Introduction to Deep Generative Models

Part 1

Deep Autoregressive Models

Fatemeh Saleh

Australian National University

Australian Centre for Robotic Vision

Generative Models

- Informally:
  - A model that can generate new data after learning from the dataset.
- More formally:
  - A generative model models the joint distribution $P(X, Y)$ of the observation X and the target Y.
  - A discriminative model models the conditional distribution $P(Y|X)$.

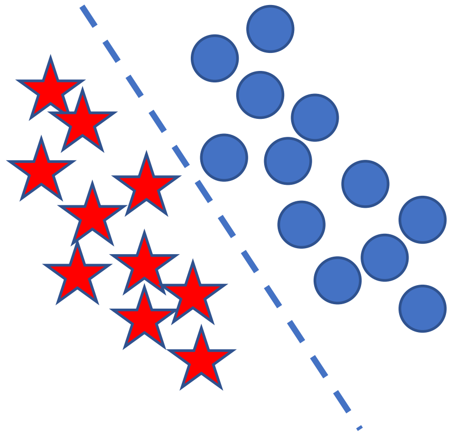
Discriminative versus Generative

- Discriminative Model
  - Tries to learn the discriminative information from the data
  - Example: Classify C1 vs C2 vs C3
    - Finds a good decision boundary by directly modeling conditional distribution $P(Y|X)$
    - Learns mappings from inputs to classes

- Generative Model
  - Tries to learn the distribution of the data
  - Models the distribution of inputs characteristic of the class
  - For classification, builds a model of $P(X|Y)$ and then applies Bayes Rule

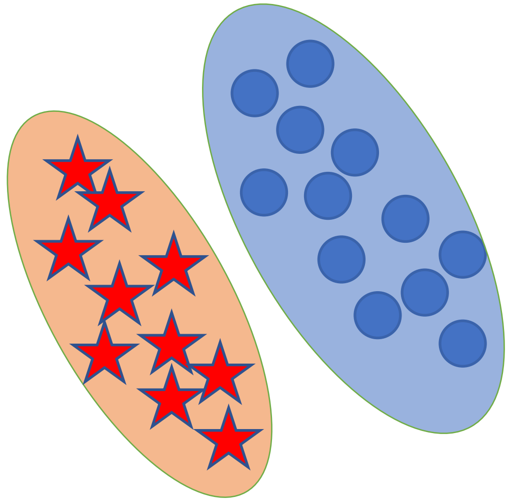
Discriminative versus Generative

Discriminative Model



Learn $P(Y|X)$ directly
Logistic regression use a sigmoid function to estimate this directly

Generative Model



Learn $P(X|Y)$ and $P(Y)$

Learning What do “stars” look like?

The class Prior
p(y=0) p(y=1)

Discriminative versus Generative

- Suppose that: $P(X|Y)$ and $P(Y)$
- New example x

likelihood   Prior

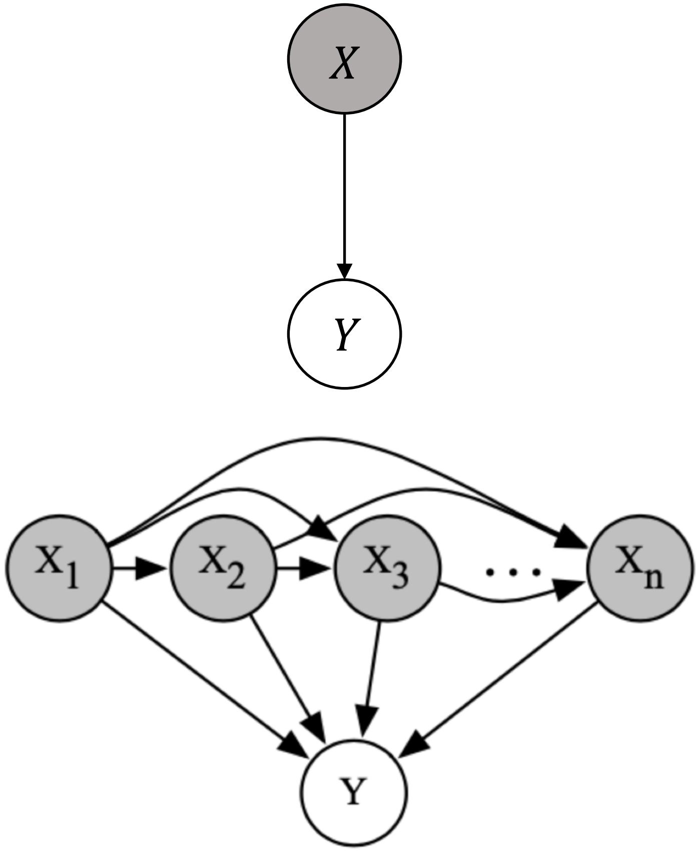
- We can then compute $P(y = 1|x) = \dfrac{p(x|y = 1)p(y=1)}{p(x)}$ (Bayes Rule)

