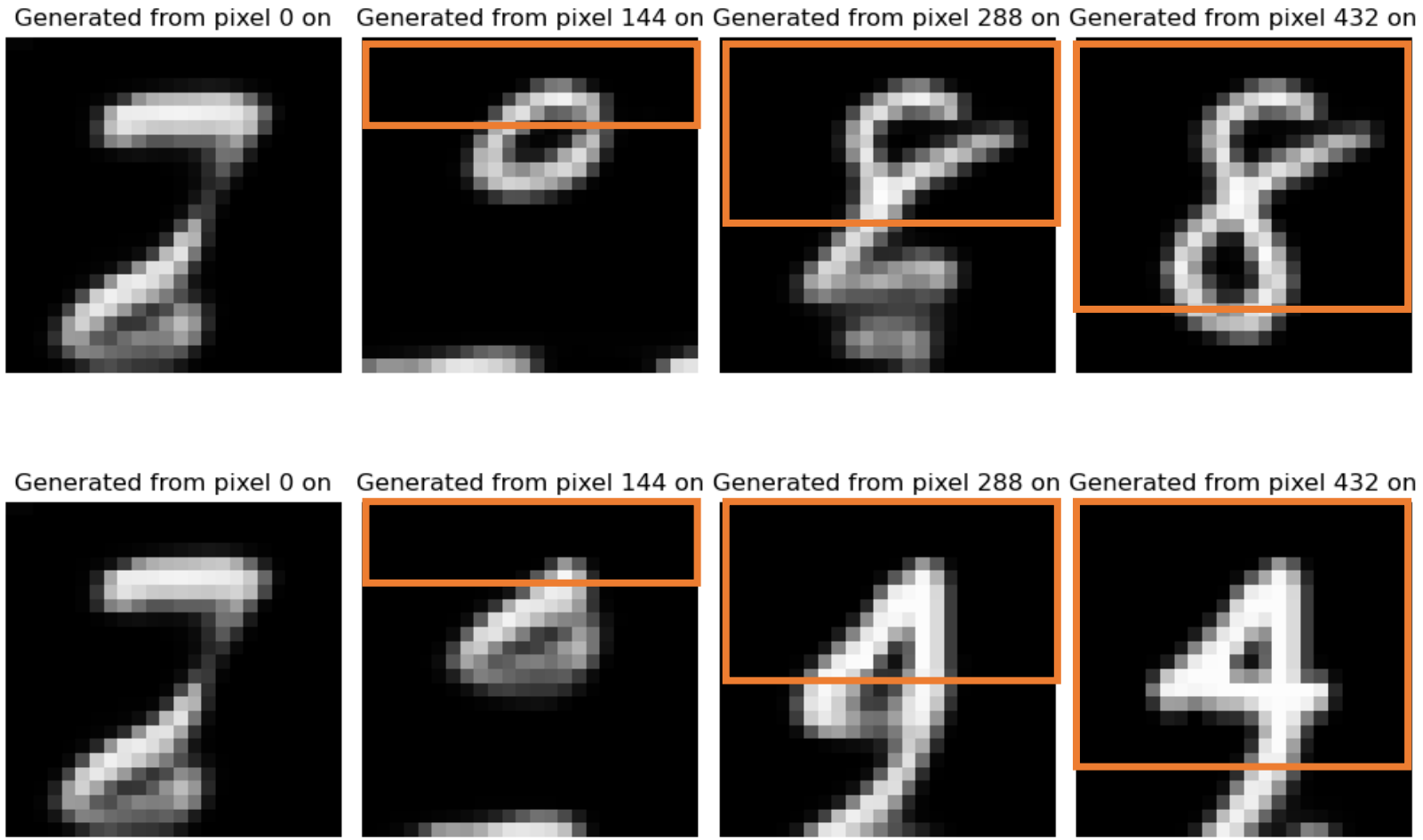






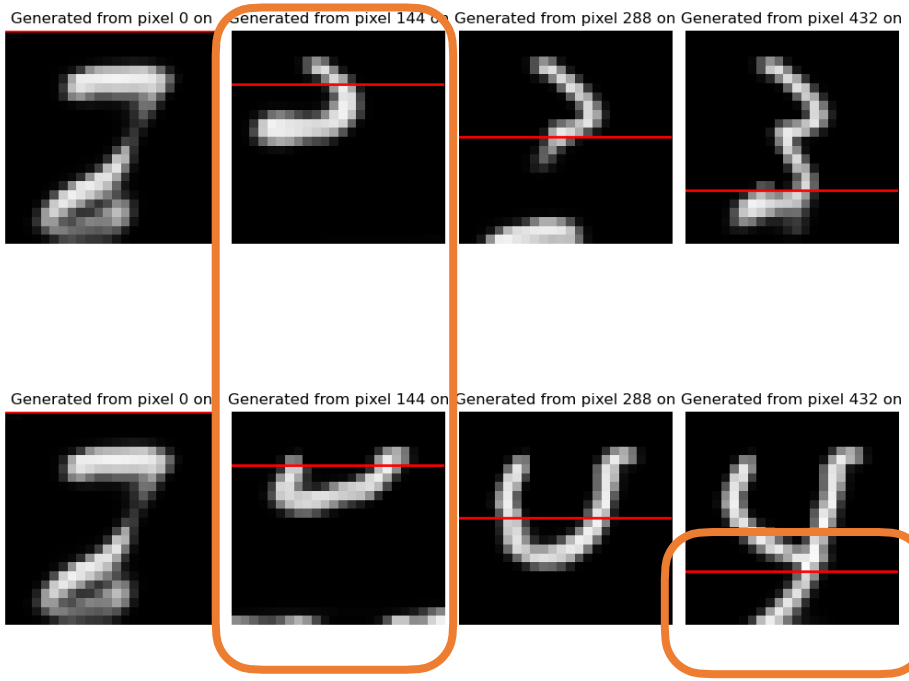


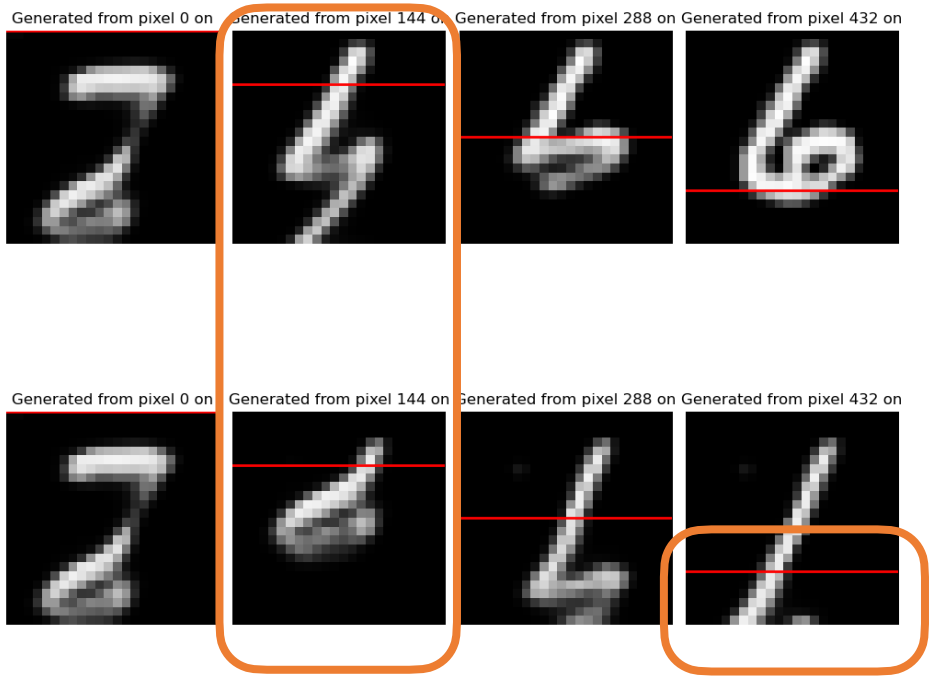
Training using 24x24 sized images



What can we observe?

More examples





What happened here? And why?

Observations

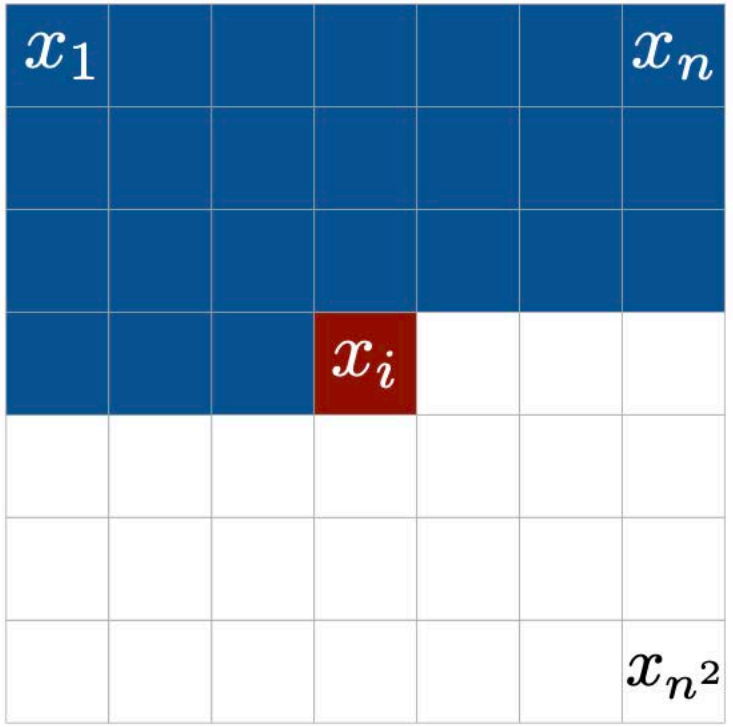
- We can see that the model has a hard time producing the entire image.
- This can be linked to a short attention span - i.e., the model has a very limited receptive field in the recursive network.
- Furthermore, we can see that the model's output was always fully deterministic (let's look into this later).

Hochschule Flensburg
University of Applied Sciences

PixelRNN: Applying RNNs to Images (van den Oord et al., 2016)

Hochschule Flensburg
University of Applied Sciences

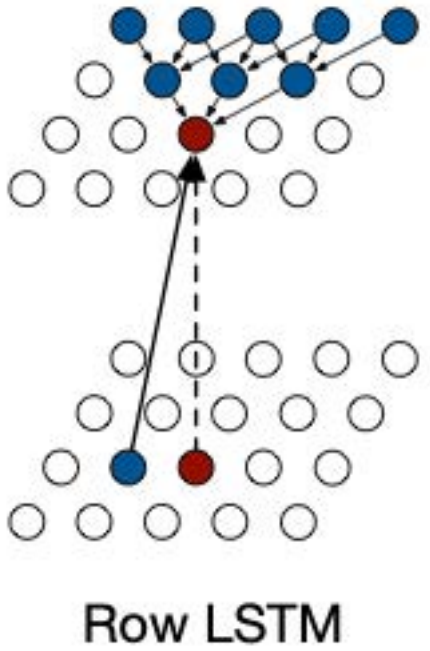
- PixelRNN models the conditional distribution of each pixel given all previous pixels.
- Each pixel is conditioned on all previously generated pixels.
- The completely sequential generation, however, comes with a couple of drawbacks:
  - Long-range dependencies are prone to vanishing gradients
  - Sequential generation is very slow (and not parallelizable)



PixelRNN: Using more context: RowLSTM

Hochschule Flensburg
University of Applied Sciences

- PixelRNN proposes to not only use pixel-wise regression, but row-wise regression.
- Unidirectional sweep: process rows top→bottom; within each row, scan left→right.
- Causal feature extractor:
  a masked 1-D conv (along columns) produces input-to-state terms for all positions in the row, using only pixels above or to the left (no peeking right/below)
- LSTM aggregation: an LSTM runs across the row (left→right), carrying long-range horizontal context; stacking layers deepens vertical context (via the masked conv inputs).
- Causality for channels: predict sub-pixels in a fixed order (R, then G, then B) with appropriate masks so each channel only sees previously generated values.



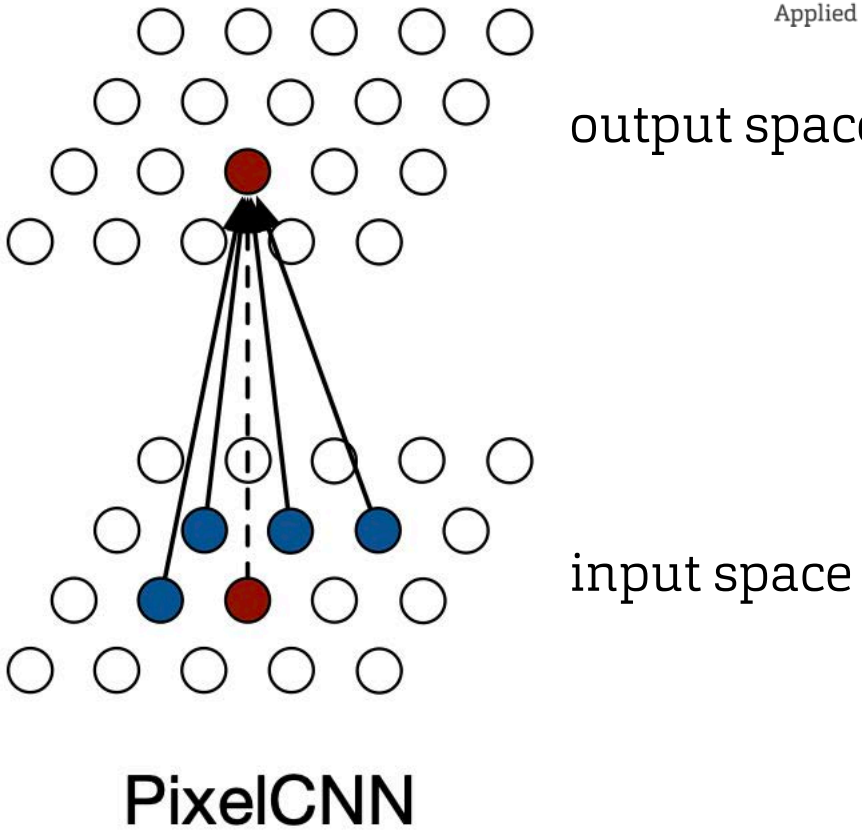
RowLSTM: Why it is better than pixel-wise prediction

Hochschule Flensburg
University of Applied Sciences

- Aggregation using two mechanisms (convolution over columns, LSTM over rows) achieves a larger effective receptive field than pixel-wise regression at the same depth.
- Recurrent state captures long horizontal dependencies (lines, strokes, edges).
- Training is done parallel over pixels - sampling remains sequential but faster than full 2-D recurrences (parallel across rows to some extent).

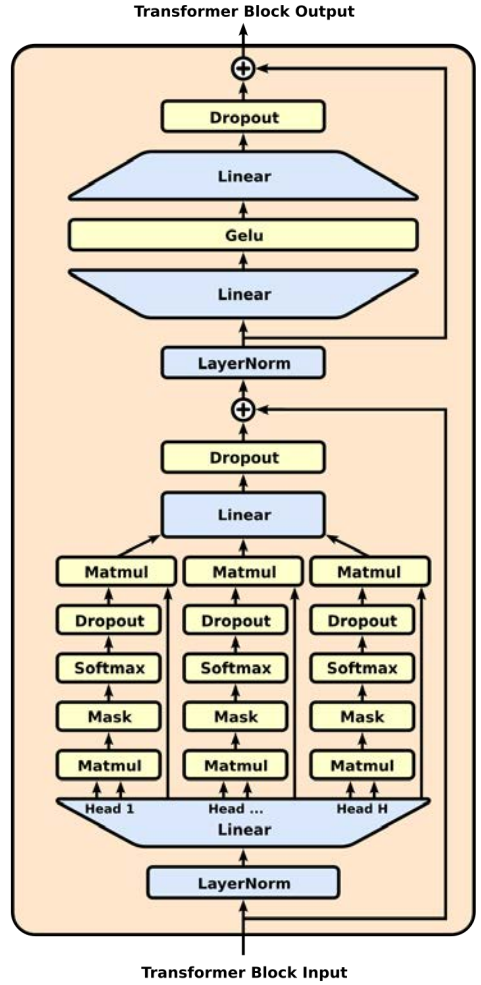
PixelCNN

- In the same paper, van den Oord et al. proposed also to use a convolutional architecture to perform image generation.
  - PixelRNN, while effective, was computationally expensive due to its sequential nature.
  - PixelCNN was introduced to make training and generation more parallelizable.
- Ideas:
  - Replace recurrent layers with masked convolutions.
  - Each pixel is predicted only from previously generated pixels (enforcing autoregressive dependency).



Auto-regressive Model: GPT

- The transformer architecture is also employed in autoregressive generation.
- The most notable application is large language modeling, and here, in particular the generative pretrained transformer (GPT).
- GPT models apply the autoregressive principle using the Transformer decoder architecture.
- Each token predicts the next one, conditioned on all previous tokens, but instead of recurrence, GPT uses causal self-attention.



Summary: From PixelRNN to GPT

- GPT keeps the autoregressive idea but replaces convolution with attention.
- Enables long-range dependencies and domain-general modeling.
- The same principle later extends to ImageGPT, etc.
- PixelCNN taught models to predict one pixel at a time.
  - GPT made that principle universal
  - predicting any next element in a sequence, across any domain.

Hochschule Flensburg
University of Applied Sciences

Hochschule
Flensburg
University of
Applied Sciences

Masked Prediction

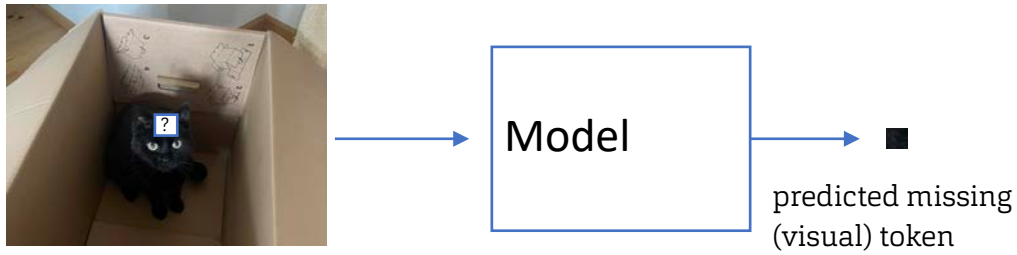
Limitations of autoregressive models

- Autoregressive models (PixelRNN, PixelCNN, GPT, etc.)
  - Generate data sequentially, one token/pixel at a time.
  - Each step conditions on all previous outputs.
  - Provides exact likelihood but is slow at generation.

- Limitations of Autoregressive Generation
  - Sequential bottleneck: must predict in order → no parallel decoding.
  - Local dependencies: harder to capture global structure early.
  - Inefficient sampling: especially for high-resolution images.

Masked Prediction Models

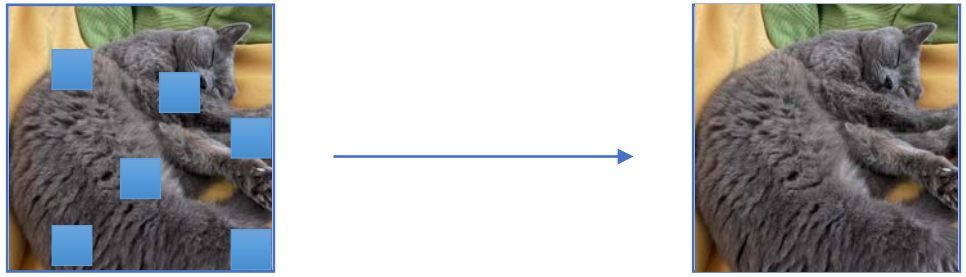
- Instead of generating sequentially, we predict missing parts of the data given the rest.



- Training objective:
  Randomly mask a subset of pixels/tokens → model learns to fill in the missing ones.
- Advantages:
  - Parallelizable over all masked positions.
  - Learns bidirectional or global context.

Making use of the global context

- Instead of “painting left-to-right,” the model “fills in the blanks” anywhere in the image.
- Randomly mask a subset of pixels (or tokens) in the input.
  - The model learns to reconstruct the masked parts given the visible context.
  - This utilizes bidirectional dependencies (and potentially global context)



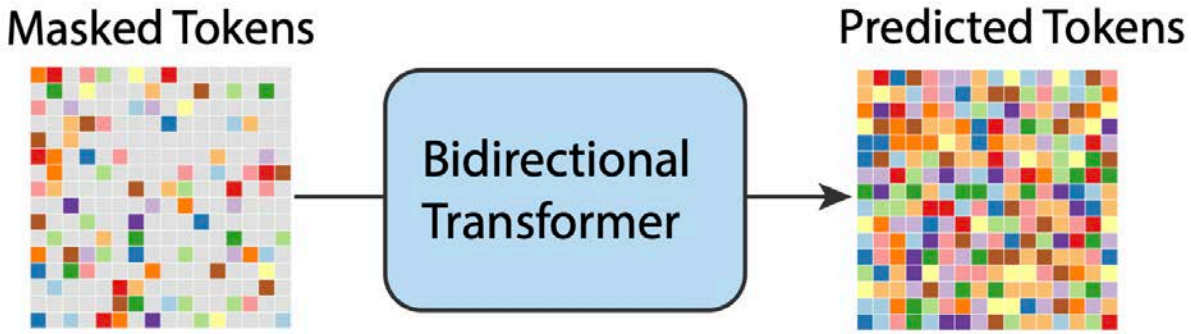
Mask selection

Select 15-40% of tokens to be masked

Prediction

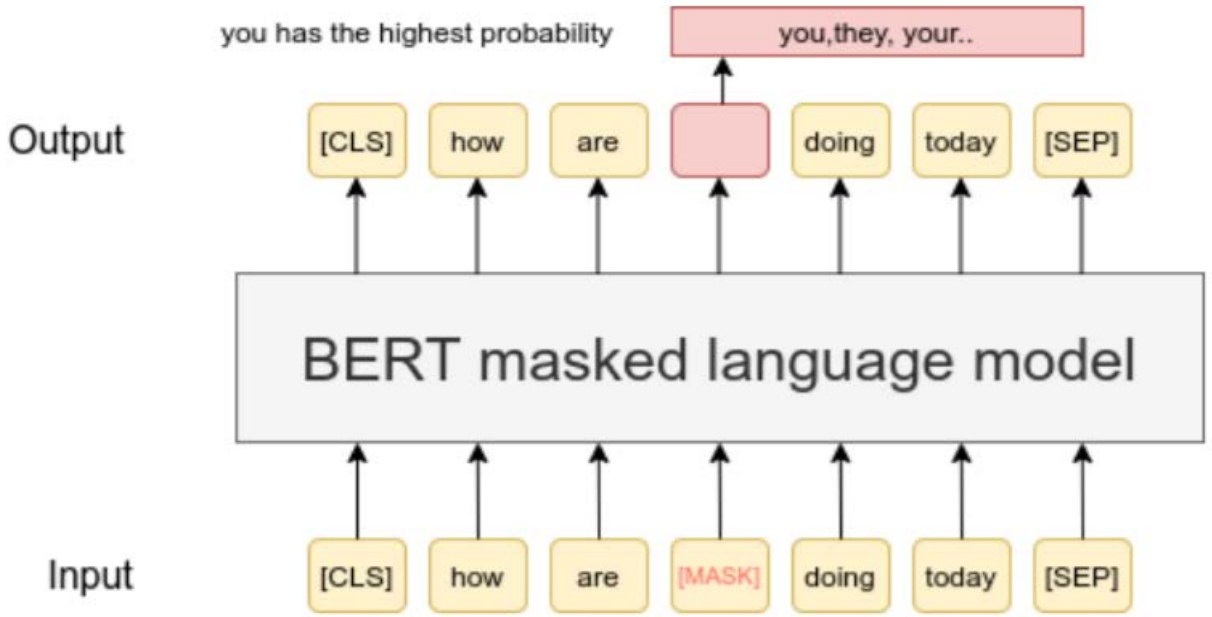
Inference for Masked Prediction Models

- There are two main ways masked models generate new content:
- One-shot reconstruction:
  - Fill in all masked positions in a single forward pass (fast but approximate).
- Iterative refinement:
  - Predict masked tokens -> update -> re-mask uncertain ones -> repeat (e.g., MaskGIT).
  - This allows coherent, high-quality generation — parallel but iterative.



Case study: BERT

- Goal: Learn deep bidirectional contextual representations of text by predicting missing words.
- Input sentence: some words are randomly masked (e.g., 15%).
- The model predicts the original words from surrounding context.
- Objective: maximize likelihood of masked tokens only.

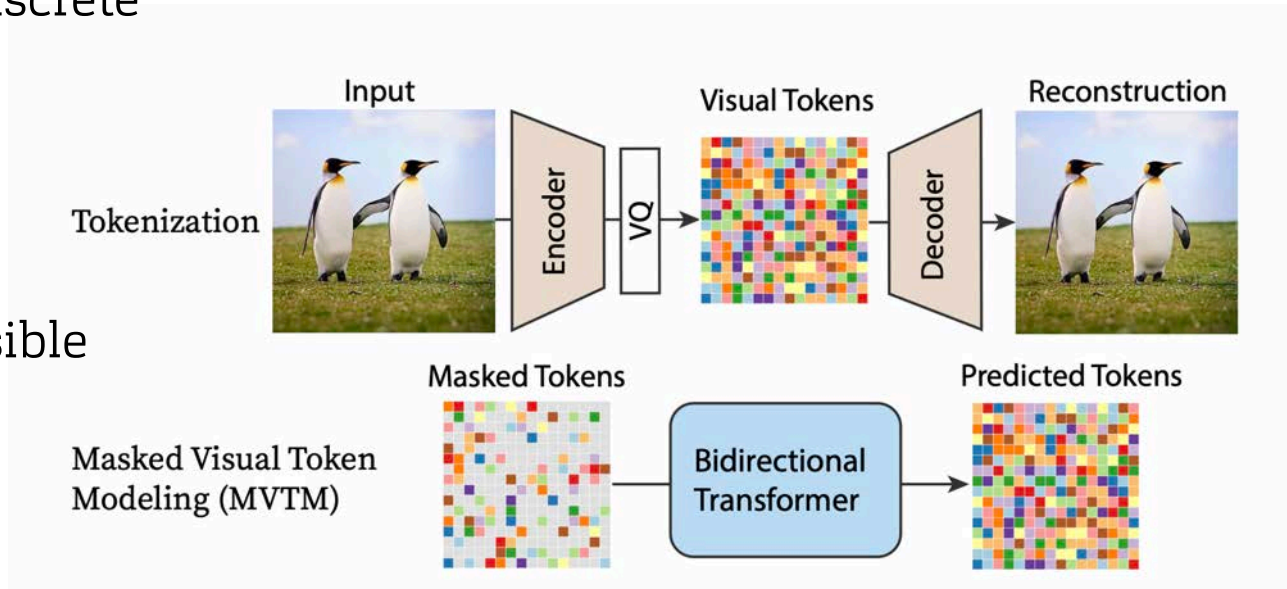


https://www.sbert.net/examples/sentence_transformer/unsupervised_learning/MLM/README.html

Case study: MaskGIT (Chang et al., 2022)

Hochschule Flensburg
University of Applied Sciences

- Fast, high-quality image generation using a masked token prediction approach
- Images are tokenized (e.g., using a VQ-VAE) into discrete codebook entries.
- During training:
  - Randomly mask a fraction of image tokens.
  - Model predicts the original tokens given the visible ones (BERT-style).
- During generation:
  - Start from fully masked tokens.
  - Predict all tokens in parallel, assign confidence scores, and progressively unmask the most confident ones.



Summary: Masked Prediction Models

- Train by masking parts of the input and predicting the missing content.
- Learn bidirectional context — each token sees both left and right neighbors.
- Enable parallel training across all masked positions.
- Can generate via iterative refinement (e.g., MaskGIT) or one-shot filling.

Hochschule
Flensburg
University of
Applied Sciences

Generation without Determinism

(in autoregressive and maskeds models)

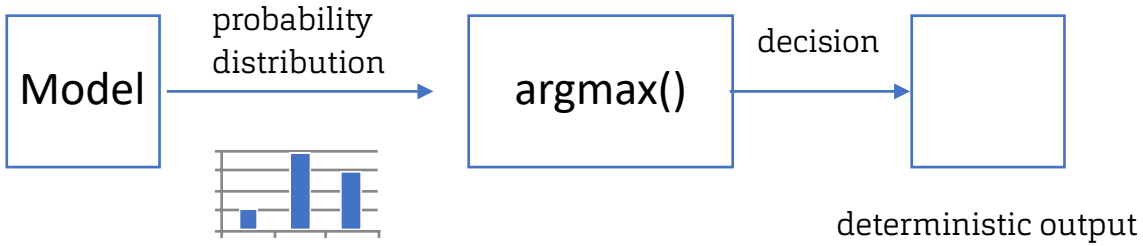
Deep Learning Models are Fully Deterministic

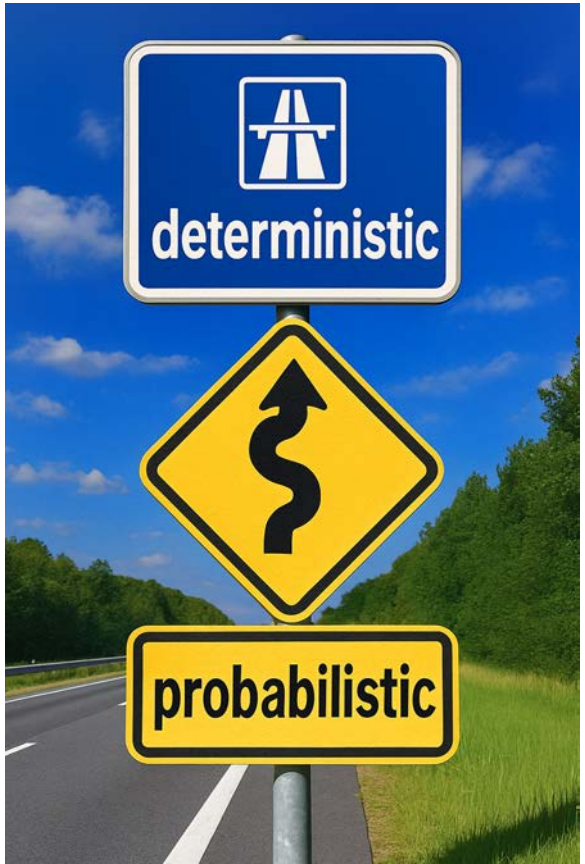
- If we start with the same initial value, we must expect that the model always answers with the same response.
- This creates an "initialization" problem for us - how can we generate more samples without knowing their distribution upfront?
- Our generation process is clearly missing **entropy**.
- We will need to incorporate **more randomness** into the generation process.

Deterministic vs. Generative Thinking

Hochschule Flensburg
University of Applied Sciences

- When we train a model (e.g., RNN on MNIST) with cross-entropy, we teach it to predict a distribution over the next pixel or token.
- But if at test time we always take the argmax (the most likely next value):
  - we get a single, deterministic completion.
  - we never explore what the distribution actually represents.
- So the model in fact does learn uncertainty, but we never use it.

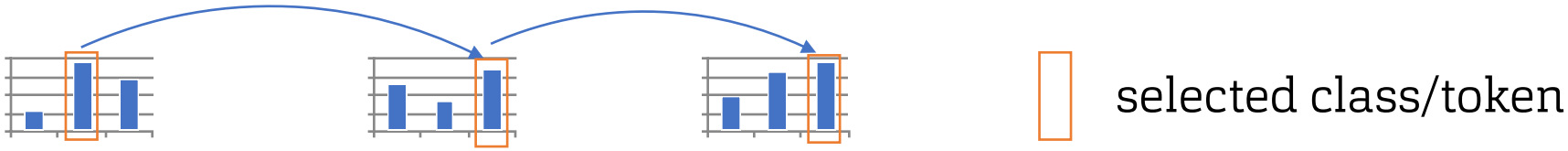




Generative Models should not be deterministic

Hochschule Flensburg
University of Applied Sciences

- Problem so far:
  - The model learns uncertainty, but we ignore it.
  - Each step collapses the entire distribution to a single choice.
  - This produces one “average” or “mode” sample — often blurry, repetitive, or dull.
  - We lose the diversity that makes generative models generative.
- We could also look at it like this:
  The model has learnt so many likely options how to go ahead, but we ignore all but one.



- Only a single possible "predictive path" is followed.

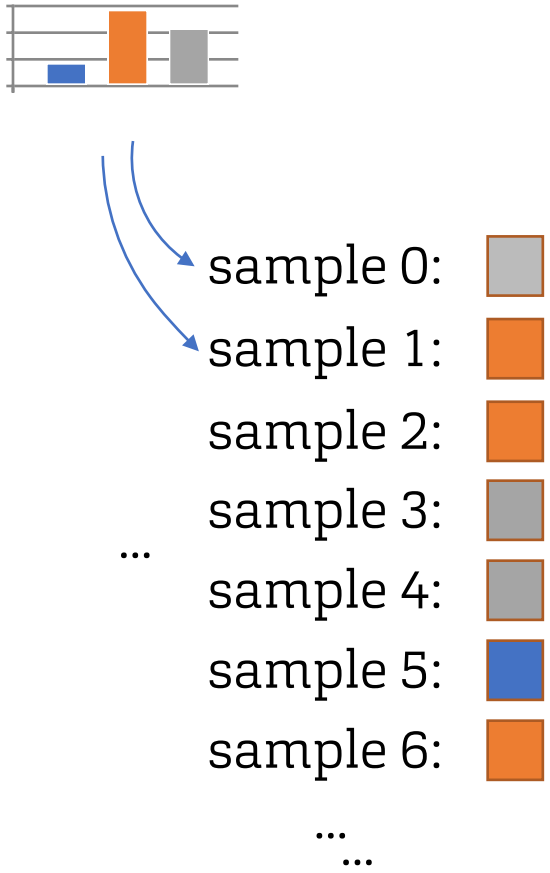
The power of sampling

- What we actually learnt, however, is a probability distribution.
- It tells us the probability a certain output should occur.
- We can thus use this, and **sample** according to this probability distribution.
- Sampling reveals what the model believes could happen — not just what's most likely.
- On average, sampling according this distribution will give us exactly the same distribution of tokens.


```
Cross-entropy teaches the model
probabilistic structure.
Sampling is how we see that
structure in action.
```

Sampling

- Sampling according to the distribution means to add entropy to the generation process.
- Each prediction step defines a probability distribution over possible next symbols.
- Instead of collapsing it to the most likely choice, we draw a random sample — introducing controlled randomness (entropy) into the sequence.
- This is **not adding noise to the generation**. We need it to utilize the distribution.



Hochschule Flensburg
University of Applied Sciences

Models can be over-confident

- However, models can become very overconfident (as discussed in ADL).
  - Especially deep learning models tend to have very high posteriors.
  - This is the result of two key factors:
    - Cross-entropy penalizes uncertain "right" decisions almost as badly as uncertain "wrong" decisions.
    - The softmax operator increases differences in logits by using the exponential function.

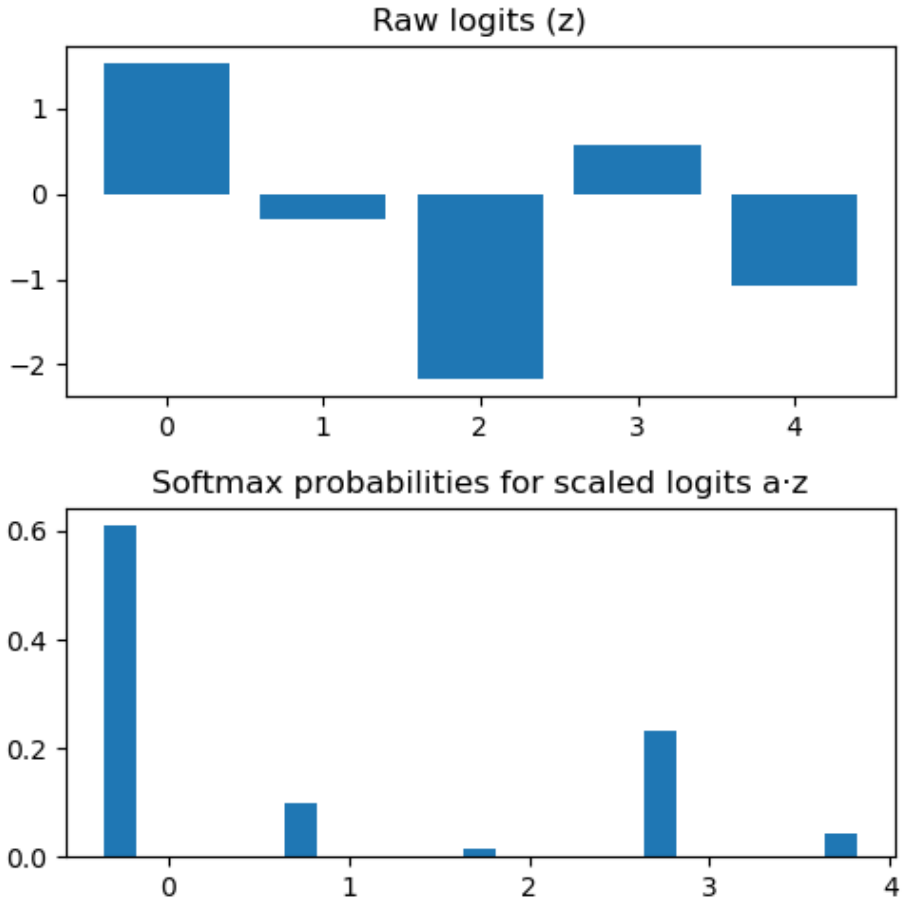
Hochschule Flensburg
University of Applied Sciences

The role of softmax in overconfident models

- The softmax function converts the model's logits z_i into "probabilities":

$$p_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

- This operation amplifies differences in logits — even small gaps in z_i produce large differences in probabilities.
- During training with cross-entropy, the model is rewarded for using high-confidence predictions.
- This leads to a reinforcement loop:
  - Cross-entropy rewards higher confidence.
  - Softmax exponentially magnifies those logit gaps.
  - Deep networks push logits to extreme values to minimize loss.



Temperature Scaling

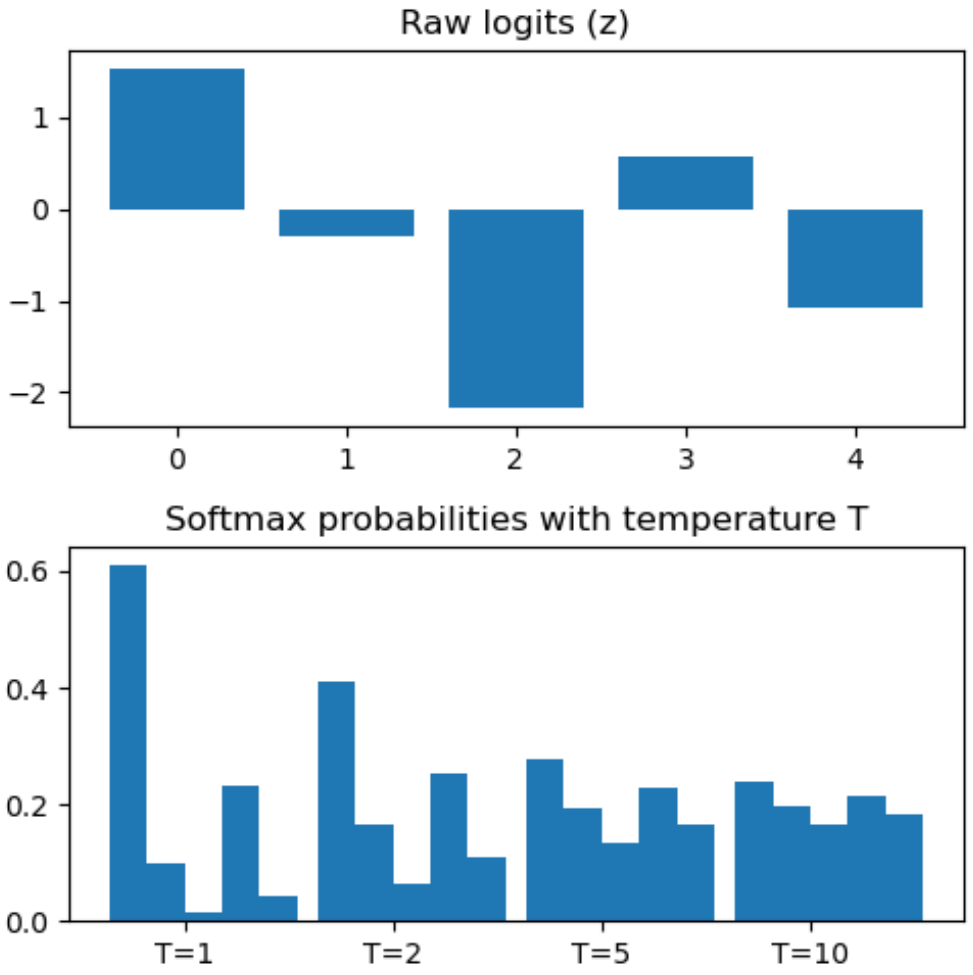
Hochschule Flensburg
University of Applied Sciences

- Temperature scaling is one method to enrich the diversity of a model's sampling process.
- The core idea is to scale the logits before the softmax using a scaling parameter:

$$p_i = \frac{e^{z_i/T}}{\sum_k e^{z_k/T}}$$

- The parameter T is called the temperature. We can use it to control the sampling process:
  - $T \to 0^+$: Distribution becomes pointed (low entropy)
  - $T = 1$: No change
  - $T > 1$: Flatter distribution, higher entropy, more data variance in sampling
  - $T \to \infty$: Flat distribution (not sensible in this context)

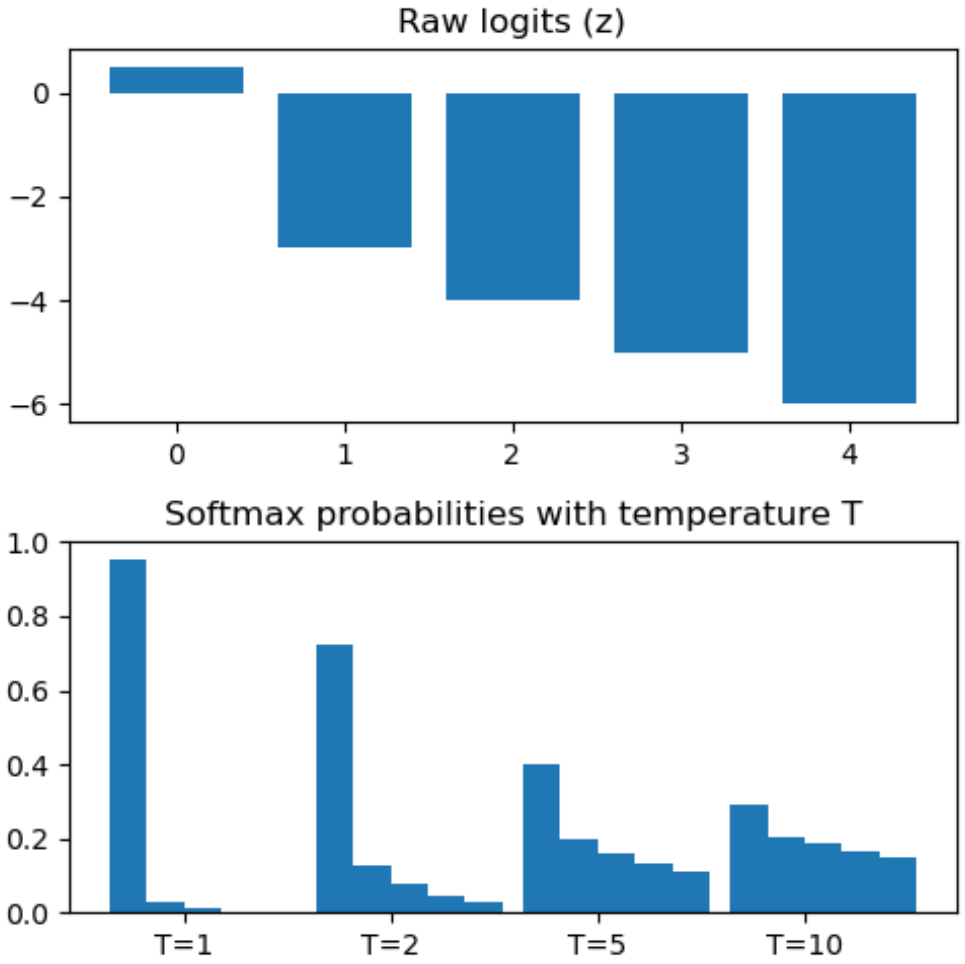
Effect of temperature scaling



- Values $T > 1$ lead to a the probabilities being more closely to each other.
- If we sample from these probability distributions, we get a higher entropy, and thus our models become more "creative" (but also less true to the original distribution).
- Rule of thumb: T is typically selected in the range [1.2 ... 2] for more creative outputs and [0.2 ... 0.8] for more deterministic outputs.