Posterior   evidence

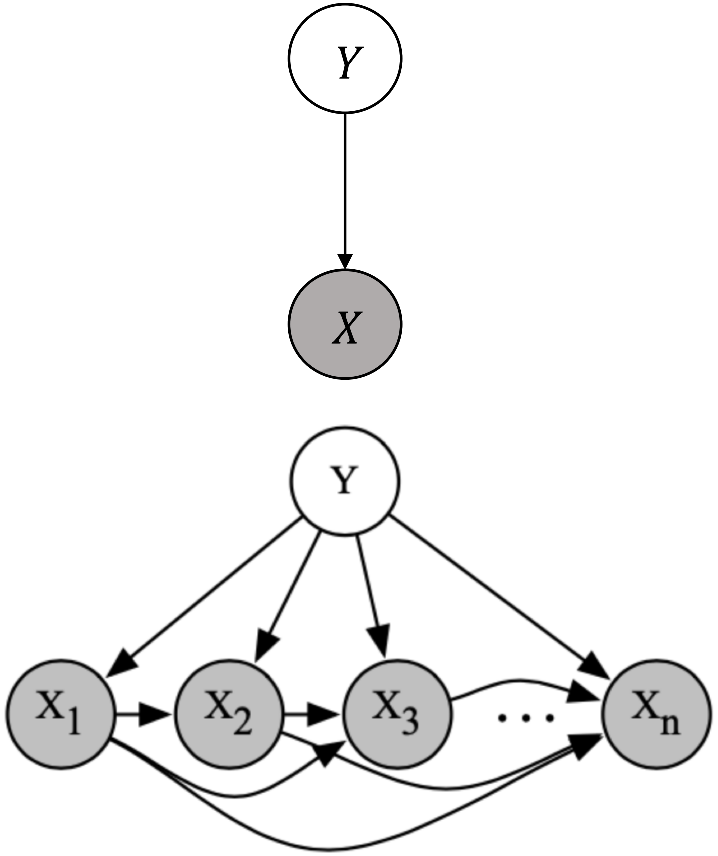
- where $p(x) = \sum_y p(x, y) = p(x|y = 1)p(y = 1) + p(x|y = 0)p(y = 0)$

Discriminative versus Generative

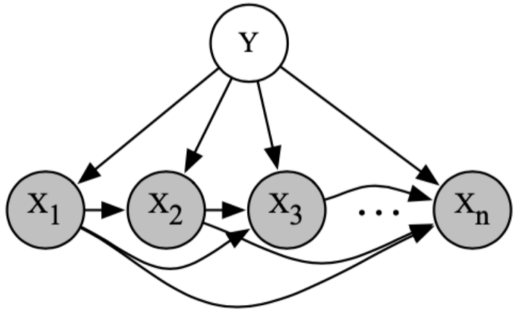
Discriminative Model



Generative Model

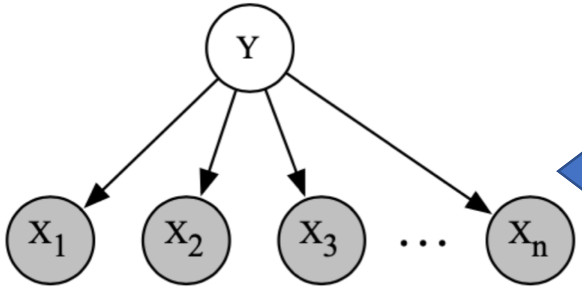


Generative Model (naive Bayes)



$$P(y, x_1, x_2, \ldots, x_n) = P(y)P(x_1|y)P(x_2|y, x_1) \ldots P(x_n|y, x_1, \ldots, x_{n-1})$$

$$x_1 \perp x_2 \perp \cdots \perp x_n \mid y$$

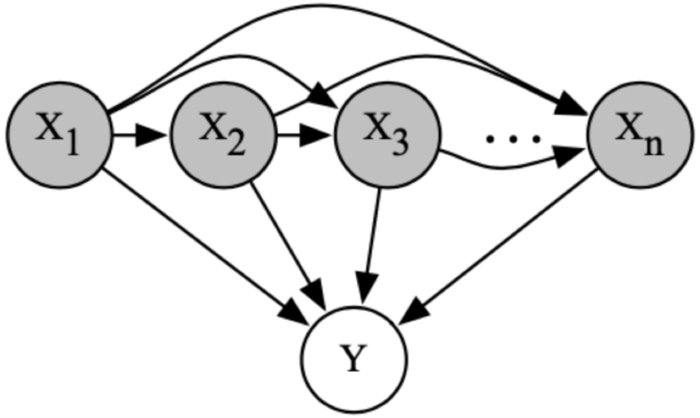


$$P(y, x_1, x_2, \ldots, x_n) = P(y)P(x_1|y)P(x_2|y) \ldots P(x_n|y) = P(y)\prod_{i=1}^{n} P(x_i|y)$$

Estimate parameters from training data. Predict with Bayes rule:

$$P(Y = 1|x_1, x_2, \ldots, x_n) = \frac{P(Y = 1)\prod_{i=1}^{n} P(x_i|Y = 1)}{\sum_{y=\{0,1\}} P(Y = y)\prod_{i=1}^{n} P(x_i|Y = y)}$$

Discriminative Model (logistic regression)



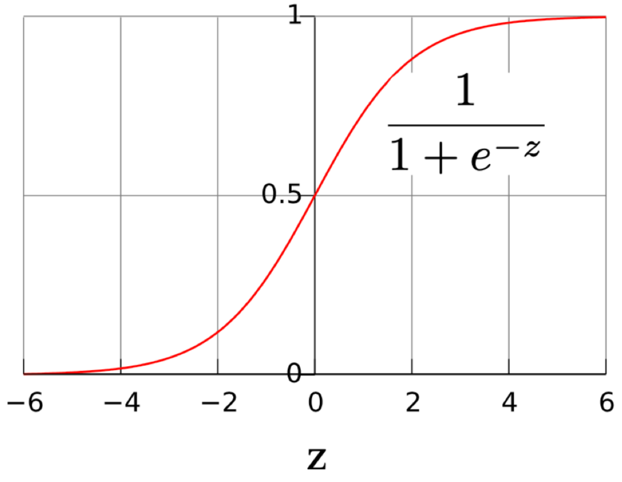
$$P(Y = 1|X;\alpha) = f(X,\alpha)$$

$$z(\alpha,\mathrm{X}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i$$

$$P(Y = 1|X;\alpha) = \sigma(z(X,\alpha))$$

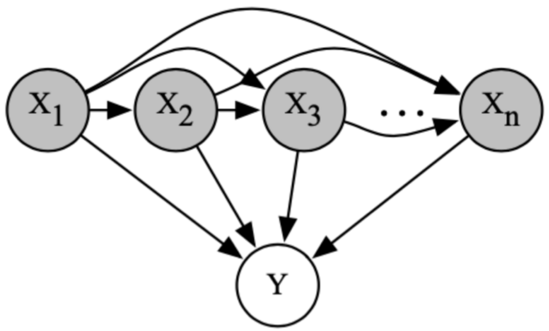
where

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

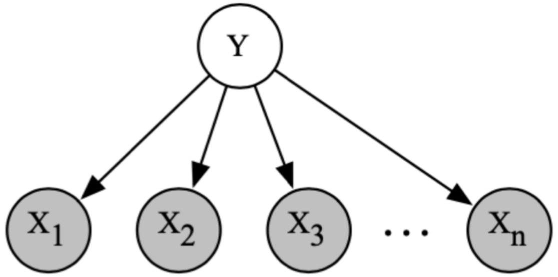


Discriminative versus Generative

Discriminative (logistic regression)



Generative (Naïve Bayes)



Discriminative models are powerful

- Logistic model does not assume $x_i \perp x_{i-1} | \, y$, unlike naïve Bayes
- This can make a big difference in many applications
    - For example, in spam classification, let $x_1 = 1[\text{“}bank\text{” } in\ email]$ and $x_2 = 1[\text{“}account\text{” } in\ email]$
    - Regardless of whether spam, these always appear together, i.e. $x_1 = x_2$
    - Learning in naïve Bayes results in $p(x_1|y) = p(x_2|y)$, thus naïve Bayes double counts the evidence
    - Learning with logistic regression sets $\alpha_1 = 0$ or $\alpha_2 = 0$, in effect ignoring it

Generative models are still very useful

- Using a conditional model is only possible when X is always observed

Credit: https://deepgenerativemodels.github.io/

Generative Models

- Given a training set of examples, e.g., images of cats:

$x_i \sim P_{data}$
$i = 1,2, \ldots, n$

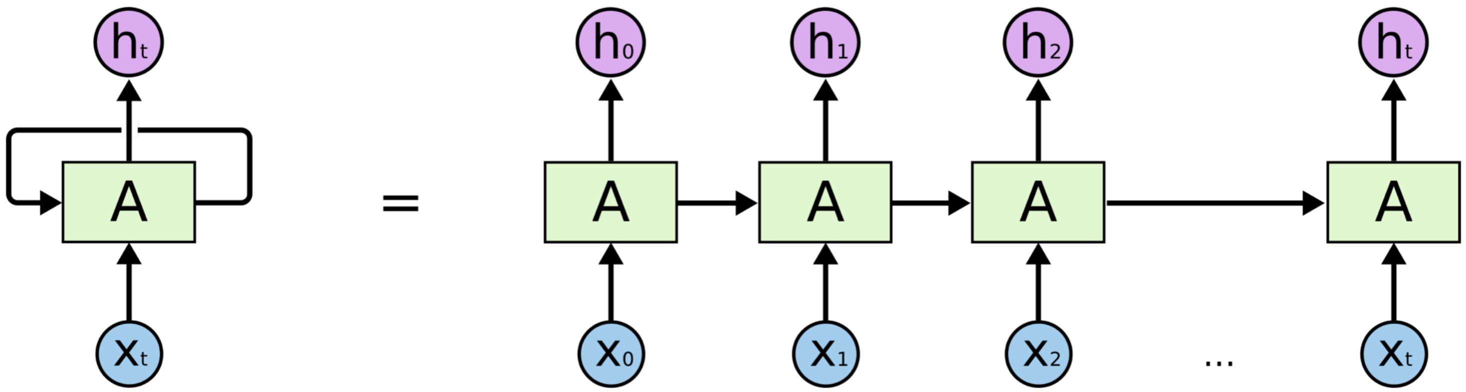
- We want to learn a probability distribution $p(x)$ over images x for
  - **Generation:** If we sample $x_{new} \sim p(x)$, x_{new} should look like a cat (sampling)
  - **Density estimation:** $p(x)$ should be high if x looks like a cat, and low otherwise
  - **Unsupervised representation learning:** The model should be able to learn what these images have in common, e.g., ears, tail, etc. (features)