Hochschule Flensburg
University of Applied Sciences

Effect of temperature scaling in overconfident models



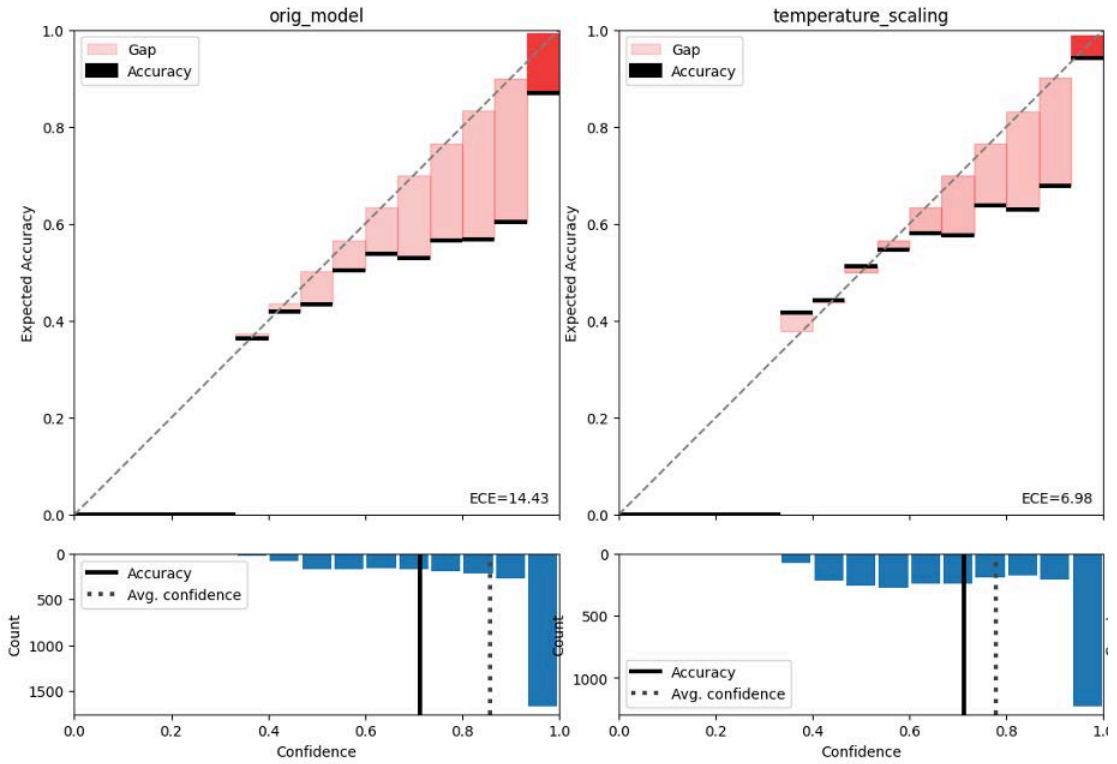
- Overconfident models typically come with a very high probability for a single class.
- There again, temperature acts as a leverage to reduce the confidence of the model and have a higher diversity in the generation process.
- Note that this only works if we use sampling from probability distributions, as the argmax is not changed by the temperature.

Hochschule Flensburg
University of Applied Sciences

Link to ADL: Temperature scaling as model calibration

Hochschule Flensburg
University of Applied Sciences

- We discussed temperature scaling also in advanced deep learning.
- There, the idea was very similar, but the target was another: to use the temperature to achieve a model that is more truthful to the actual probability distribution.
- This means: When the model says “80%,” it’s correct about 80% of the time.
- To do this, we use a post-hoc processing on a held-out validation set.
- Because softmax is monotonic in each logit, argmax decisions and accuracy don’t change; only the confidence changes.



Another diversity strategy: Top-k Sampling

Hochschule Flensburg
University of Applied Sciences

- Even with temperature scaling, the model can still produce
  - very unlikely tokens (if temperature is high), or
  - repetitive / deterministic outputs (if low).
- Top-k Sampling tries to balance diversity and coherence
  - Keep only the k most probable tokens from the model's distribution.
  - Set the probabilities of all other tokens to zero, then renormalize.
  - Sampling is done only among these top-k candidates.
- k acts as a hyperparameter:
  - Smaller k → more deterministic.
  - Larger k → more diversity, but potentially less coherence.

And another one: Nucleus Sampling

- Instead of fixing k, choose the smallest set of tokens whose cumulative probability ≥ p.
- Adaptive strategy, adapting to the output distribution of the model
- Sample from this “nucleus” of high-probability tokens.
- Intuition for images:
  - Top-k limits how many color values you consider.
  - Nucleus sampling adapts — if the model is uncertain (flat distribution), it allows more possible pixel values, leading to smoother textures.

Summary: Strategies to increase output diversity

Hochschule
Flensburg
University of
Applied Sciences

- Temperature Sampling:
  - Adjust logits before softmax:

$$p_i = \frac{e^{z_i/T}}{\sum_k e^{z_k/T}}$$

    T > 1 → more randomness / diversity.
- Top-k / Nucleus Sampling:
  - Restrict to top-k or top-p probabilities, then sample → balances coherence and diversity.
- Noise Injection:
  - Add Gaussian noise to hidden states or logits during sampling.
  - Encourages stochasticity in pixel dependencies.

Hochschule
Flensburg
University of
Applied Sciences

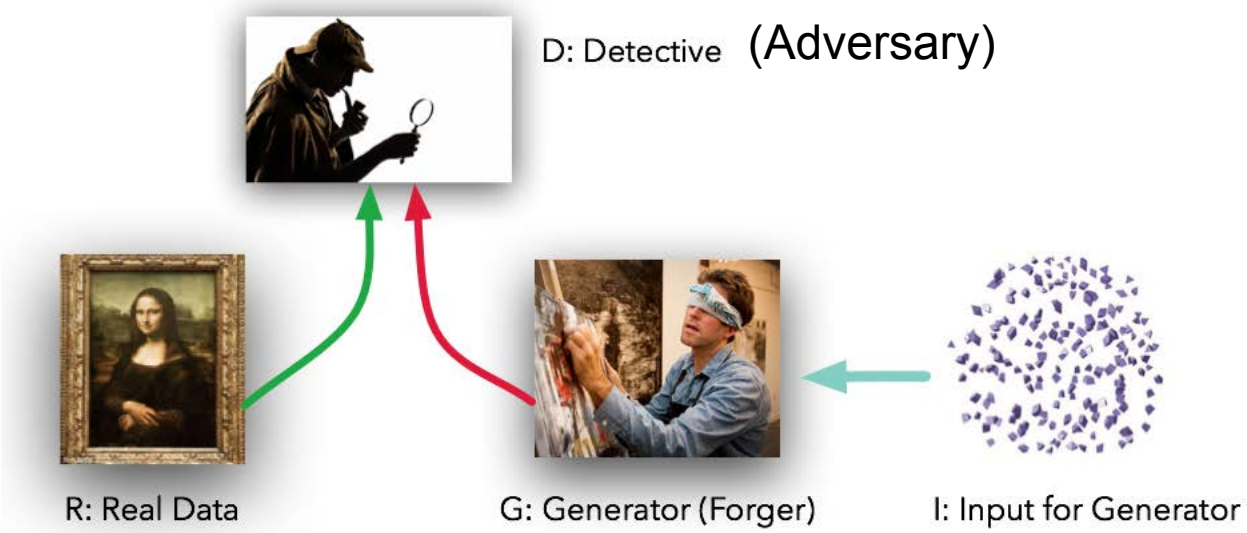
Adversarial Networks

Another approach to generation

- So far we have considered auto-regressive generation, i.e., we have an initial set and we continue with new data.
- The nice part about this is that we can use known loss functions (e.g., cross-entropy, L1)
- However, it would also be great to be able to do a one-shot prediction of the entire image.

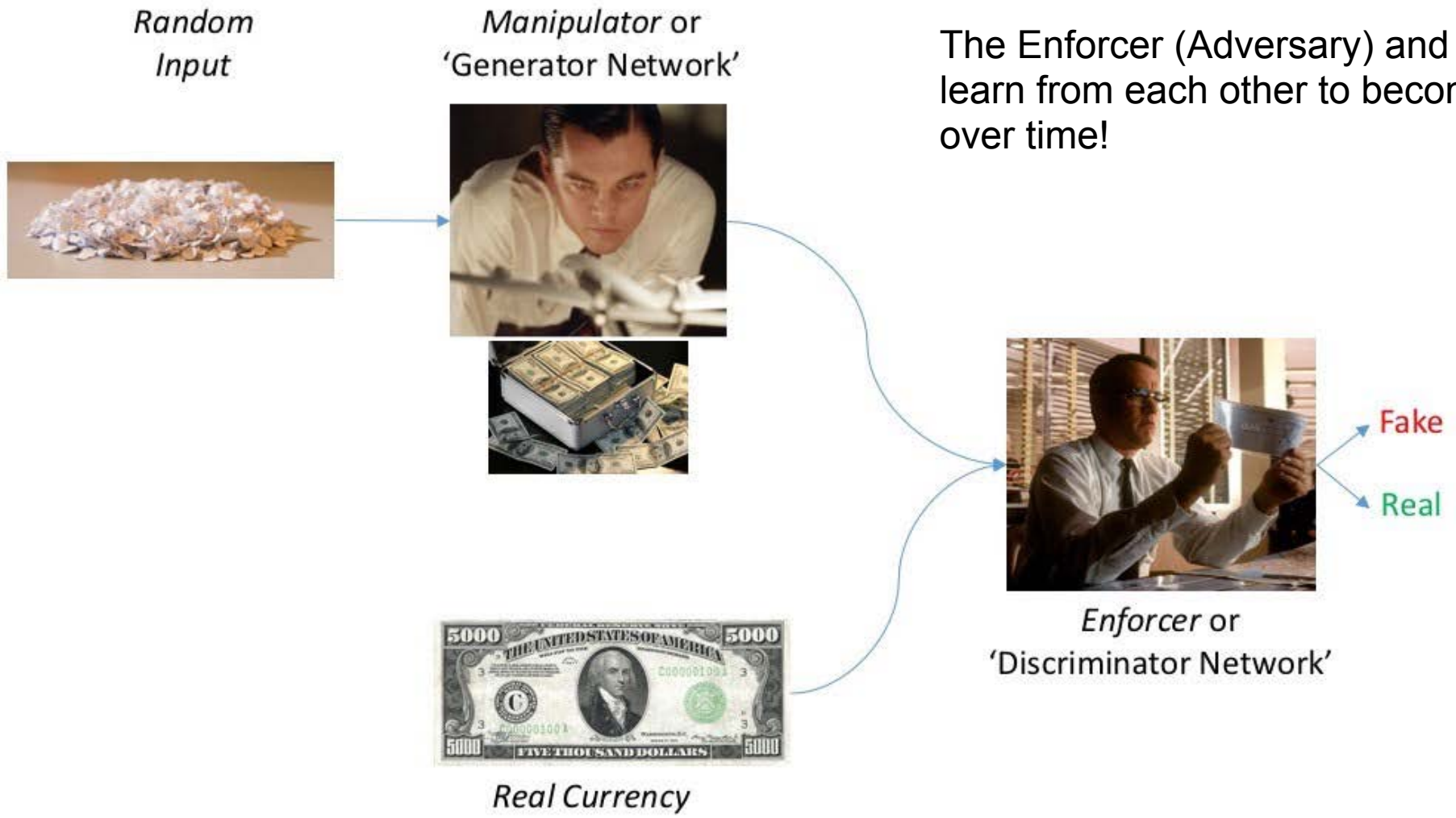
Hochschule Flensburg
University of Applied Sciences

We could play a game



- Instead of comparing the generated (forged) image to what we have in mind as photorealistic, we could also ask a detective to compare it with a set of real images.
- If the generator is really good, the detective can't discriminate the real (R) and forged (G) image any more.
- But then he himself can learn from that and improve, so the generator needs to become better.

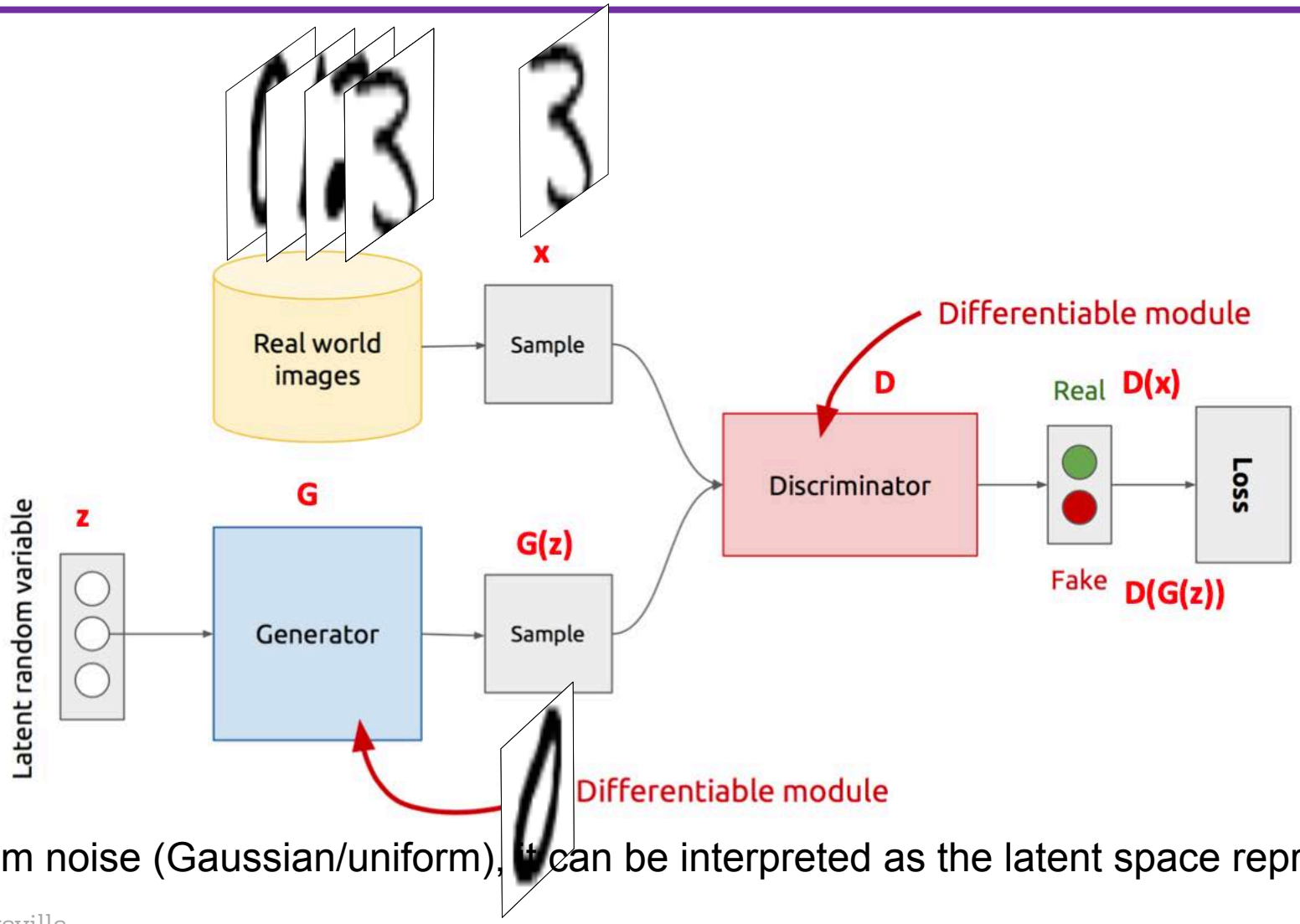
The forging game with currencies



The Enforcer (Adversary) and the Generator learn from each other to become better over time!

https://towardsdatascience.com/catch-me-if-you-can-a-simple-english-explanation-of-gans-or-dueling-neural-nets-319a273434db

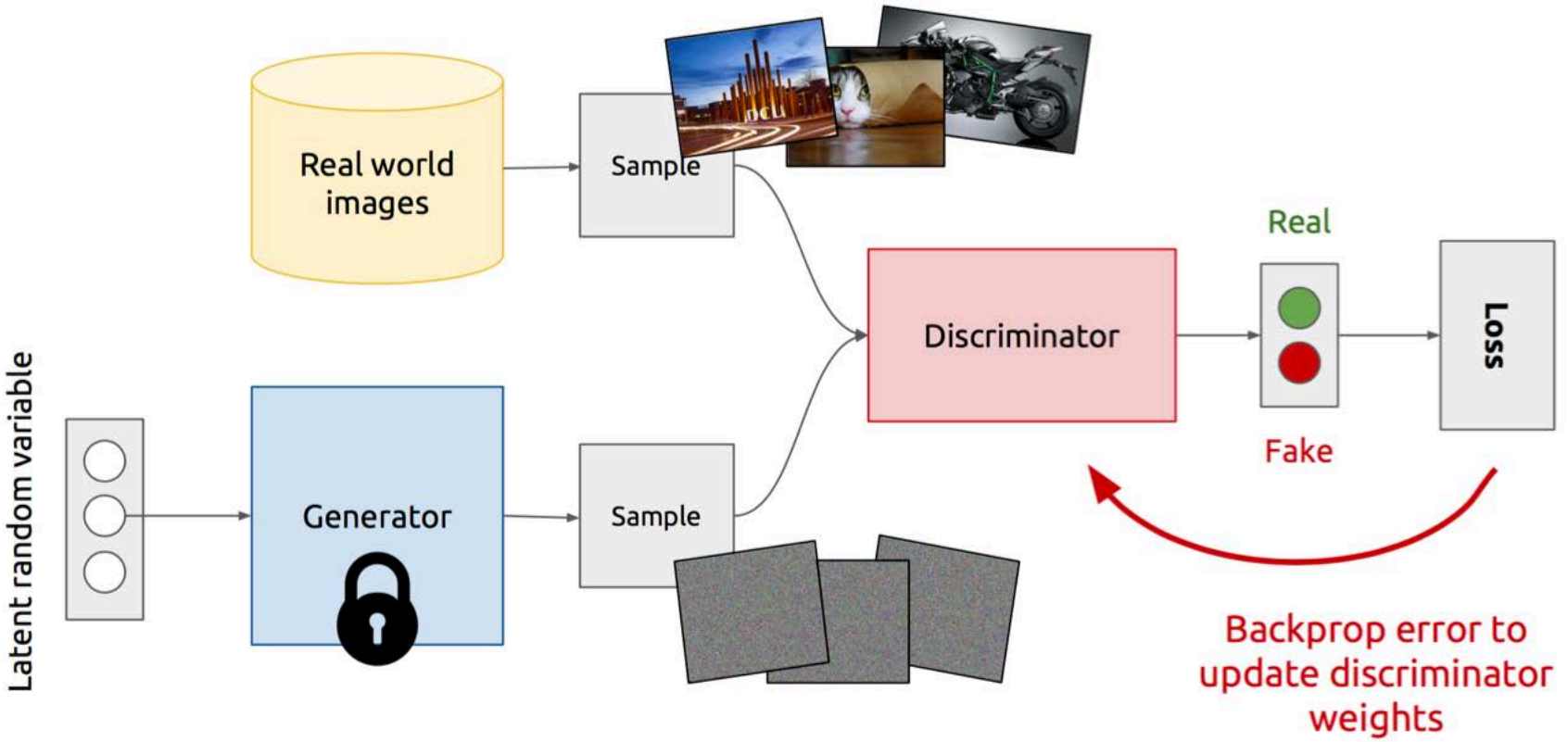
Generative Adversarial Networks



Z is a random noise (Gaussian/uniform), it can be interpreted as the latent space representation of an image.

https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

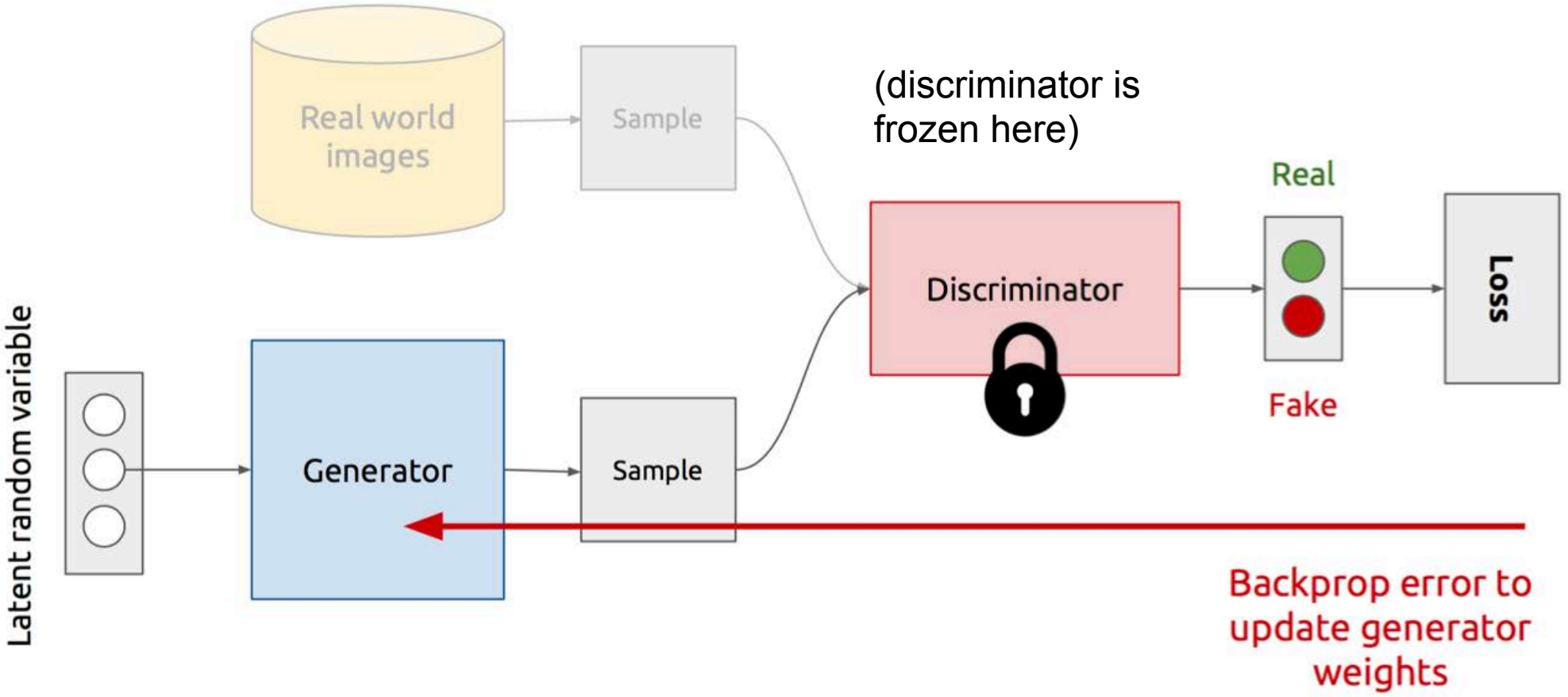
First step: Minimize the loss of the discriminator



Initially, the generator has no idea how to generate realistic images.

https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Step 2: Maximize the loss of the discriminator, training the generator



https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Generative Adversarial Networks

- GANs are formulated as a minimax game, where the
  - Discriminator is trying to maximize the reward $V(D, G)$ (or minimize the loss)
  - Generator is trying to minimize the reward of the discriminator $V(D, G)$

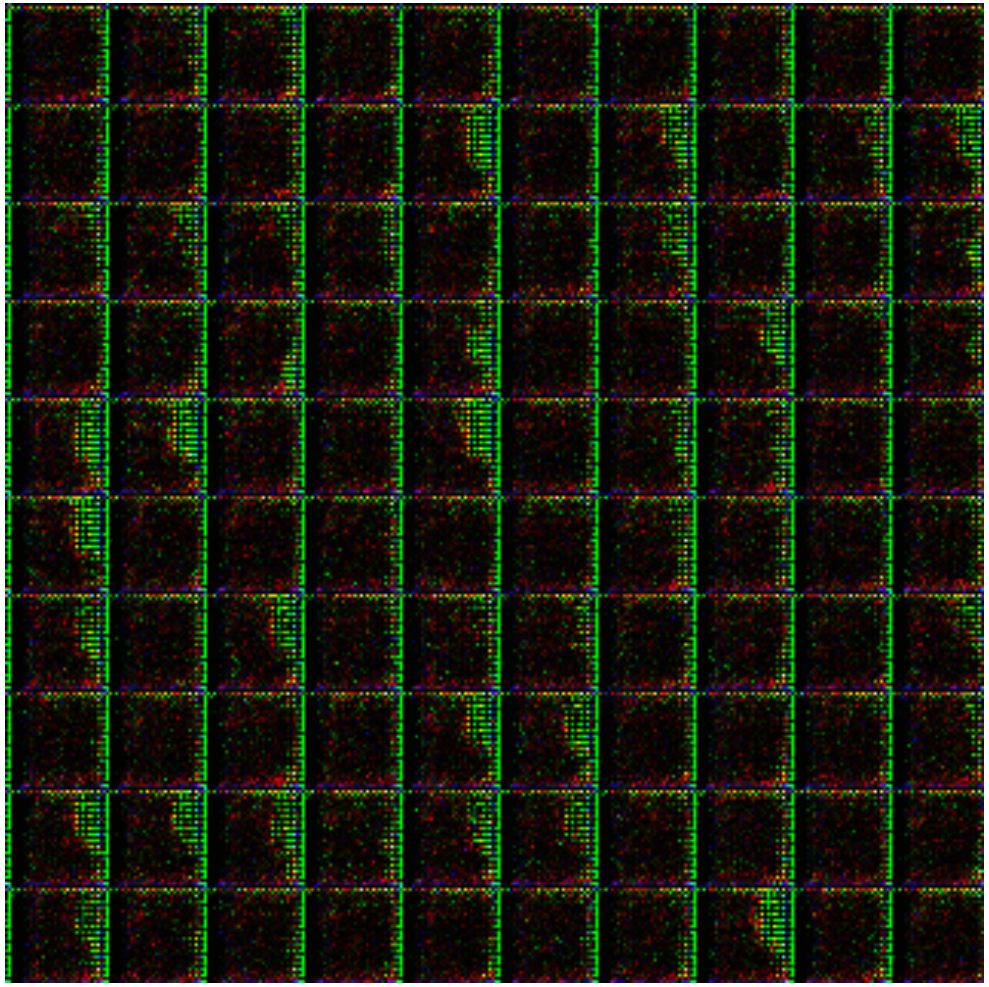
$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Expected value (log likelihood) of predicting the real images correctly

Expected value (log likelihood) of predicting the generated images correctly

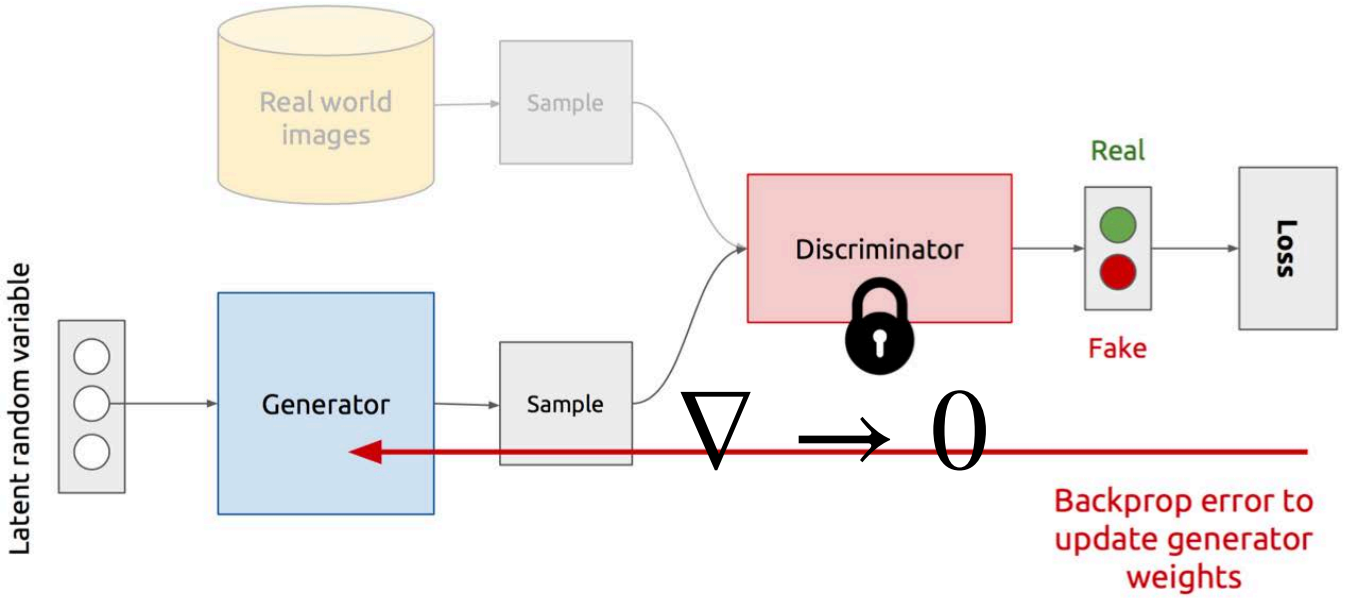
  - The Nash equilibrium is achieved, if neither the generator nor the discriminator can improve to become better.

Let's see how the generator learns to create realistic images



Video here: https://openai.com/content/images/2017/02/gen_models_anim_2.gif

Vanishing Gradient strikes back!



- If the discriminator becomes too good at discriminating fake from real images, the loss (and thus the gradient) goes towards 0.
- This means that the generator can also no longer improve, since it learns from the gradients of the discriminator.

GANs: Advantages and Problems

- GANs are great because:
  - They are less prone to overfitting (as they never see the training data)
  - They are good at capturing the overall distribution of the data set.

- GANs are horrible, because:
  They are known to be subject to a number of issues that makes them hard to train
  - Non-Convergence (e.g., due to vanishing gradients)
  - Mode Collapse

Non-Convergence

Hochschule Flensburg
University of Applied Sciences

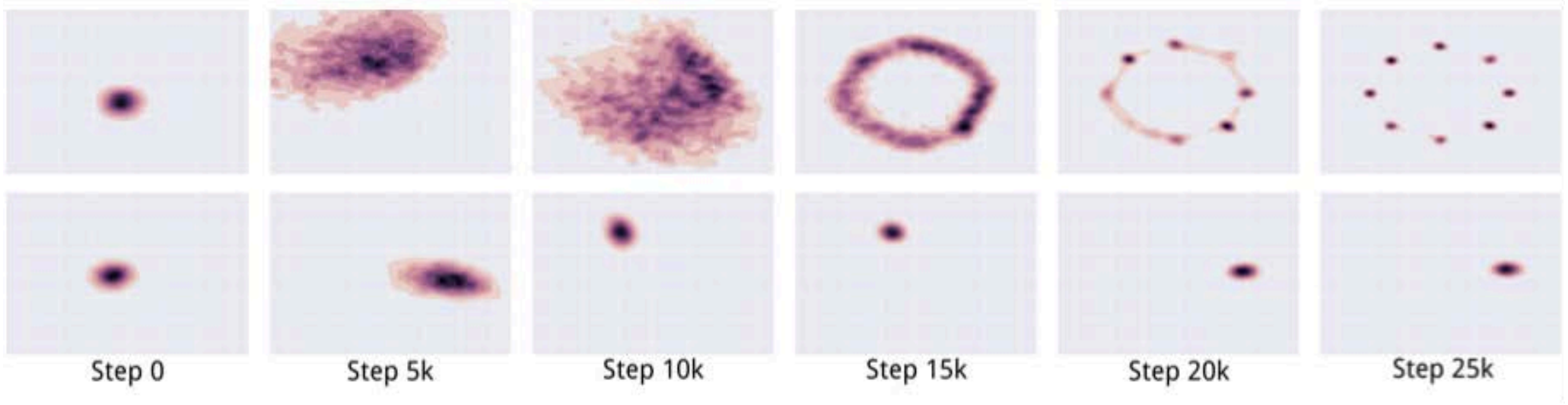
- Deep Learning models (in general) involve a single player
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - Problem: With non-convexity, we might converge to local optima.
- GANs instead involve two (or more) players
  - Discriminator is trying to maximize its reward.
  - Generator is trying to minimize Discriminator's reward.
  - SGD was not designed to find the Nash equilibrium of a game.
  - Problem: We might not converge to the Nash equilibrium at all.

https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Mode Collapse

- There is no incentive for the GAN to cover the full distribution of the training set (sample diversity).
- It can thus happen, that it rotates through the modes of the data distribution.

Target distribution

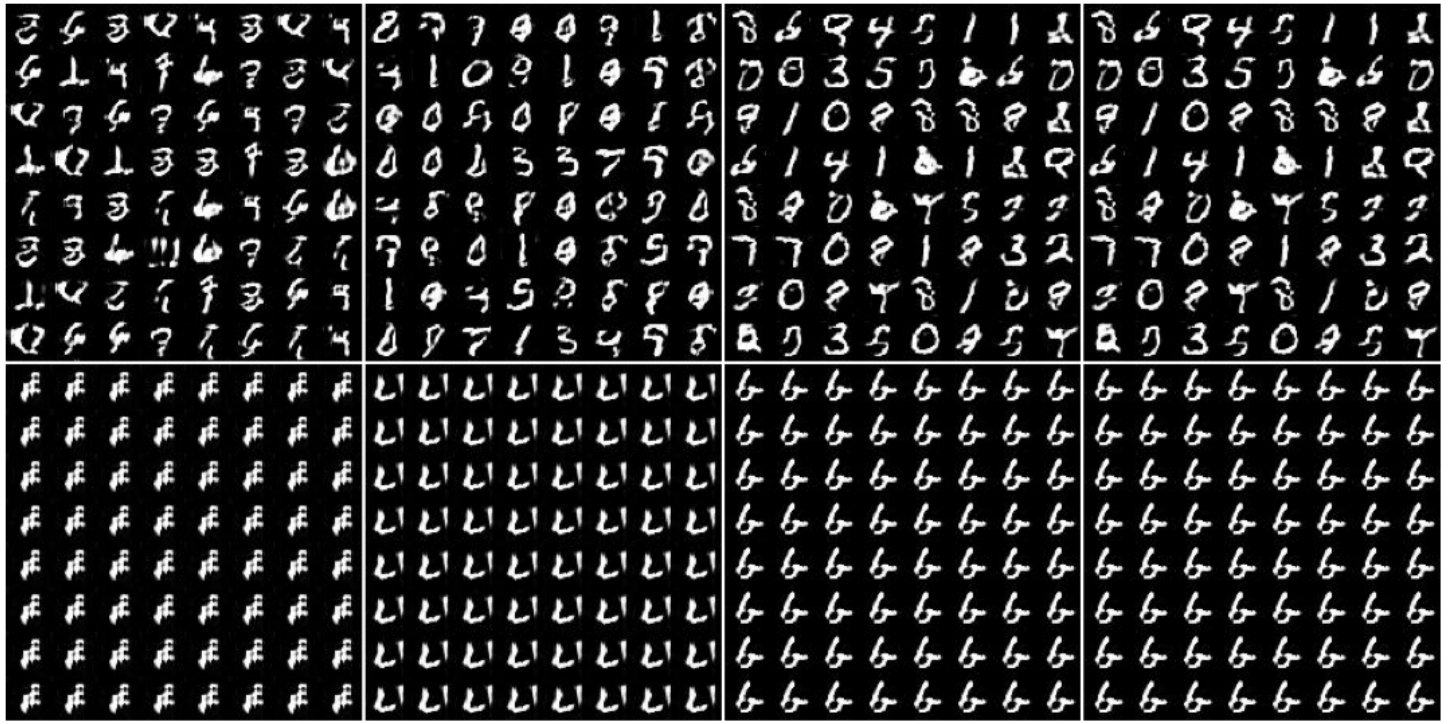


Converged training

Mode collapse

Metz, Luke, et al. "Unrolled Generative Adversarial Networks." arXiv preprint arXiv:1611.02163 (2016).

Mode Collapse (Examples on MNIST)



- In the lower training run, the model is not able to represent the full distribution but flips through different modes of the set.

Measures against Mode Collapse: Mini-Batch GANs (Salimans et al., NeurIPS, 2016)

(heuristical solution)

Hochschule Flensburg
University of Applied Sciences

- At mode collapse the generator produces good images, but shows no diversity.
- Thus, the discriminator can't tag them as fake, and the generator can't learn to improve.

- To address this problem, we can have the discriminator look at an entire mini-batch of images.
- If there is a lack of diversity, the discriminator will know it's fake
- And thus the generator can learn to produce more diverse samples.

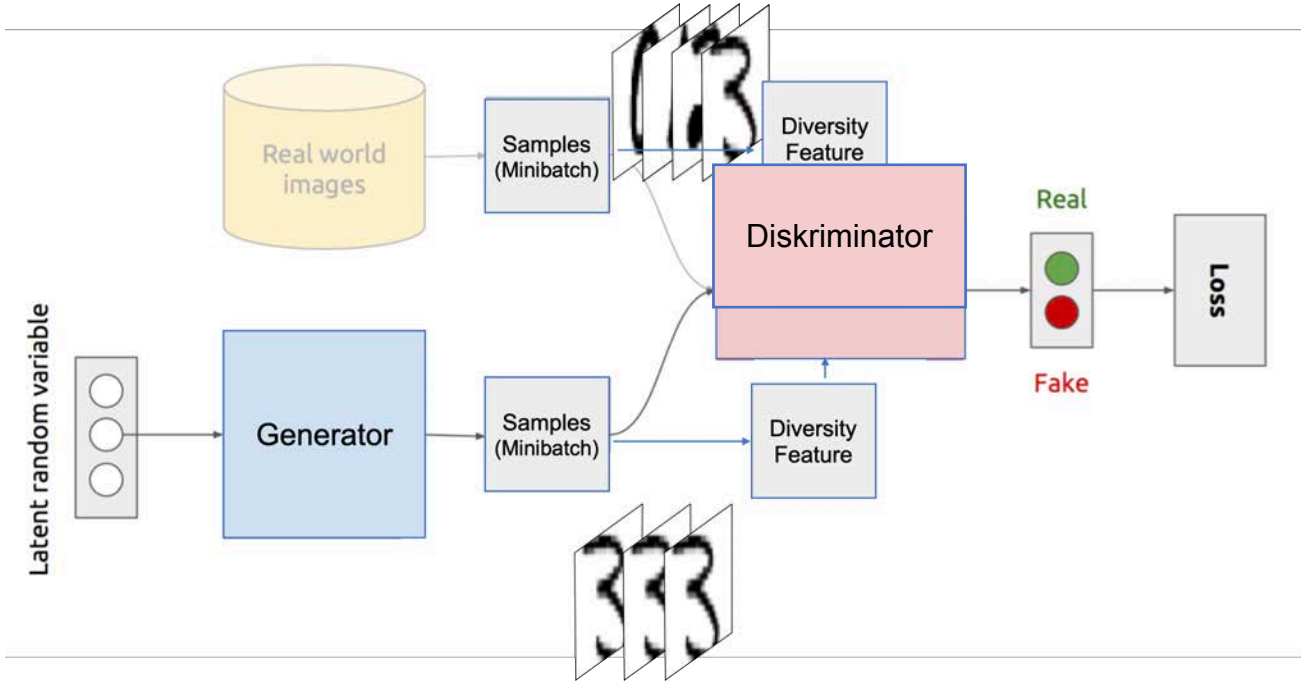
https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Salimans, Tim, et al. "Improved techniques for training gans." Advances in Neural Information Processing Systems. 2016.