Autoregressive Models

Autoregressive Models

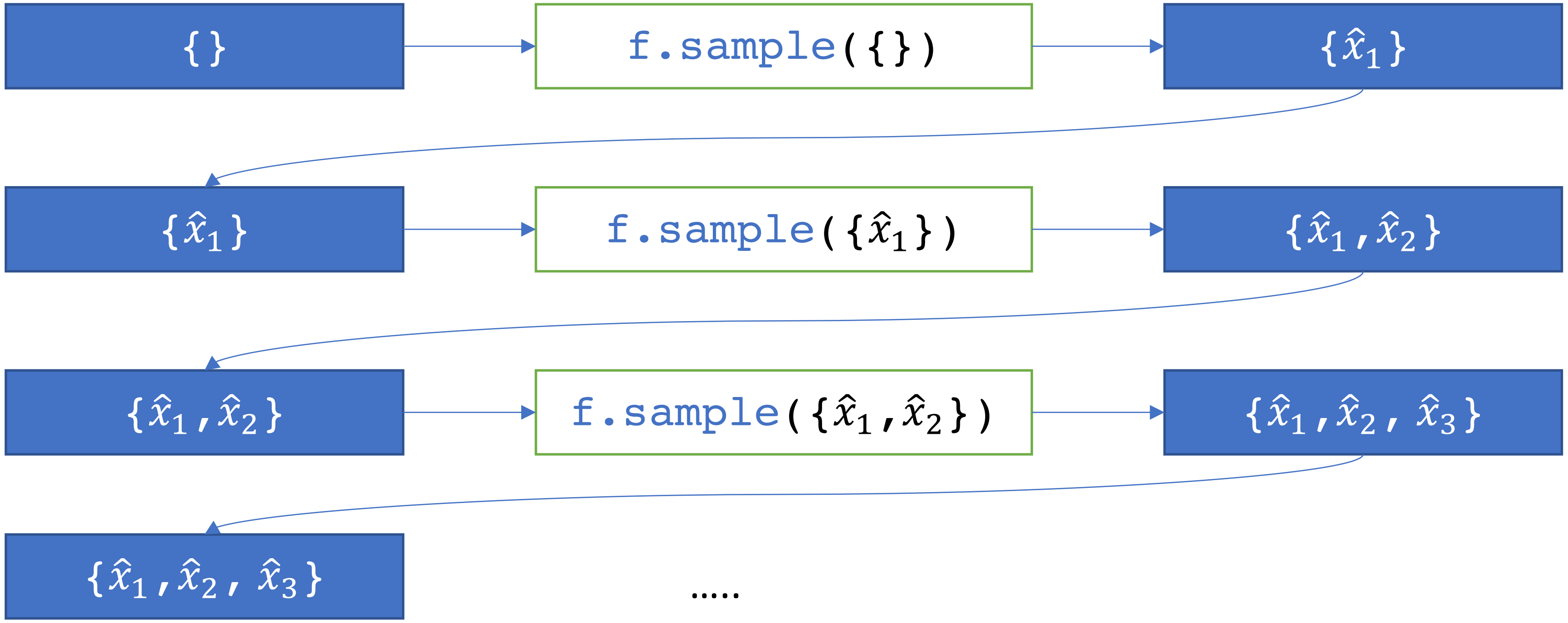
- Definition:
  - Given the observation as sequences $x_1, x_2, \dots, x_T$, we decompose the likelihood into a product of conditional distributions:

$$p(x_1, x_2, \dots, x_T) = \prod_{t=1}^{T} p(x_t | x_1, \dots, x_{t-1})$$



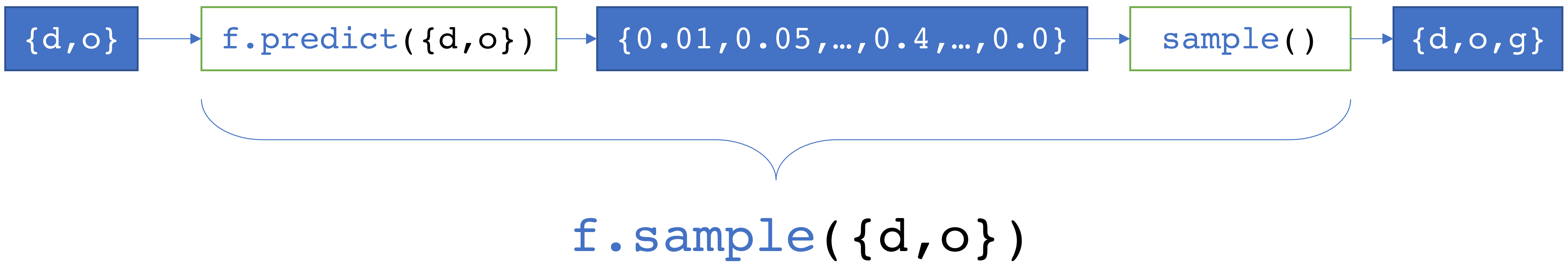
Autoregressive Models

- Given such a model `f` and a sampling function `sample`, the generative process for a full sequence is:

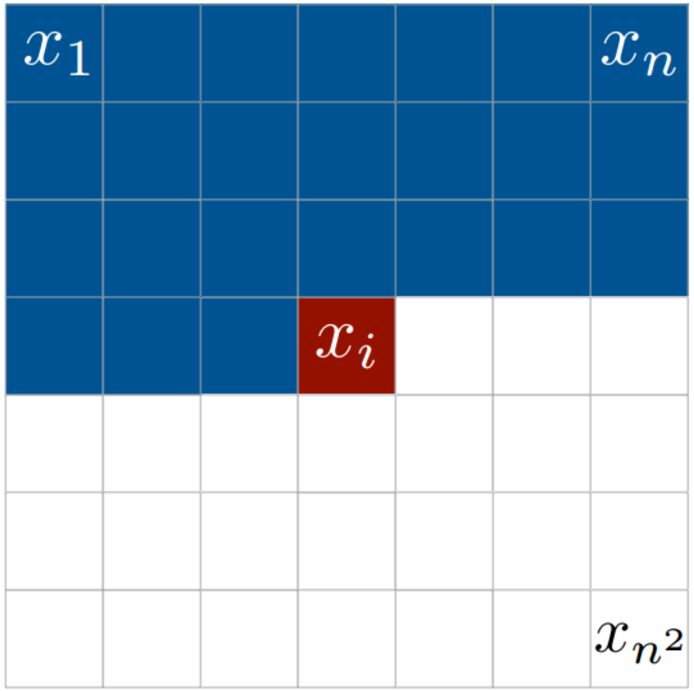


Autoregressive Models

- How `f.sample(.)` works?
  - Consider the example of character generation, where we only have `C` possible options to choose from.
  - The model maps the `d` dimensional data to `C` dimensional probability distribution and then samples from the estimated distribution.

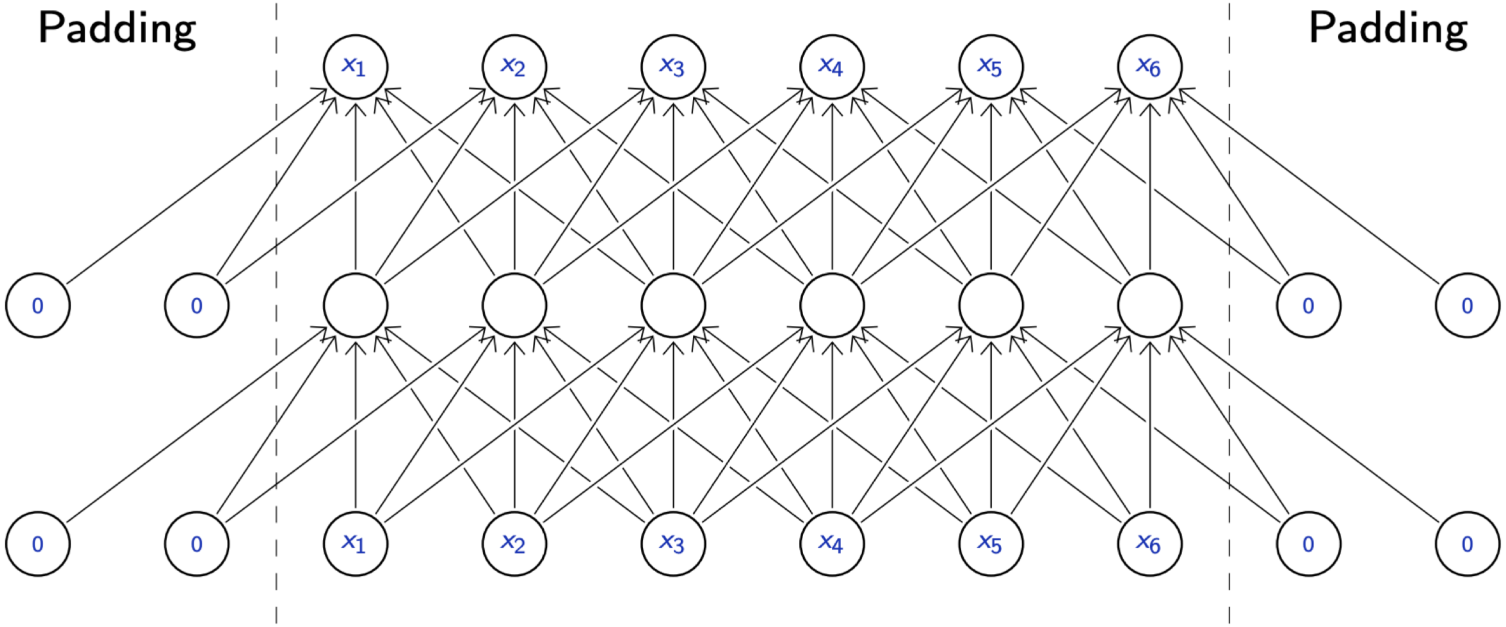


Autoregressive Models



Autoregressive Models

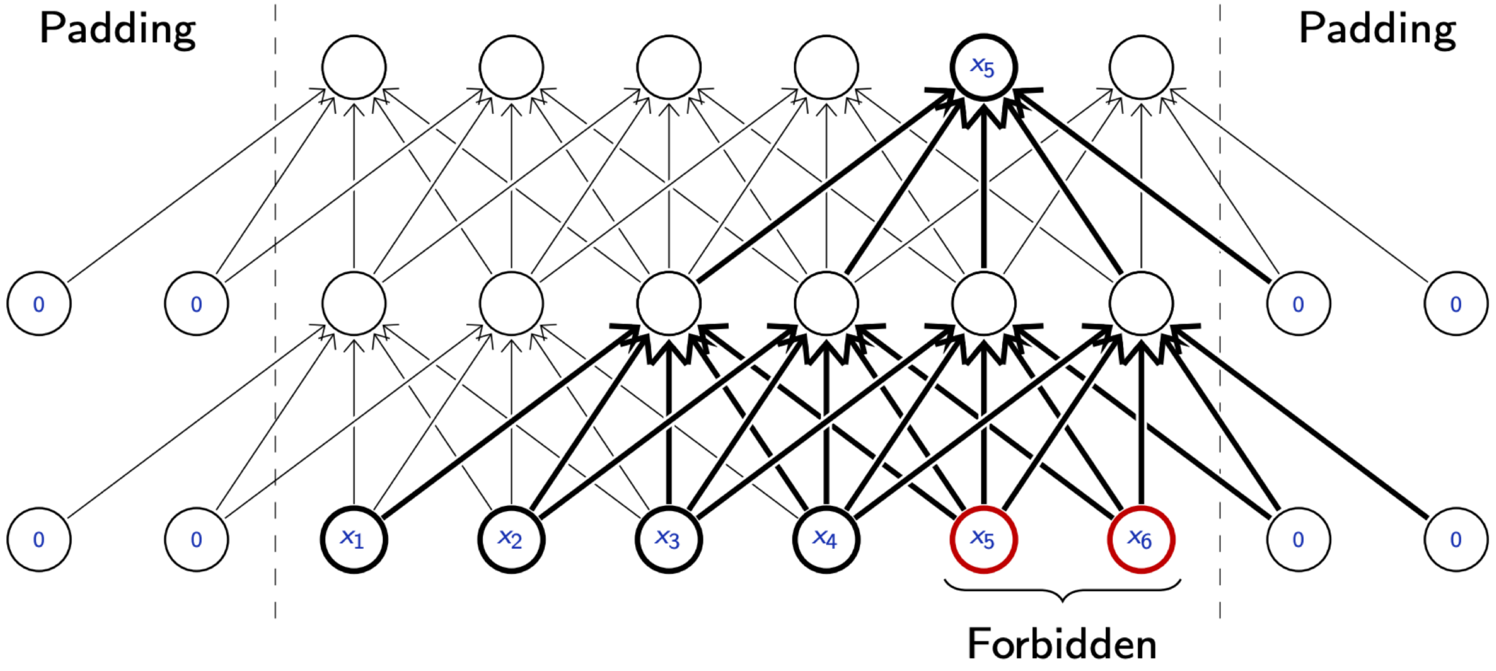
- One solution is to use *causal convolutions.*
  - How to create causal convolutions?