Minibatch-GANs (Salimans et al., NeurIPS 2016)

Hochschule Flensburg
University of Applied Sciences

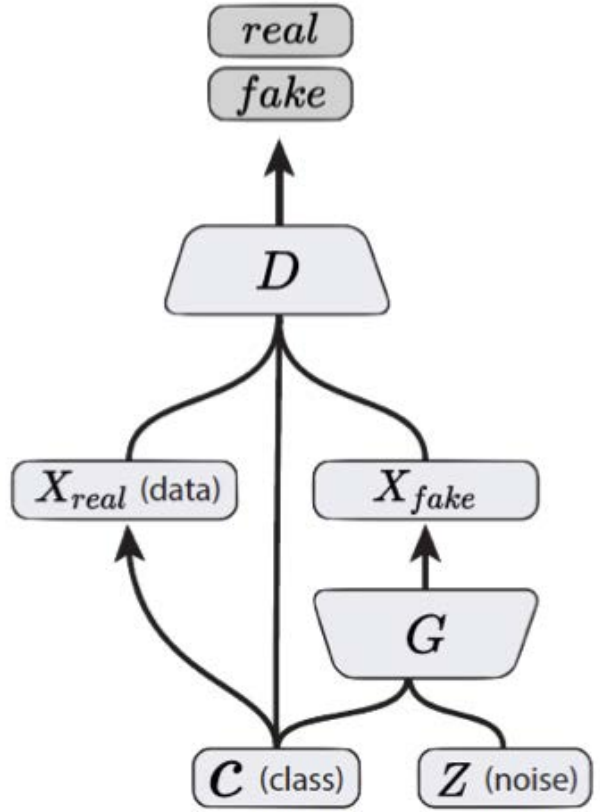
- Strategy:
  - Extract features that are discriminative of diversity in the mini-batch (e.g., L2 norm of the difference of all images to their mean)
  - Feed those features to the discriminator alongside the image
  - As those feature values differ between the real-world images and the generated images, the discriminator will use them
  - In turn, the generator will learn to generate a more diverse minibatch



https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

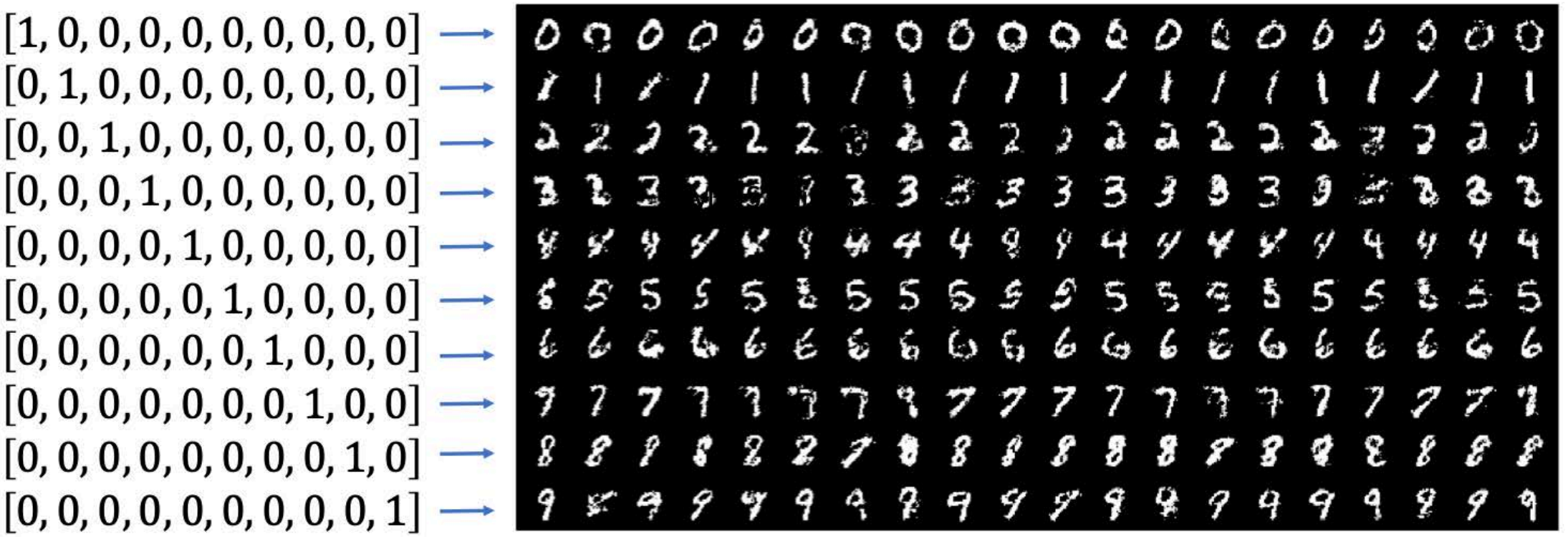
Conditional GANs

- Another architecture that is not prone to mode collapse are conditional GANs.
- They condition the network on additional information to improve multi-modal learning.
- It is, however, required to have a labelled dataset for this, as the condition controls the selection of samples.



https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Conditional GANs: MNIST



Mirza, Mehdi, and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014).

Summary

- Adversarial training is a method to mitigate the problem of the unknown suitable loss function.
- Adversarial training can be utilized to
  - Generate (realistic) images or other data
  - Perform adaptation in feature space
- Adversarial training is, however, also:
  - Sometimes not converging, or showing mode loss
  - This can be mitigated using specialized methods (minibatch, conditional GANs, etc..)

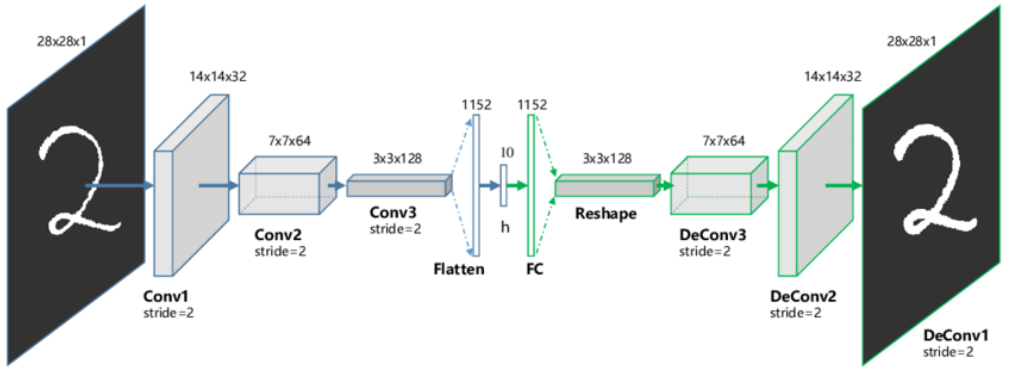
Hochschule Flensburg
University of Applied Sciences

Hochschule
Flensburg
University of
Applied Sciences

Variational Autoencoders

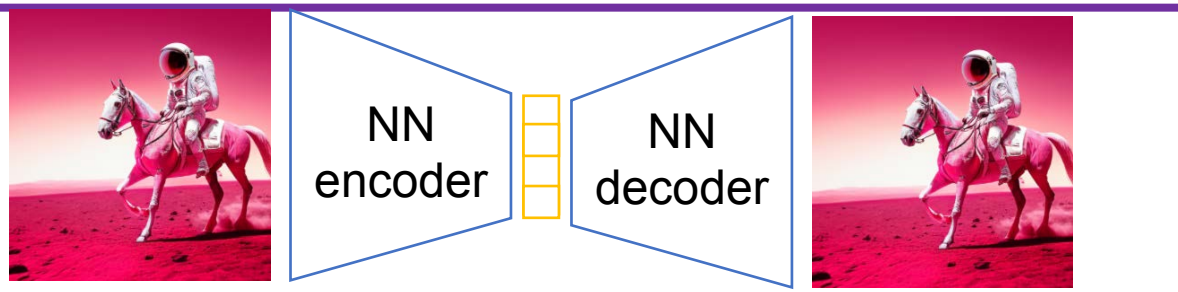
Autoencoders

- In computer vision, autoencoders are models that will encode and decode an image to retrieve the same image.
- Due to the bottleneck of the network, the feature vector there needs to be sufficiently discriminative to reconstruct the network.
- There are many applications for autoencoders, including:
  - Denoising (since noise is commonly uncorrelated to the (semantic) information of the image)
  - Projection / Clustering of Images
  - Image compression
  - Building unsupervised encoders to be used in model finetuning / transfer learning
  - Superresolution (increasing the resolution of the input image)



Could this also be used for image generation?

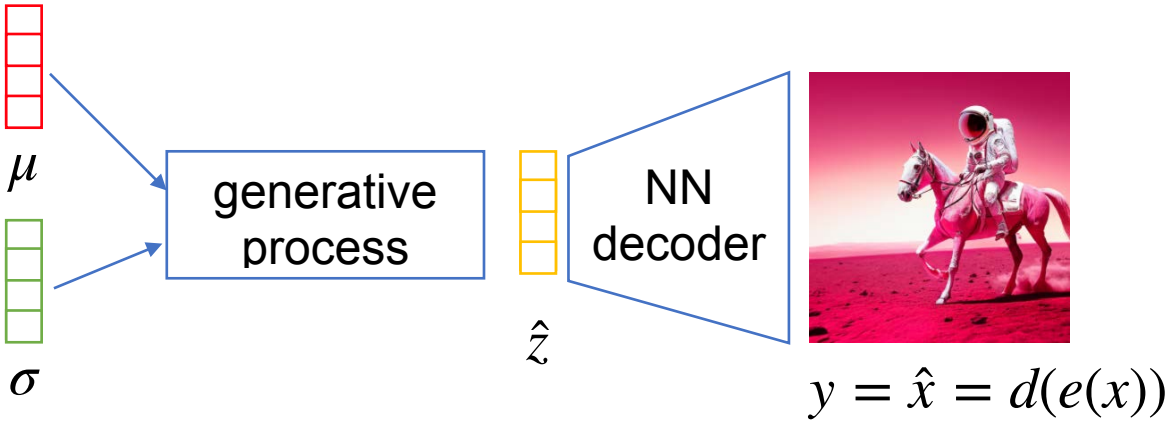
Using Autoencoders for generation



x $z = e(x)$ $y = \hat{x} = d(e(x))$

- We can split up the autoencoder into an encoder and a decoder.
- In the general case of $x \neq d(e(x))$, the autoencoder executes a lossy compression of the image.
- Could we also manipulate the latent space of the encoder to generate other images?

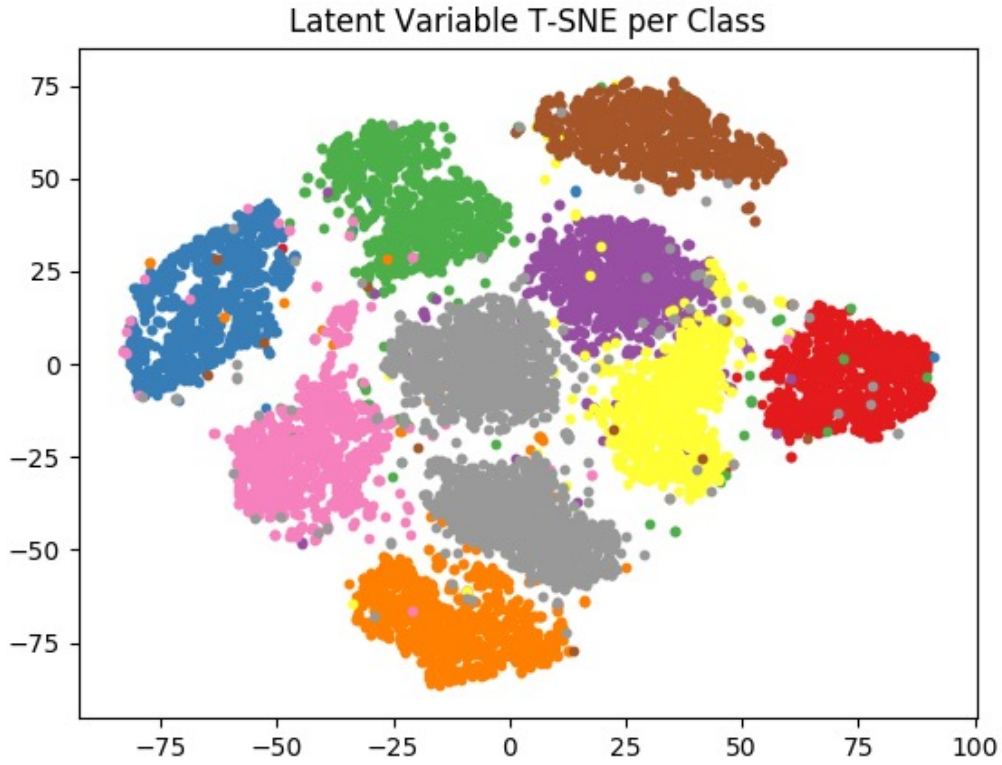
Using Autoencoders for image generation



- We could use a generative process to sample $\hat{z}$ from a distribution such as $\mathcal{N}(\mu, \sigma)$:
  $\hat{z} \sim \mathcal{N}(\mu, \sigma)$
- Then we would hope that the decoder is capable of generating realistic images from this latent space vector.

How do we want our latent space to be organized?

- Interpretability: if the derived features are semantically meaningful, and interpretable by a human, they can be easily evaluated. (e.g. noisy-OR: "features" are diseases a patient has)
- Downstream usability: the features are "useful" for downstream tasks. Some examples:
  - the more the latent spaces represents class disentanglement, the easier it is to train a classifier on the latent space
  - if, for example, a linear classifier can be trained from the latent space, this is a sign of high downstream usability for the task
  - the more discriminatory the features are, the smaller the number of samples needed

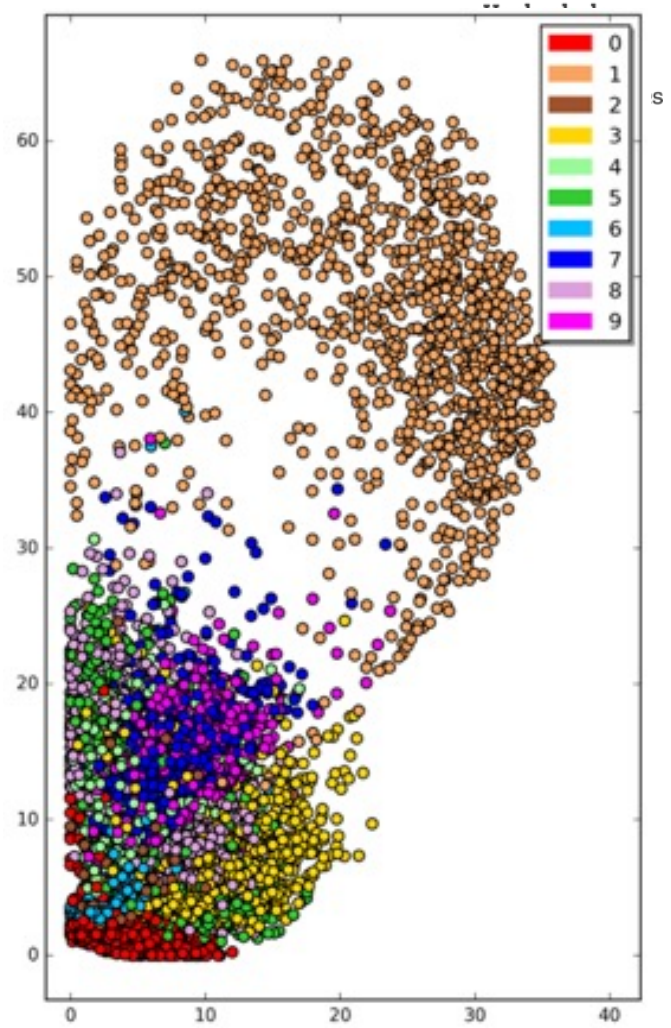


t-SNE projection of VAE-learned features of the 10 MNIST classes. Image from https://pyro.ai/examples/vae.html

https://web.cs.hacettepe.edu.tr/~erkut/cmp784.f21/slides/lec11-variational_autoencoders.pdf

The latent space of an autoencoder

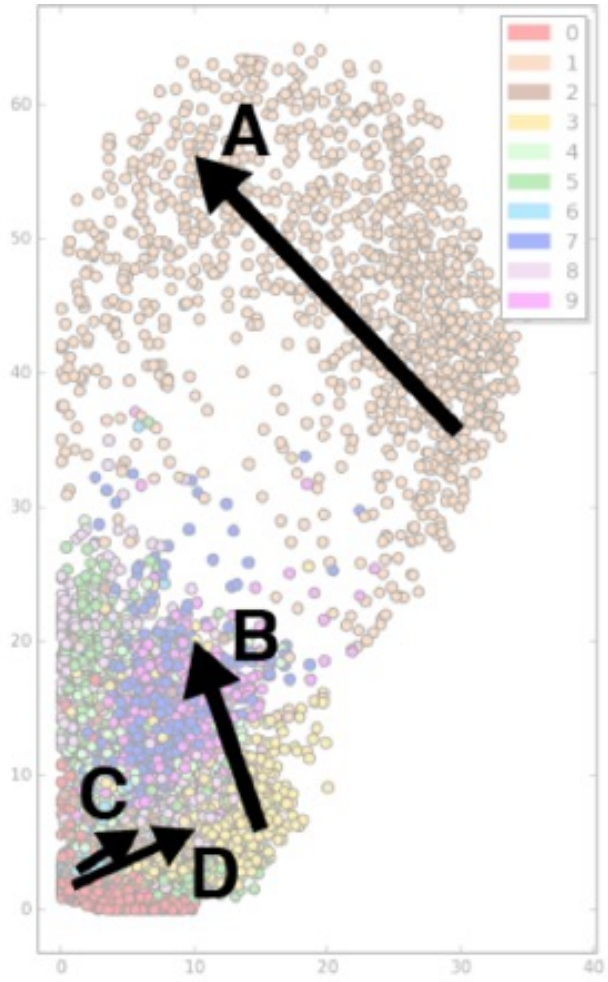
- Let's examine the latent space of a simple autoencoder trained on MNIST.
- On the right you can see the first principal components of the feature representation of each number.
- We would expect semantic clustering, i.e. similar objects appearing at similar locations.



A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

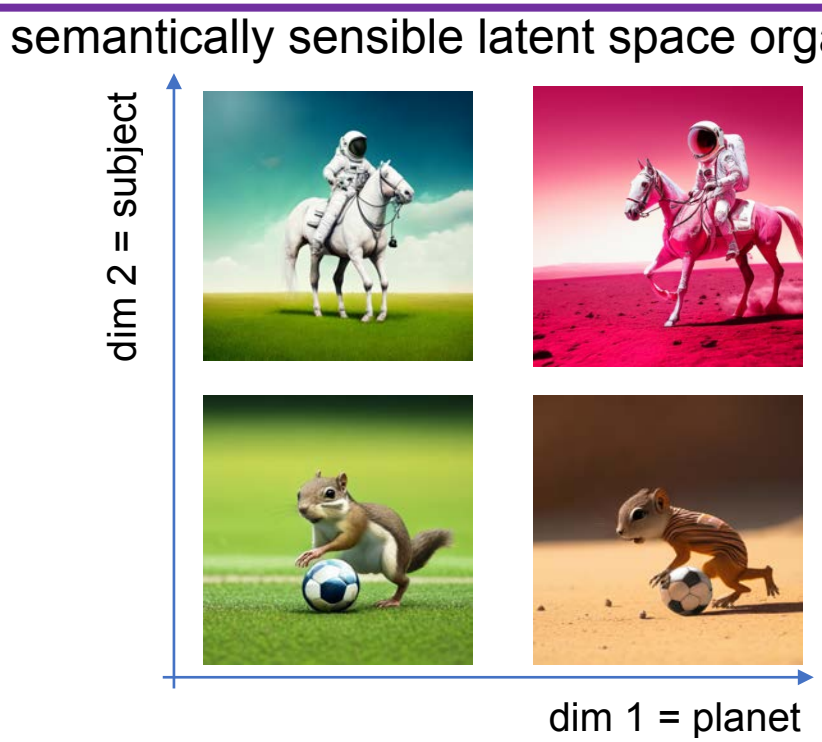
Traversing in the latent space

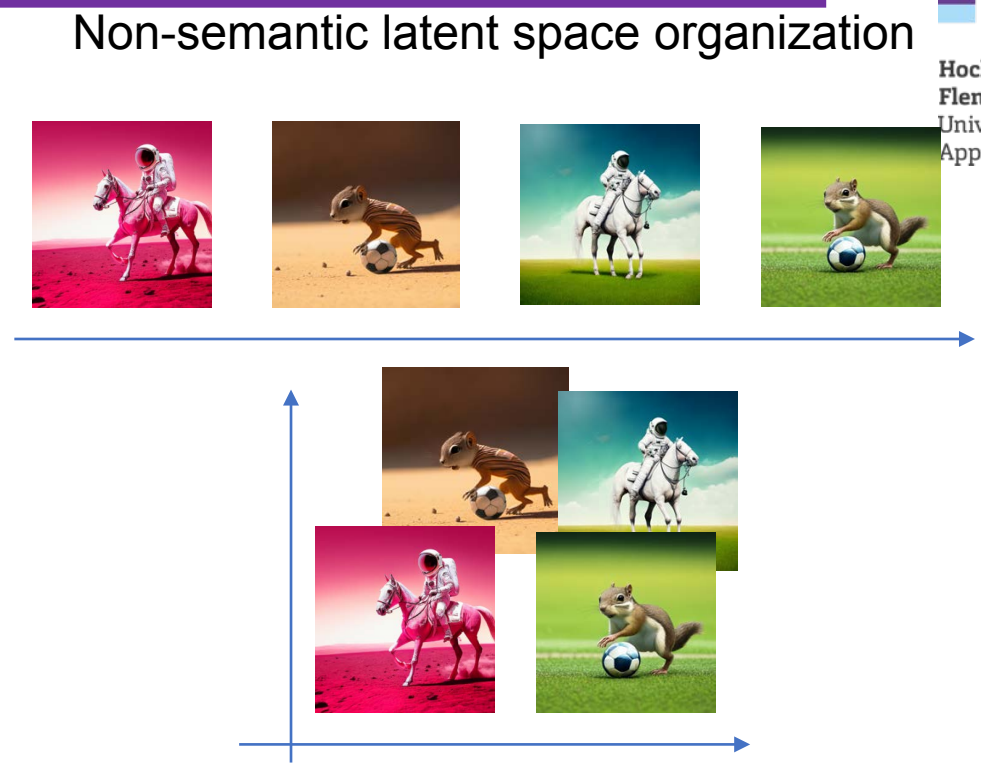
- Let's now move in latent space.
- We always start at the starting point of the arrow and then traverse through the latent space following the tail of the arrow.
- After reconstruction, this is what we observe:



https://web.cs.hacettepe.edu.tr/~erkut/cmp784.f21/slides/lec11-variational_autoencoders.pdf

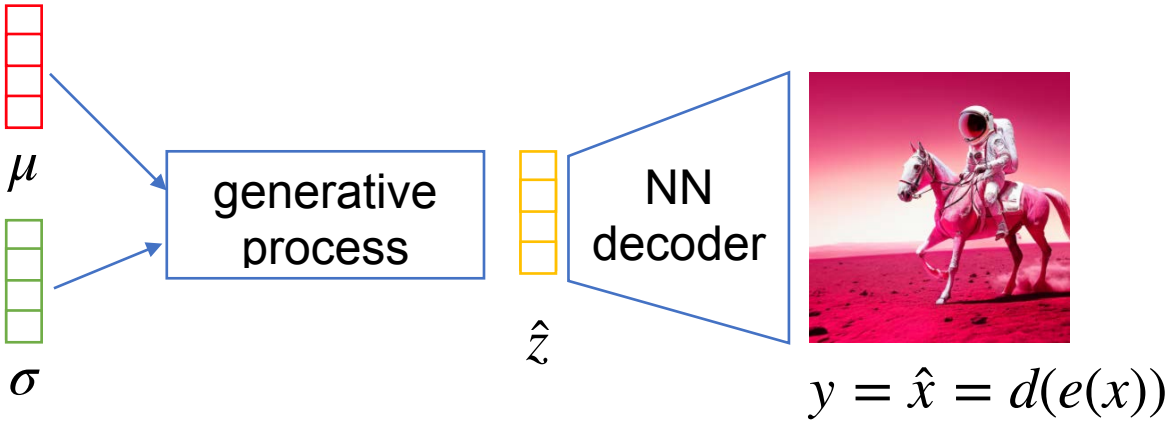
Using Autoencoders for image generation





- We have no guarantee of how the latent space is organized, since the loss function is only about the reconstruction.
- If we have a decoder of unrestricted power, it could just memorize every image of the training set and reconstruct it using an index (left side of upper drawing).
- This could be seen as case of overfitting.

Using Autoencoders for generation



- The latent space has an arbitrary organization.
- Hence, it is impossible to ensure that the autoencoder contains a decoder that is capable of decoding arbitrary latent vectors and thus we can't really use the latent space of an autoencoder for image generation.
- We should thus regularize the latent space to allow for this.

Intuition: Regularized latent space

Hochschule Flensburg
University of Applied Sciences

- If the latent space is regularized, we expect:
  - continuity, i.e., two close points in the latent space should not give two completely different contents once decoded
  - completeness, i.e., for a chosen distribution, a point sampled from the latent space should give a sensible content once decoded.

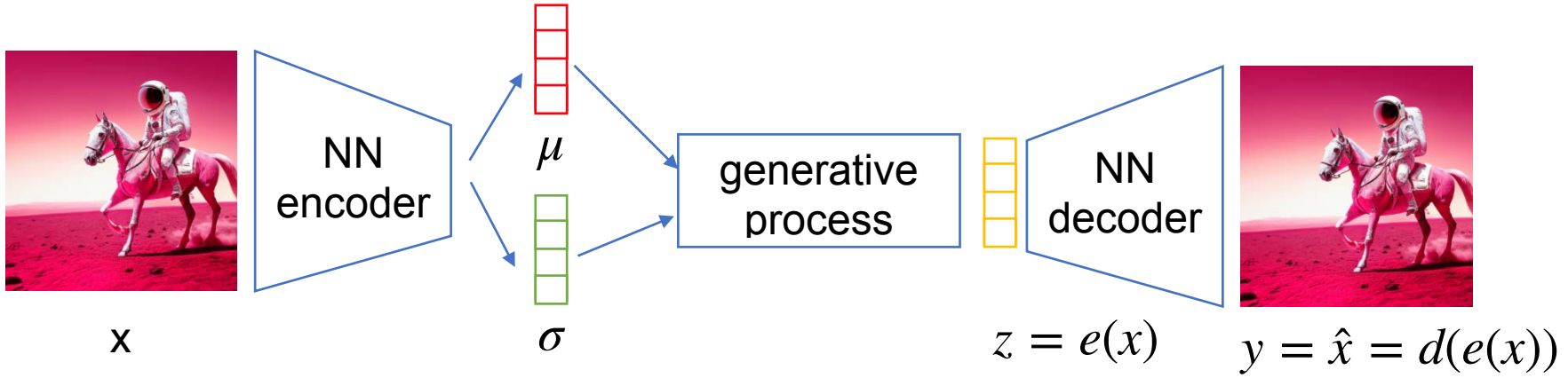







- Hence, latent space vectors with low distance should also lead to similar images when decoded.
- However, autoencoder latent spaces contain gaps and do not provide separability of classes.

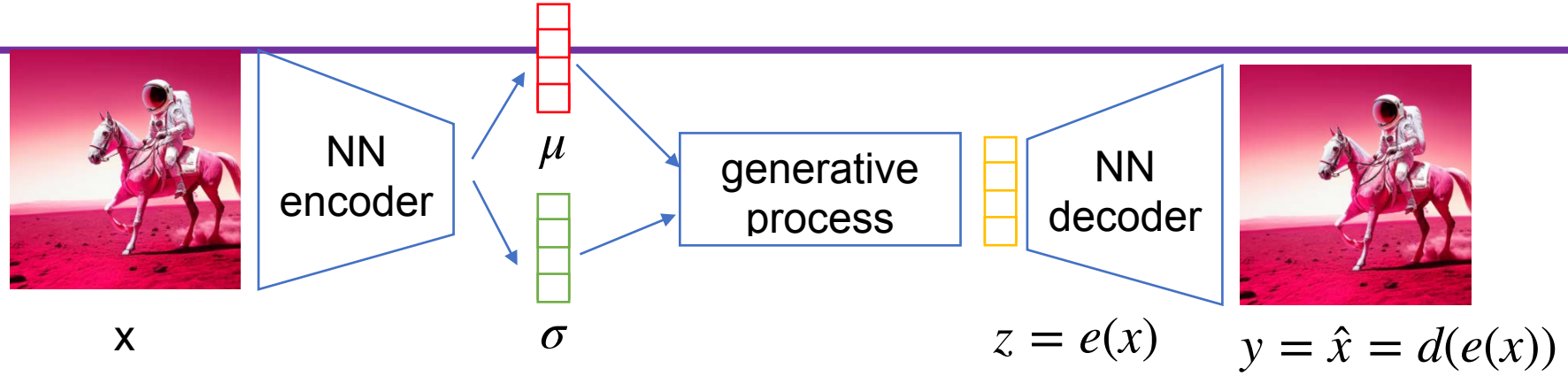
Adding variational constraints to autoencoders



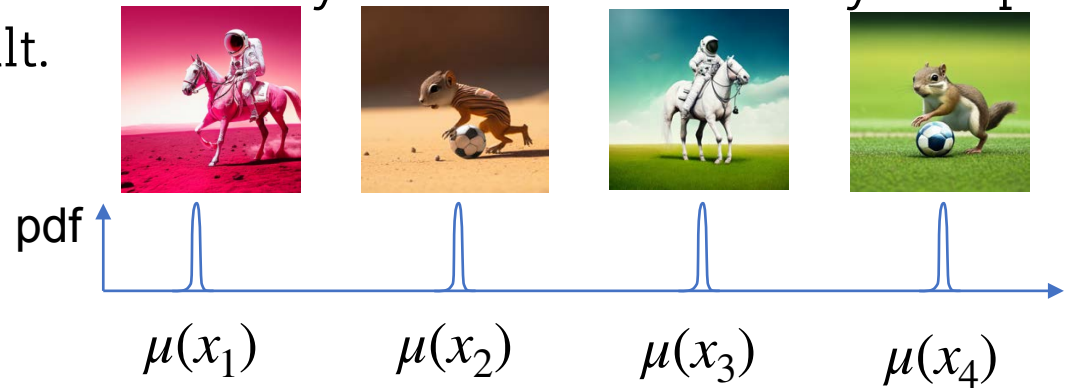
- Instead of encoding an image directly, in variational autoencoders, we use the encoder to predict the parameters of a normally distributed variable (i.e., the mean and covariance matrix) of the respective input image in latent space.
- We then consider this distribution a random process and sample from it.
- The decoder has the task to decode from the sampled latent representation.
- Hence, it is forced to learn to generate images from normal distributions.

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Doesn't this enable also another overfitted solution?



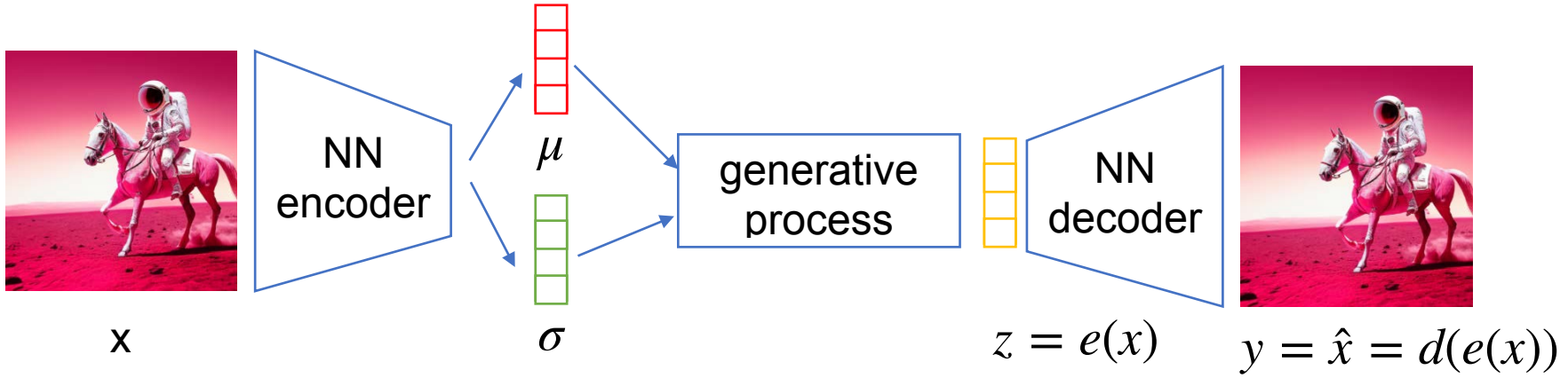
- Theoretically, now the encoder could learn to predict low variance, then nothing would change.
- Or, it could alternatively move the means very far apart in latent space, yielding the same result.



Then, we would not achieve continuity! :-(

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders: Adding a second loss term

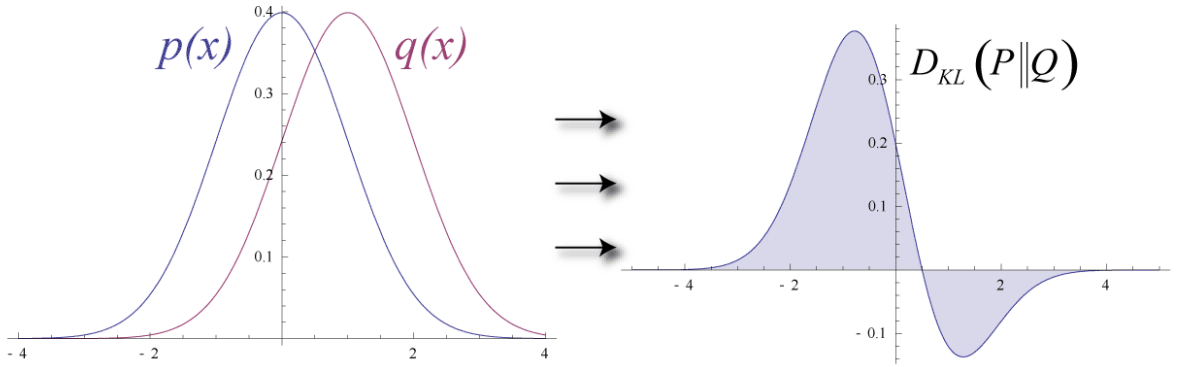


- Because of this, in variational autoencoders (VAEs) we add a secondary loss term in latent space that ensures this does not happen.
- The idea is that we want the distributions to be normal, i.e. $\hat{z} \sim \mathcal{N}(0,1)$.
- This way, the decoder must learn to decode from a noisy latent variable, and the encoder must learn to pass through the information without using the trivial solution (indexing).
- We can enforce this using a Kullback-Leibler-Divergence (KL divergence)

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Kullback-Leibler-Divergence

- A measure of how one probability distribution $p(x)$ diverges from a second $q(x)$ probability distribution.

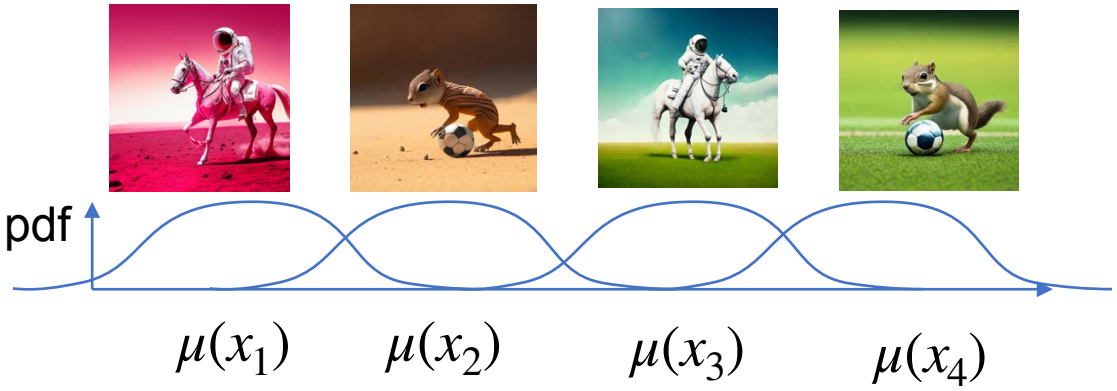


$$D_{KL}(p\|q) = -\mathbb{E}_{x\sim p}\log\frac{q(x)}{p(x)}$$

- The KL Divergence is always positive and 0 in case p(x) and q(x) are the same distribution.
- In VAEs, it is used to quantify the difference between the encoded distribution (latent space) and a predefined prior distribution.

Constraining the distribution

- A normal distribution, independent of the input, would not contain any information, so we can't go there
- We need to find a solution that is a good compromise between yielding information, and having close mean values and high standard deviations.

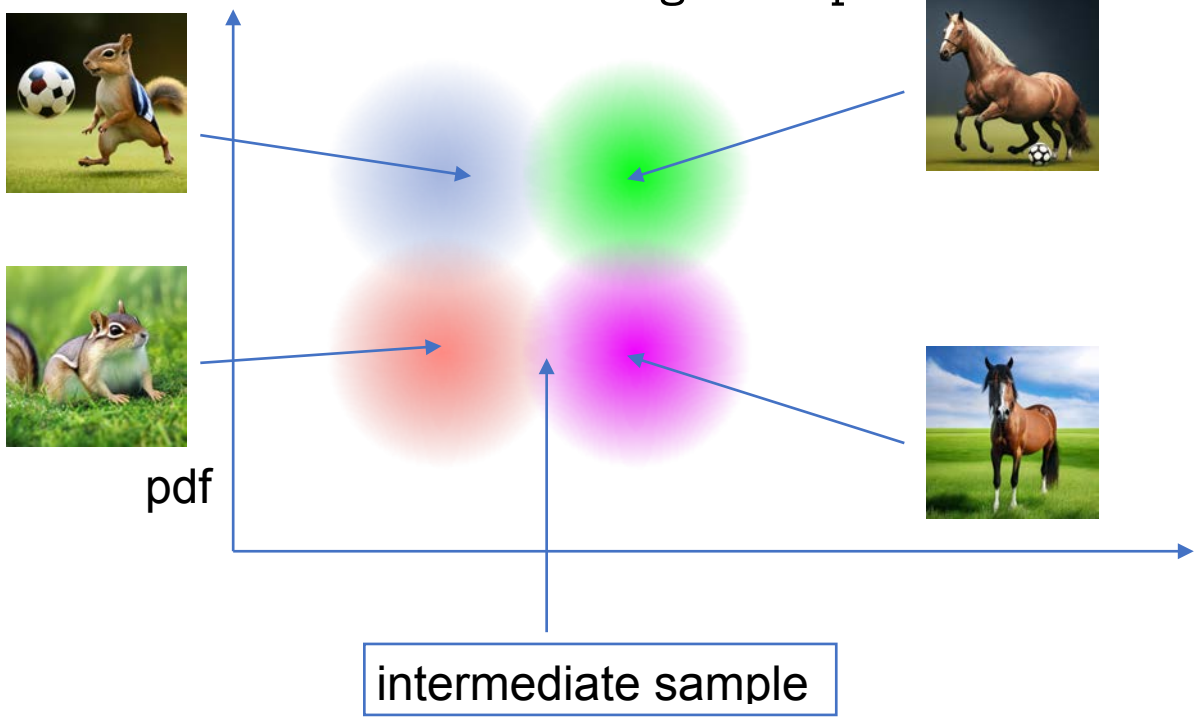


We can see that this will encourage latent space continuity

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Constraining the distribution

- How does this also encourage completeness?



- In fact, yes:
  - The model now needs to push the gaussians close to each other and maximize the variance, but at the same time allow for discrimination.
  - Just imagine that during the sampling, intermediate latent vectors will be randomly chosen.
  - The reconstruction loss is much lower, if similar latent space vectors also represent similar images.

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Example: Constraining the distributions / VAEs in action

sample $\mathbf{x}_1$

NN encoder

μ_1

σ_1

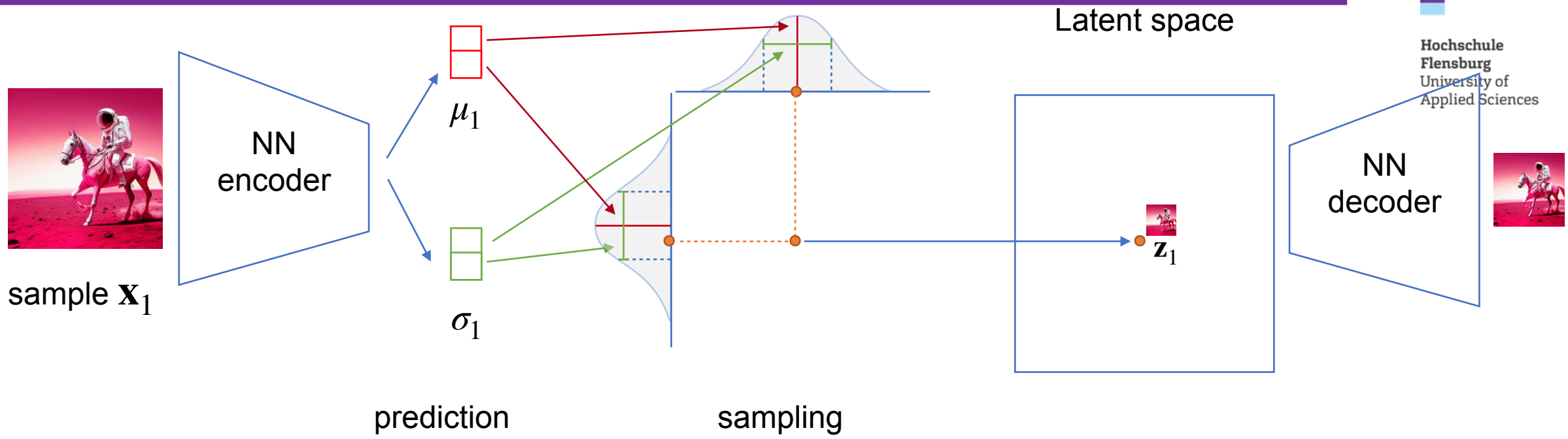
prediction

…

NN decoder

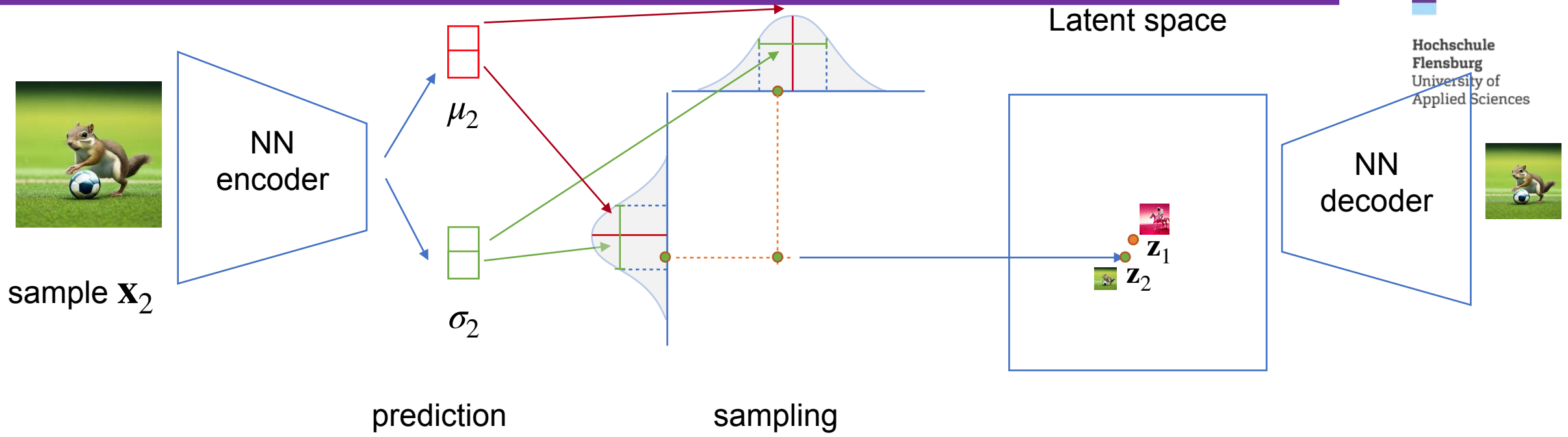
- The first sample is fed into the encoder, producing $[\mu_1, \sigma_1] = e(\mathbf{x}_1)$