Credit: https://fleuret.org/ee559/

Autoregressive Models

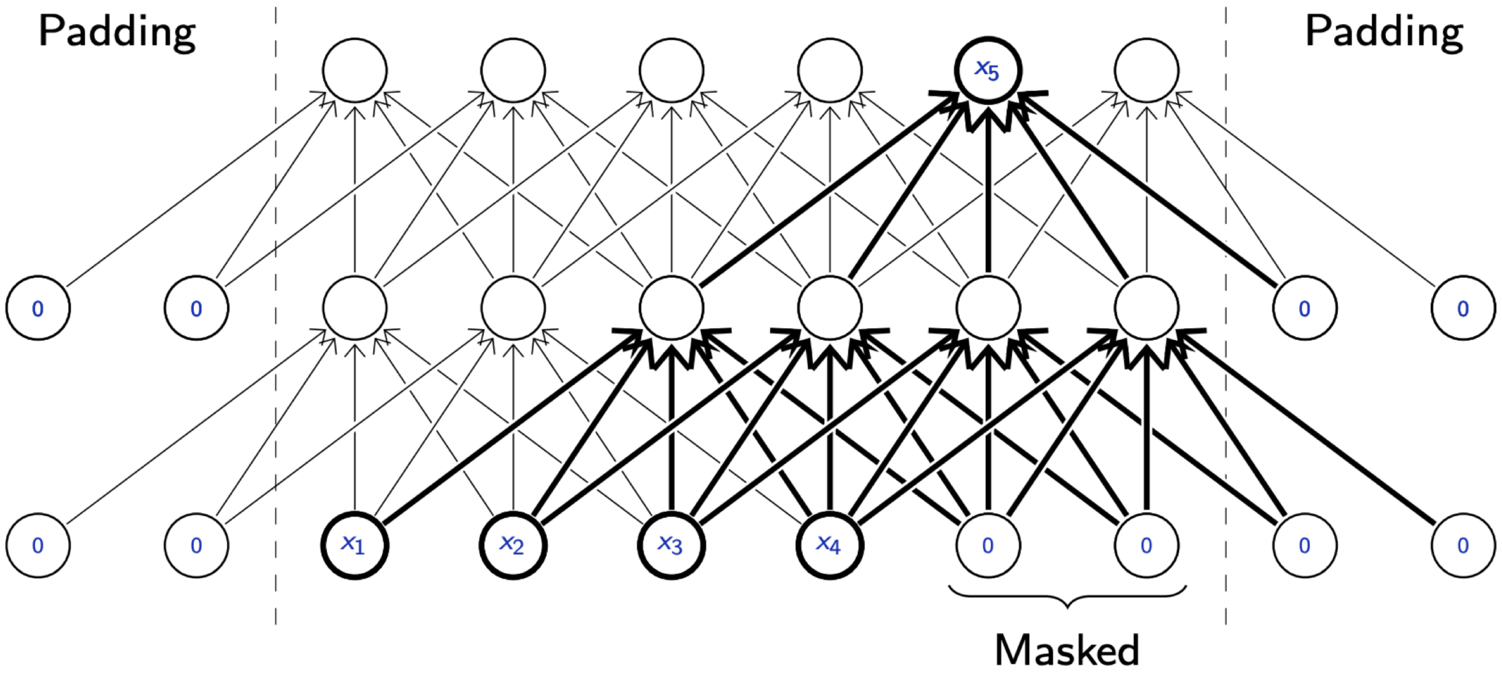
- One solution is to use *causal convolutions.*
  - How to create causal convolutions?



Credit: https://fleuret.org/ee559/

Autoregressive Models

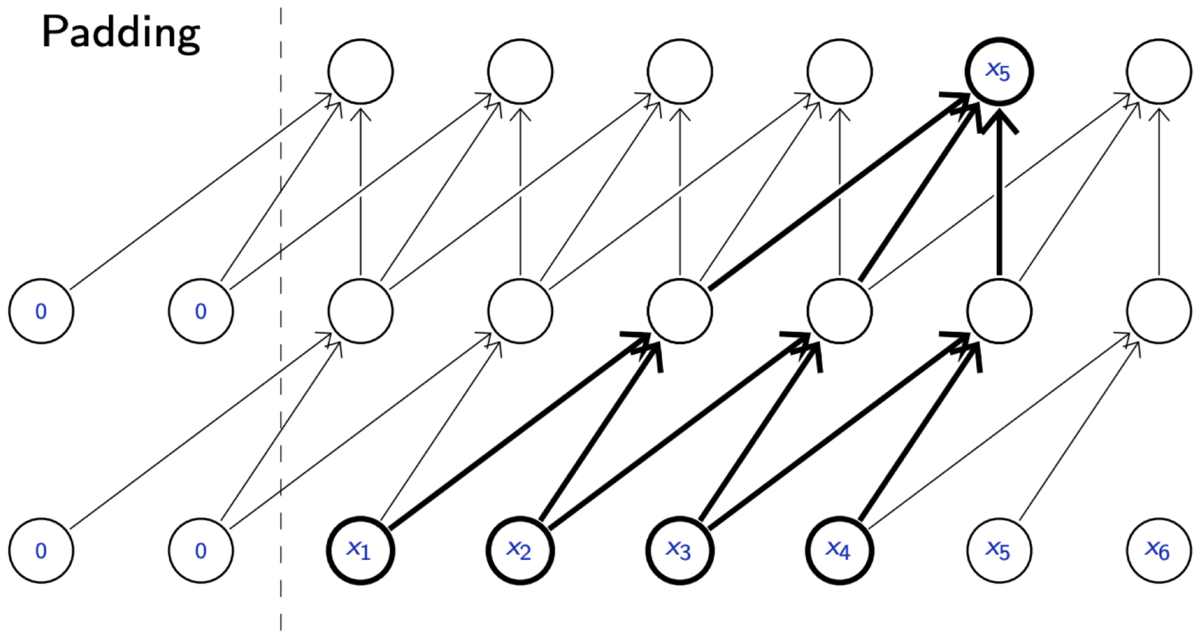
- One solution is to use *causal convolutions.*
  - How to create causal convolutions?