Example: Constraining the distributions / VAEs in action



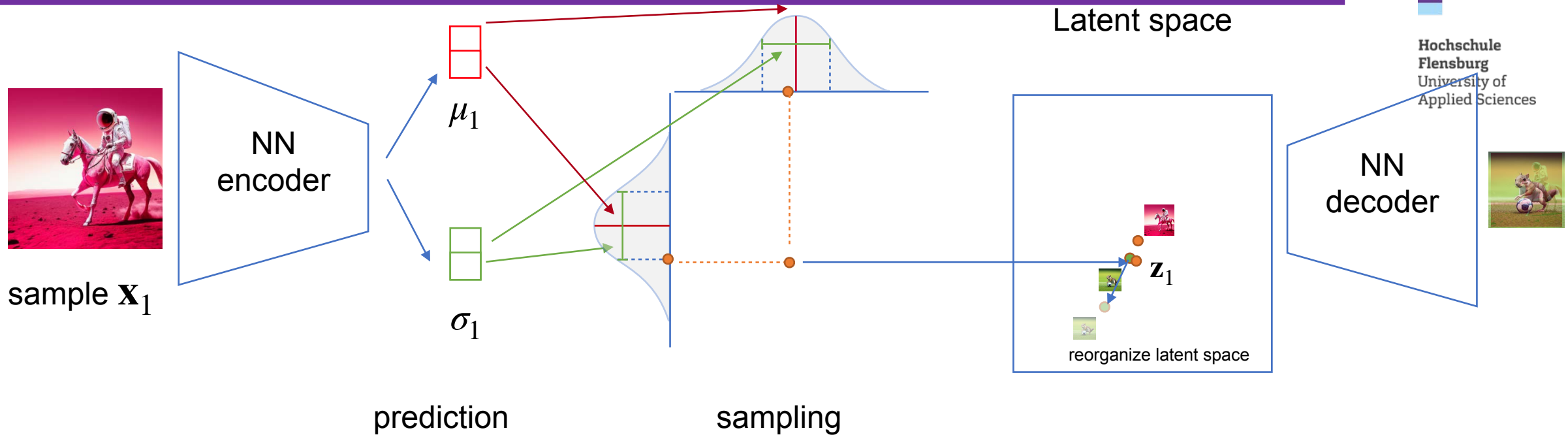
- The first sample is fed into the encoder, producing $[\mu_1, \sigma_1] = e(\mathbf{x}_1)$
- The sampling is performed using the mean and standard deviation predicted, yielding a sample $\mathbf{z}_1 \sim \mathcal{N}(\mu_1, \sigma_1)$
- Decode the latent vector to reproduce the original image

Example: Constraining the distributions / VAEs in action



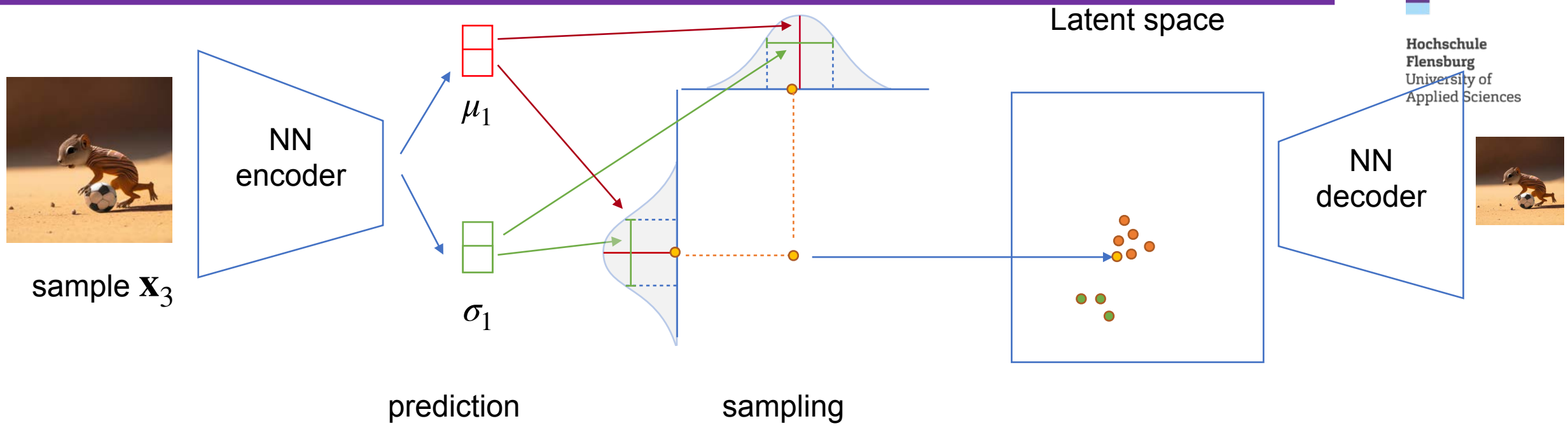
- The second sample is fed into the encoder, producing $[\mu_2, \sigma_2] = e(\mathbf{x}_2)$
- The sampling is performed using the mean and standard deviation predicted, yielding a sample $\mathbf{z}_2 \sim \mathcal{N}(\mu_2, \sigma_2)$
- Decode the latent vector to reproduce the original image

Example: Constraining the distributions / VAEs in action



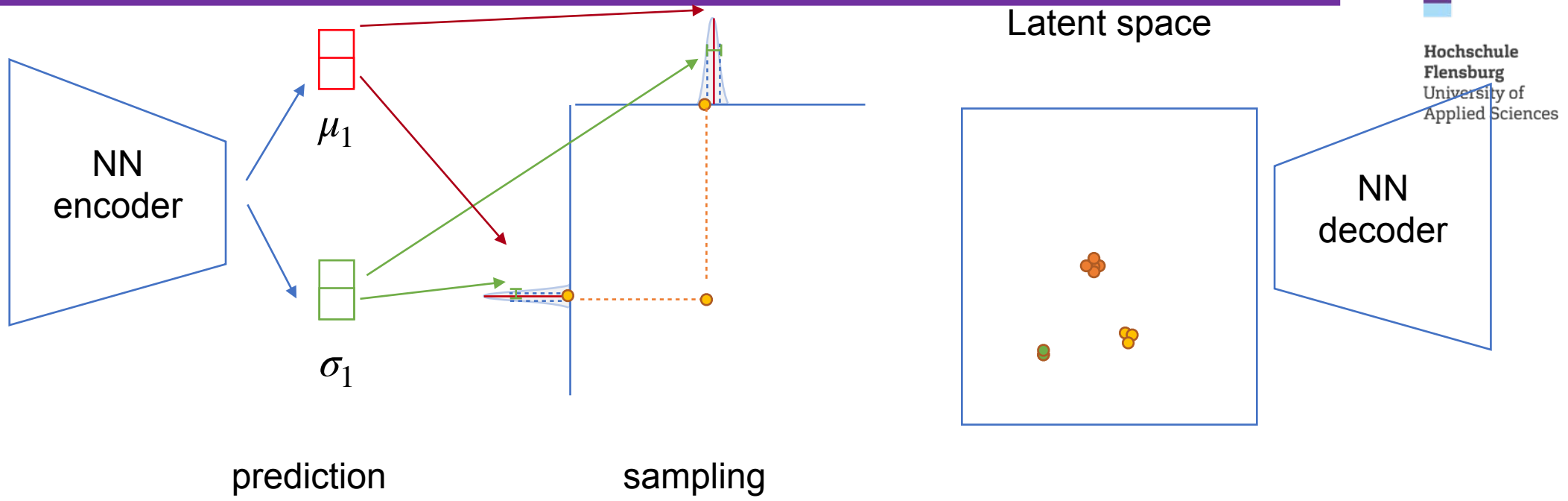
- If we refeed the first sample or a similar sample into the encoder, we again produce $[\mu_1, \sigma_1] = e(\mathbf{x}_1)$
- If the sample that is now produced is very close to the latent representation of $\mathbf{x}_2$, the decoder can't learn to decode the original image $\mathbf{x}_1$ from the latent vector.
- Hence, the encoder will have to adapt the prediction to disentangle the distributions more.

Example: Constraining the distributions / VAEs in action



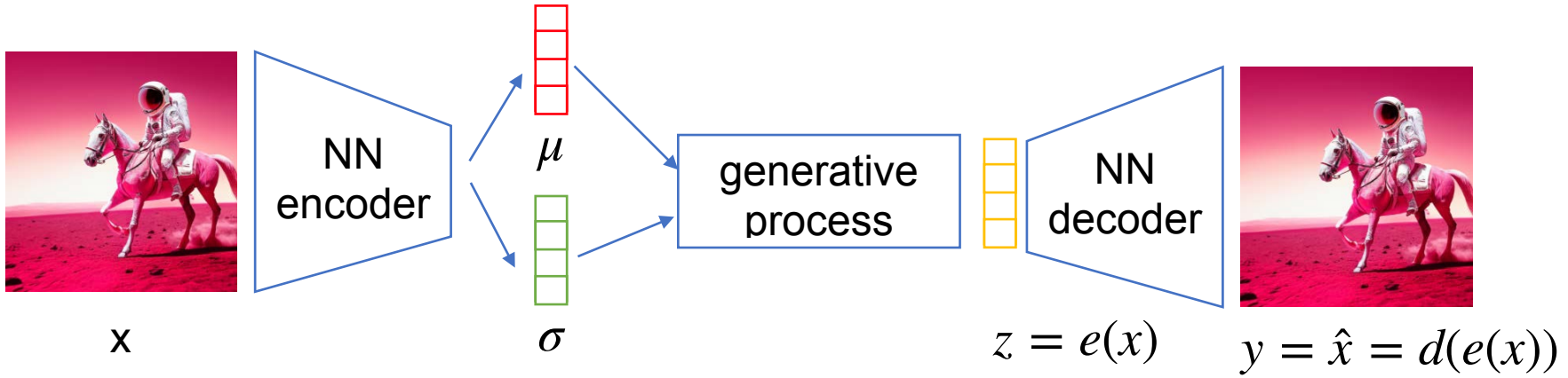
- An image $\mathbf{x}_3$ that has characteristics in-between $\mathbf{x}_1$ and $\mathbf{x}_2$ can be reconstructed without the need to push the distributions of $e(\mathbf{x}_1)$ and $e(\mathbf{x}_2)$ further apart in latent space, so the decoding can be realized without the need to further reorganize the latent space.

Example: Constraining the distributions / VAEs in action



- On the other hand, the KL divergence loss term avoids a situation depicted above where the distributions are sharp and the latent space organization is sparse.

Variational Autoencoders

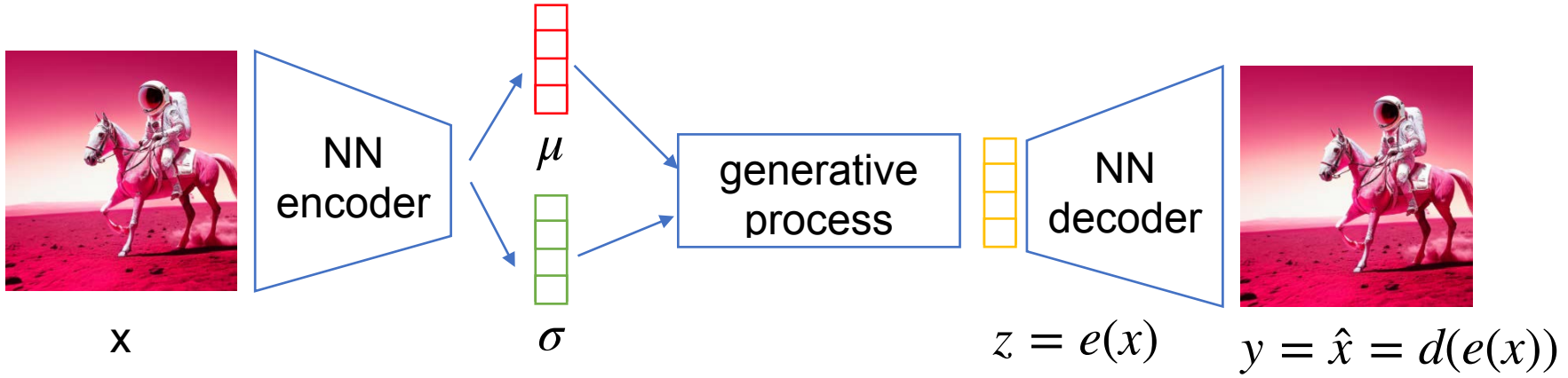


- Variational Autoencoder use both, a reconstruction loss and a regularization term (KL-Divergence).

$$L = L_{\text{recon}} + L_{KL}$$

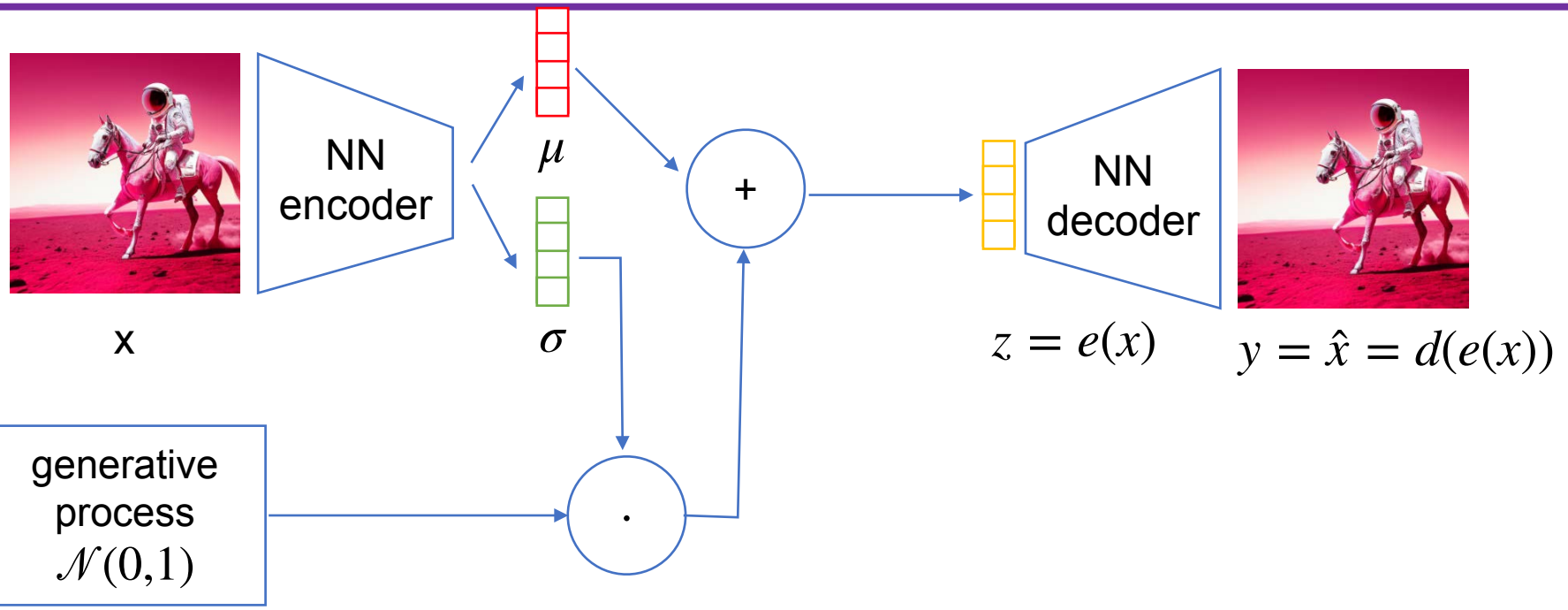
- Naturally, there is a balance between both loss terms, that can be found using a weight term.

Training Variational Autoencoders



- The generative process is, unfortunately, not differentiable.
- So we could not train this network using gradient descent with backpropagation.
- How could we approach this?

Training Variational Autoencoders: Reparametrization trick



- We move the generative process into the input space and just combine it linearly with the predicted variables (mean/std) to yield the normalized latent space.
- We call this the reparametrization trick of variational autoencoders.
- It only works, if we assume diagonal covariances, however.

VAEs: Summary

- Variational Autoencoders regularize the latent space
- They enforce the latent space to be represented as statistical distributions (normal distributions)
- During training, a latent sample is drawn from this distribution using the reparameterization trick.
- VAEs have two loss components: KL divergence and reconstruction (L1) loss

Hochschule Flensburg
University of Applied Sciences

Hochschule
Flensburg
University of
Applied Sciences

Diffusion Models

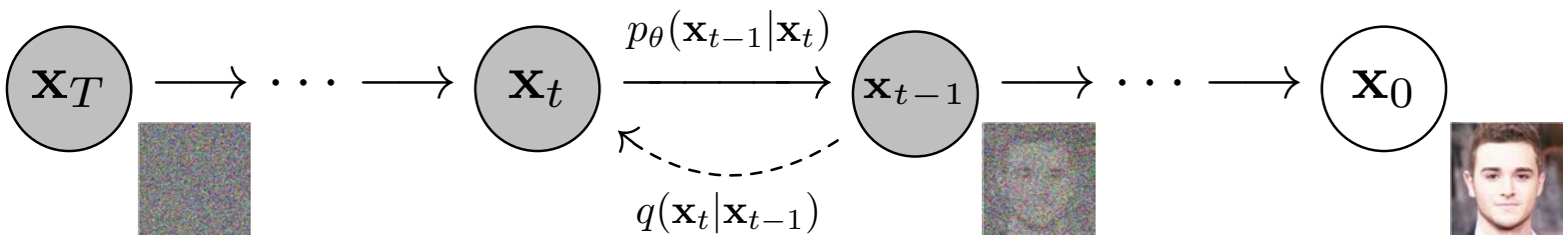
What we've seen so far

- So far, we've seen several ways to generate data:
  - Autoregressive models: build data step by step.
  - Masked models: fill in missing parts in parallel.
  - VAEs and GANs: learn to map between noise and data directly.

- Now, we approach this problem in a completely different way: by denoising.

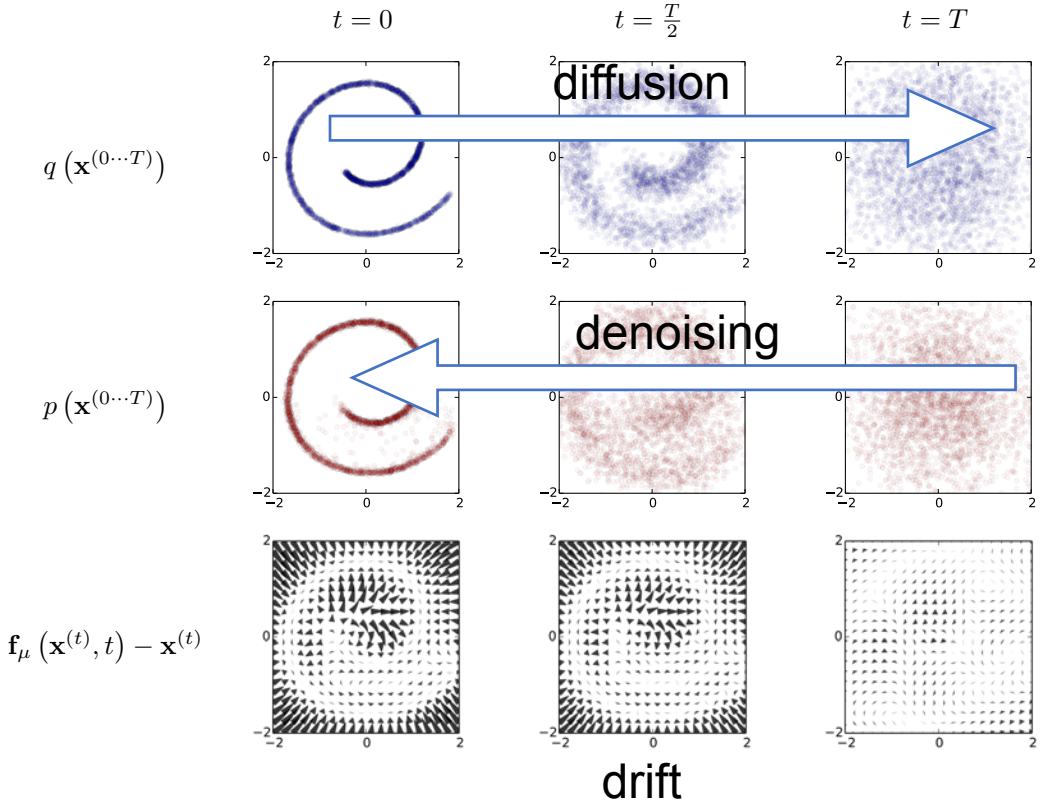
Diffusion Models: Core Idea

- Imagine gradually adding noise to an image until it becomes pure static - and then training a model to undo that destruction, one small step at a time.