Credit: https://fleuret.org/ee559/

Autoregressive Models

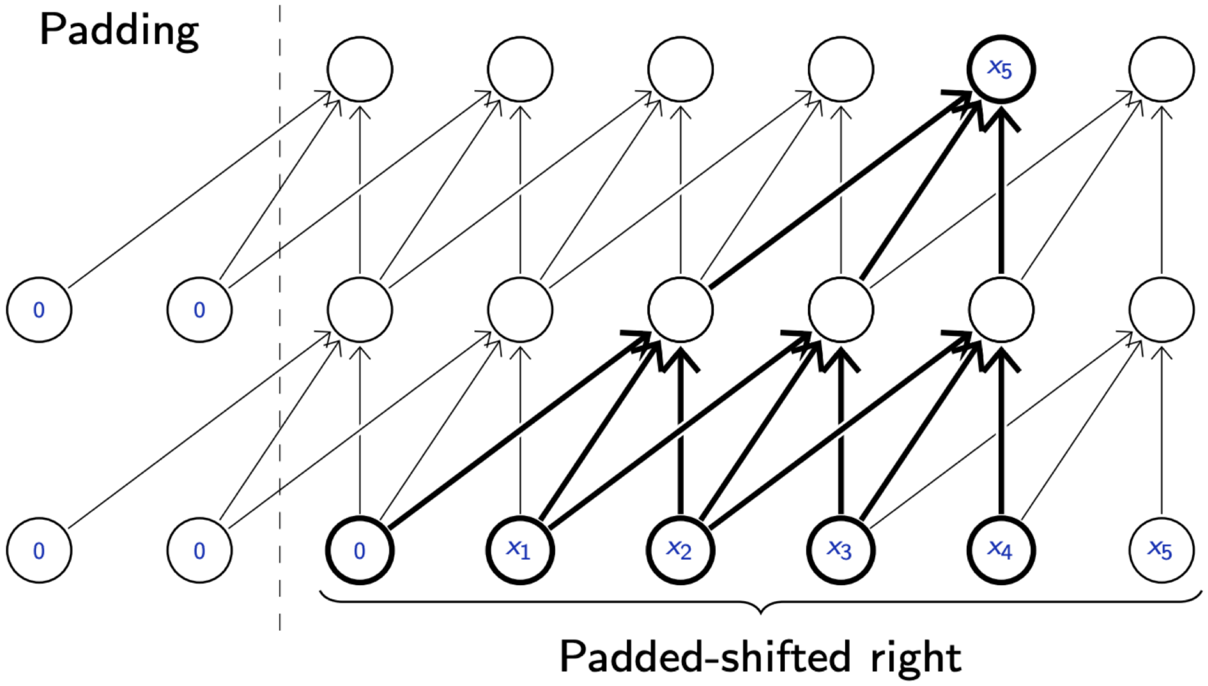
- One solution is to use *causal convolutions.*
  - How to create causal convolutions?