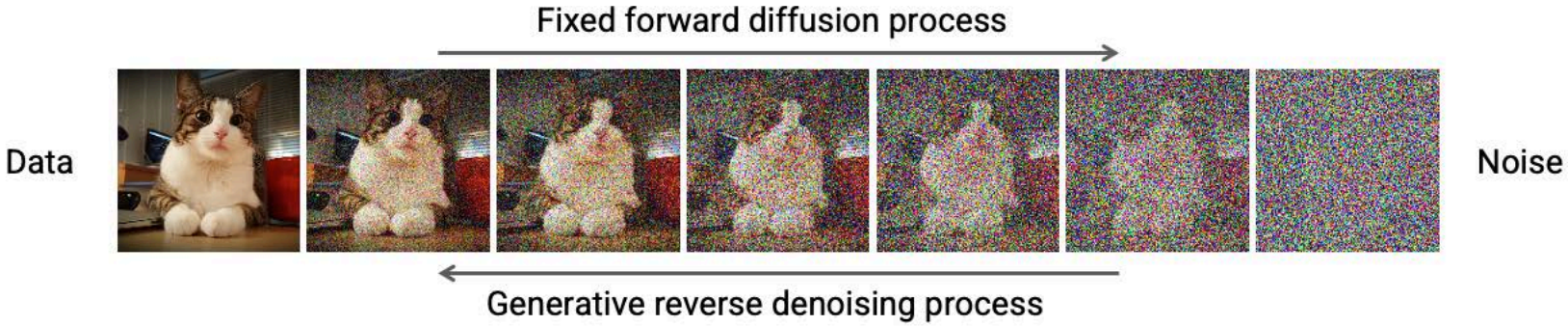
Data denoising for generation

- The idea of diffusion-based generation (and training denoising diffusion probabilistic models, DDPM) is (Sohl-Dickstein et al., 2015):
  […] to systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process. We then learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data.



https://arxiv.org/pdf/1503.03585.pdf

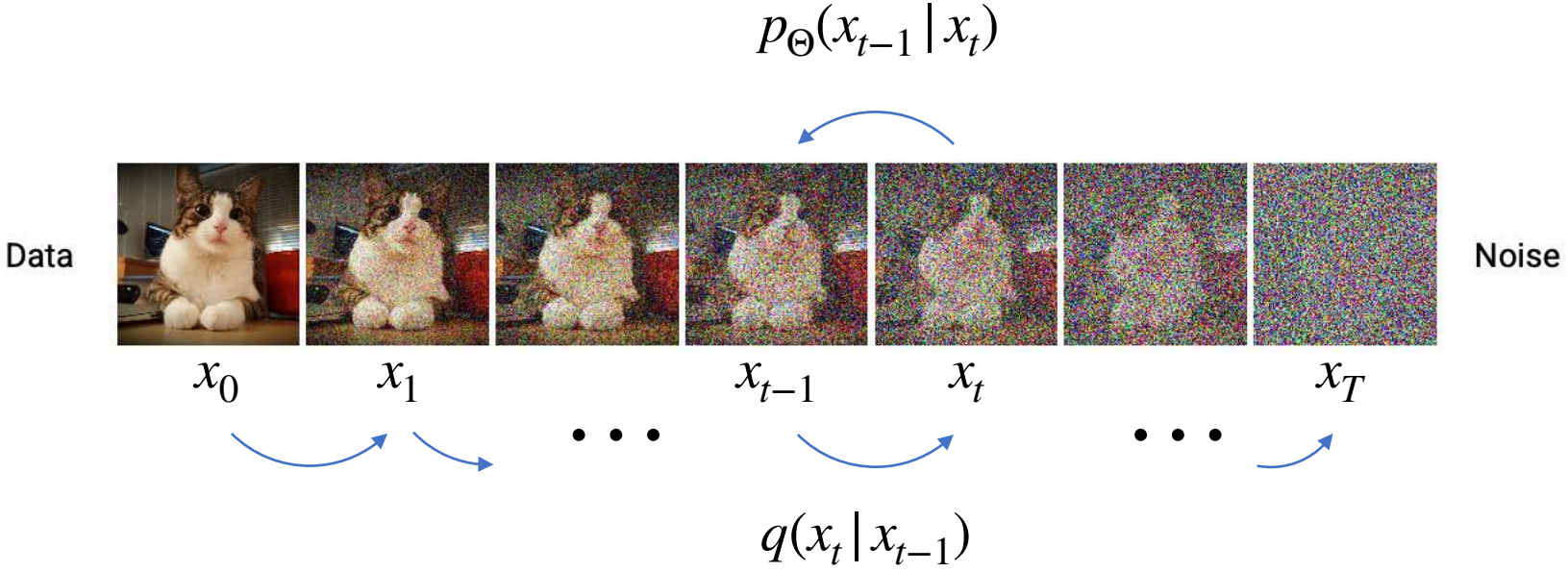
Using denoising as an image generation task



- We can formulate the image generation task as an iterative denoising task.
- The goal is to feed a model with pure noise and then gradually remove the noise until we have a clean image.
- The inverse is a stepwise process in which information dissipates - it's getting diffused.
- We can think of multiple possible permutations that would be candidates for this diffusion process - adding random noise is the simplest of them (and one with nice theoretical properties).
- Each step in this process is much simpler than directly predicting a image of the original distribution from the completely noised image.

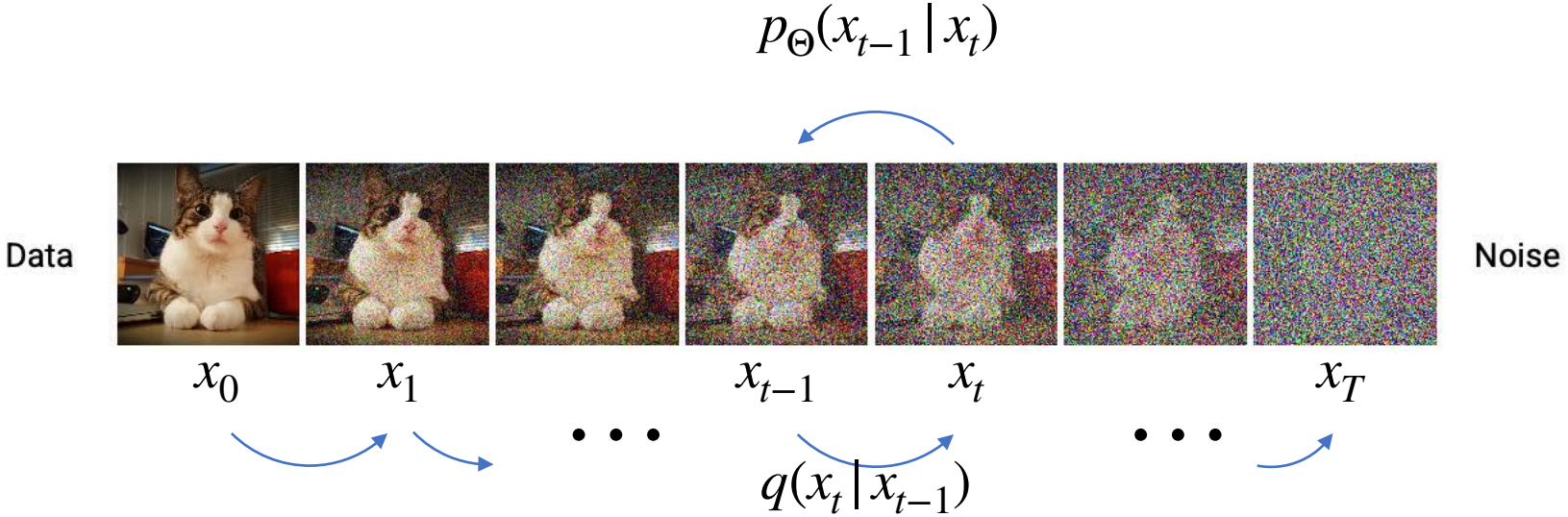
https://cvpr2022-tutorial-diffusion-models.github.io/

Denoising and diffusion



- We thus define two processes in the scope of denoising and diffusion (noising):
  - The diffusion (forward) process: $q(x_t \mid x_{t-1})$
  - The denoising (reverse) process: $p_\Theta(x_{t-1} \mid x_t)$
- The denoising process will be carried out by a model with parameters Θ.

Noising an image (Ho et al., 2020)



- The noising process can be described as a Gaussian transition from one step x_t to another x_{t-1} as:

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(\underbrace{x_t}_{\text{output}}, \underbrace{\sqrt{1-\beta_t}x_{t-1}}_{\text{mean}}, \underbrace{\beta_t I}_{\text{variance}}\right)$$

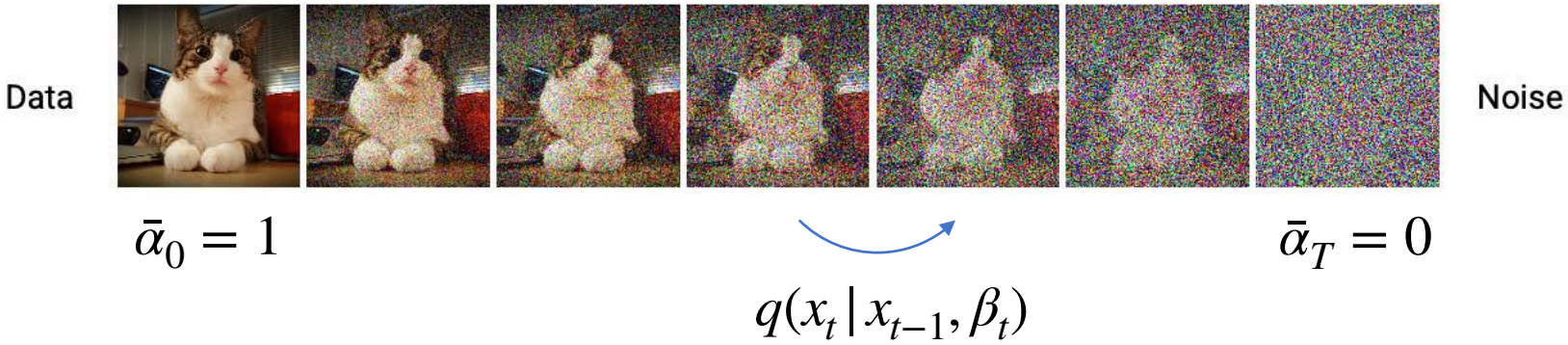
- This means the values of x_t can be expressed as the following sampling process:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1} \quad \text{where } \epsilon_{t-1} \sim \mathcal{N}(0,I)$$

- The parameter β_t decides on the variance of the process.

https://arxiv.org/abs/2006.11239
https://cvpr2022-tutorial-diffusion-models.github.io/

Noising schedule



- We can accumulate the β_t to yield an individual denoising level $\bar{\alpha}_t$ for each time step t:

$$\bar{\alpha}_t = \prod_{k=0}^{t} (1 - \beta_k)$$

- This means that we can now reformulate the noising process in a non-iterative closed form:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$

- We call the distribution of values β_t the noising schedule for the process. It is designed, so that $\bar{\alpha}_0 = 1$ and $\bar{\alpha}_T \to 0$, i.e. $x_T = \mathcal{N}(0,1)$.

What could we predict from a noised image?

- Given a noised image, we could either predict:
  - The noise within the image
  - The image itself

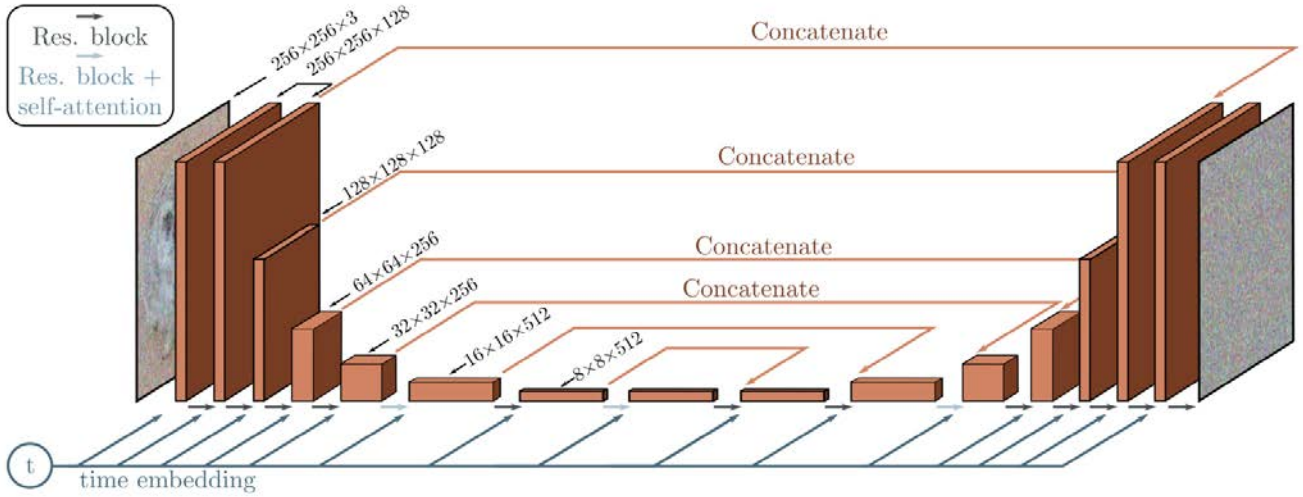
- Given exact knowledge about the noise (which is additive to the image), we can reconstruct the original image.

- Knowledge about the noise is rather easy for t=T, and more challenging for earlier time steps, but if we only want to predict the noise from one time step to another it's becoming much easier.

https://arxiv.org/pdf/2006.11239.pdf

How the model predicts the noise

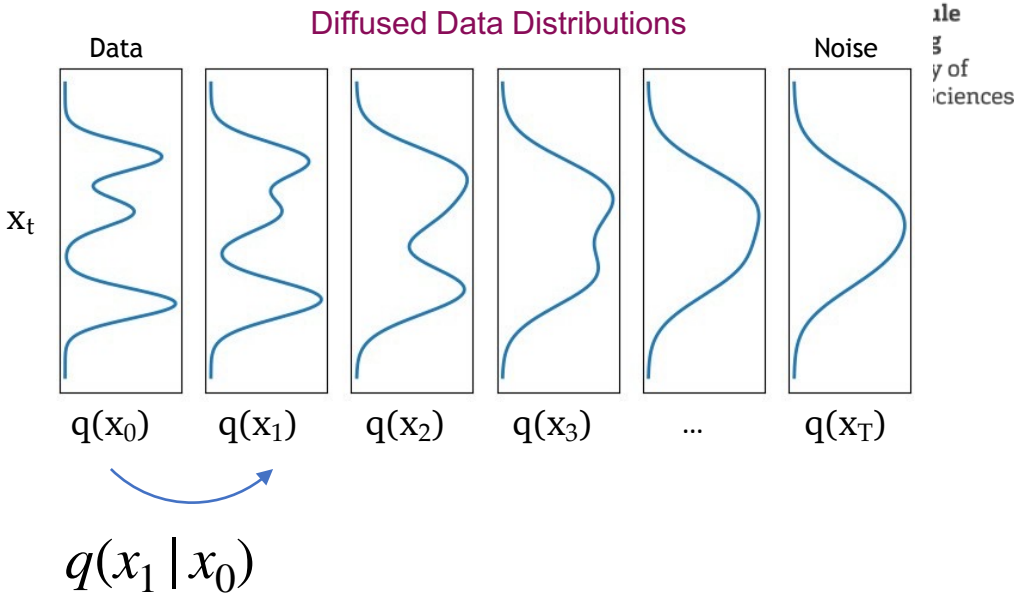
- The architecture used by the original DDPM model is a UNet-style encoder/decoder with self-attention at the bottleneck level.
- In order to help the model out in determining the magnitude of noise it will need to predict, it is informed about the timestep t using a sinusoidal embedding (like in transformers).



https://learnopencv.com/denoising-diffusion-probabilistic-models/

Data distribution from the image to the noise

- As stated, the fully noised image follows a normal distribution, while the original image follows its normal image data distribution.



- We can express the diffused data distribution $q(x_t)$ using the original data distribution $q(x_0)$ as:

$$q(x_t) = \int q(x_0) q(x_t | x_0) dx_0$$

- $q(x_t | x_0)$ is a diffusion kernel - effectively a Gaussian convolution.

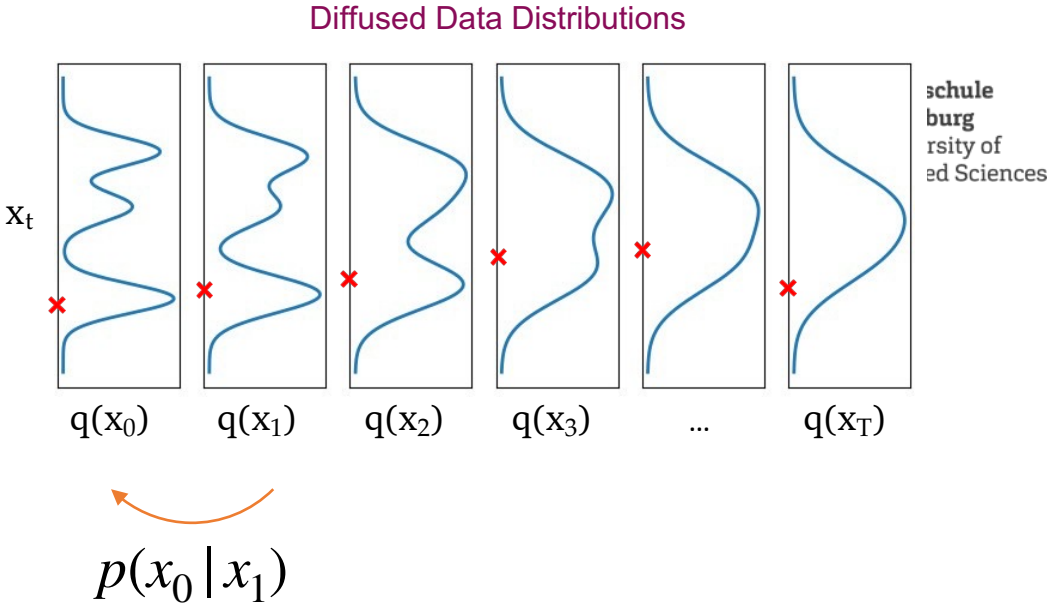
https://www.cs.unc.edu/~ronisen/teaching/fall_2022/pdf_lectures/lecture7-8_diffusion_model.pdf

Prediction of a less noisy image

- We have no analytic means of determining $p(x_{t-1} \mid x_t)$.

- But, similar to the forward process, we can model the change applied in diffusion using a normal distribution:

$$p_\Theta(x_{t-1}, x_t) = \mathcal{N}(x_{t-1}, \mu_\Theta(x_t, t), \sigma_t^2 I)$$



- The mean and standard deviation of the distribution are unknown, but can both be predicted using a neural network (in practice, we can set σ to a fixed value).
- This is very much like in VAEs, where we also used a network to predict the mean/std of a normal distribution.
- Note that all of these values (mean/std) are per coordinate of the image.

https://www.cs.unc.edu/~ronisen/teaching/fall_2022/pdf_lectures/lecture7-8_diffusion_model.pdf

Training a denoising model

- A DDPM can be trained using the following scheme proposed in the original paper:

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

predicted noise from noised image

noise, $\epsilon \sim \mathcal{N}(0, I)$

https://arxiv.org/pdf/2006.11239.pdf

Sampling from a DDPM

- The following algorithm describes image generation (sampling) from a denoising diffusion propabilistic model (DDPM):

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

start with a random noise image
start at the fully noisy time step T,
then gradually approach t=1

predict the improved image by gradually removing the predicted noise ϵ_Θ

Summary: Diffusion Models

- Start from random noise and iteratively denoise to generate data.
- Train by learning to reverse a fixed noising process.
- Model predicts the added noise (or clean data) at each step.
- Stable training, unlike GANs — no adversarial dynamics.
- Produce high-quality, diverse samples, but slow generation.

Hochschule Flensburg
University of Applied Sciences

Hochschule
Flensburg
University of
Applied Sciences

Summary

Summary

- Each model family trades off sample quality, speed, and training stability
- Autoregressive models: Generate data sequentially, each step conditioned on previous ones.
- Masked prediction models: Predict missing parts given visible context (e.g., BERT, MaskGIT).
- GANs (Generative Adversarial Networks): Learn to generate via competition between generator and discriminator.
- VAEs (Variational Autoencoders): Learn a latent space via probabilistic encoding/ decoding.
- Diffusion models: Generate by iterative denoising from random noise.

Fill the gap

- _____ models generate data step-by-step, where each element depends on all previous ones.
- _____ models predict missing parts of the input given the visible context.
- In GANs, a generator and a _____ compete in an adversarial game.
- GANs are known for producing sharp, high-fidelity outputs but can suffer from poor training _____.
- _____ models learn a continuous _____ space that can be sampled to create new data.
- _____ models generate by gradually removing noise from a random sample.

Hochschule Flensburg
University of Applied Sciences