Credit: https://fleuret.org/ee559/

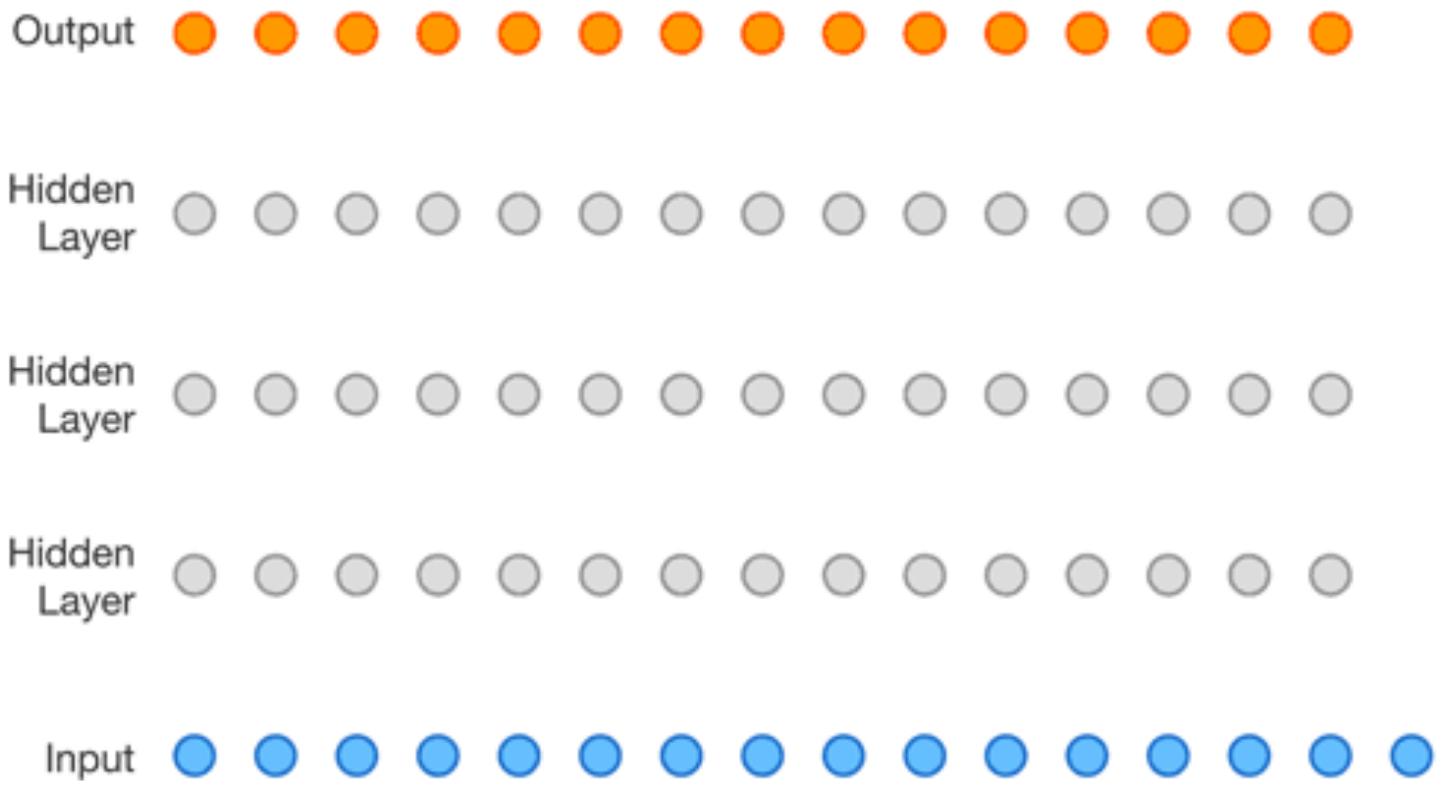
Autoregressive Models

- One solution is to use *causal convolutions.*
  - How to create causal convolutions?



Credit: https://fleuret.org/ee559/

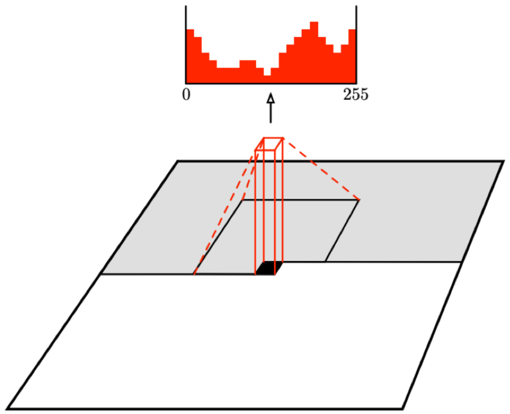
Autoregressive Models



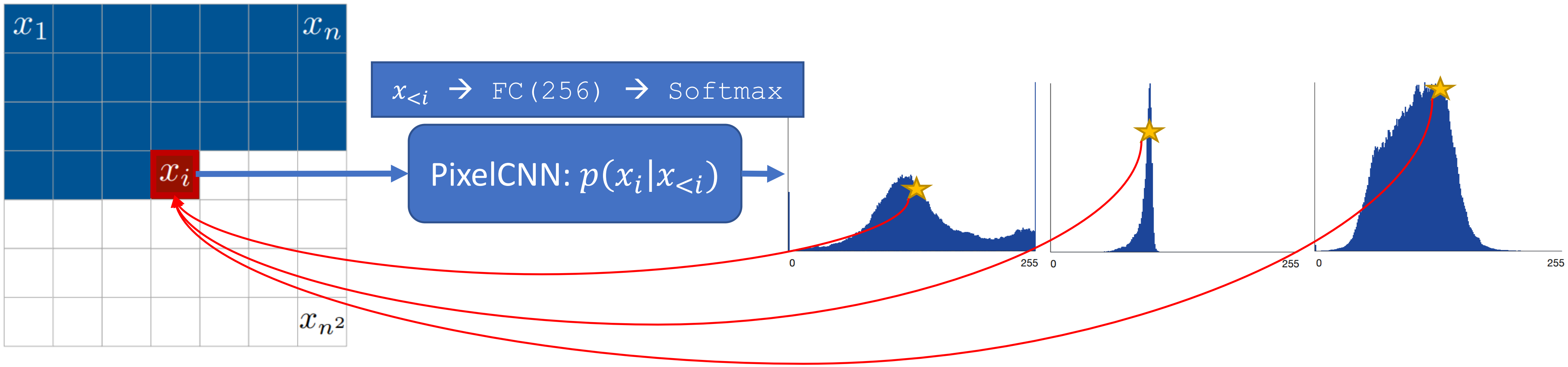
van der Oord et al. 2016

Autoregressive Models

- Example: PixelCNN



1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0



van der Oord et al. 2016

An Example of Autoregressive Generation

Human Motion Prediction

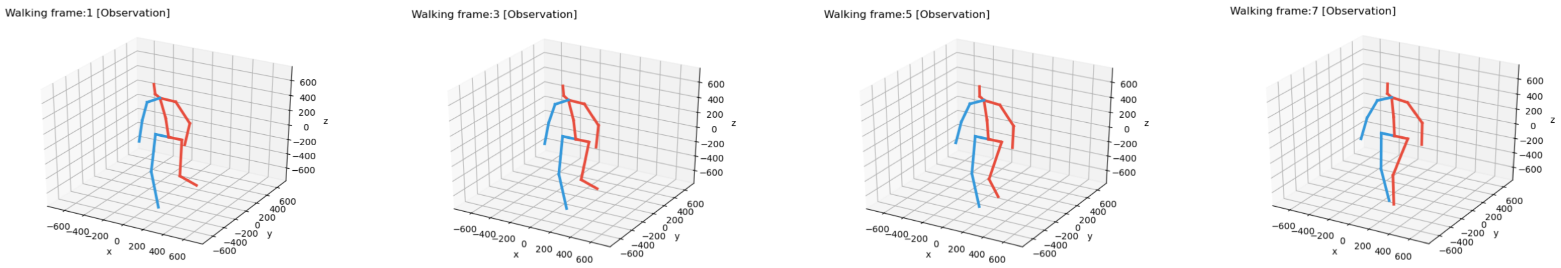
Autoregressive Models

- Let’s look at an implementation of autoregressive human motion prediction. This code contains
  - Definition of the `model`
  - training data
  - `training` script
  - `sampling` script

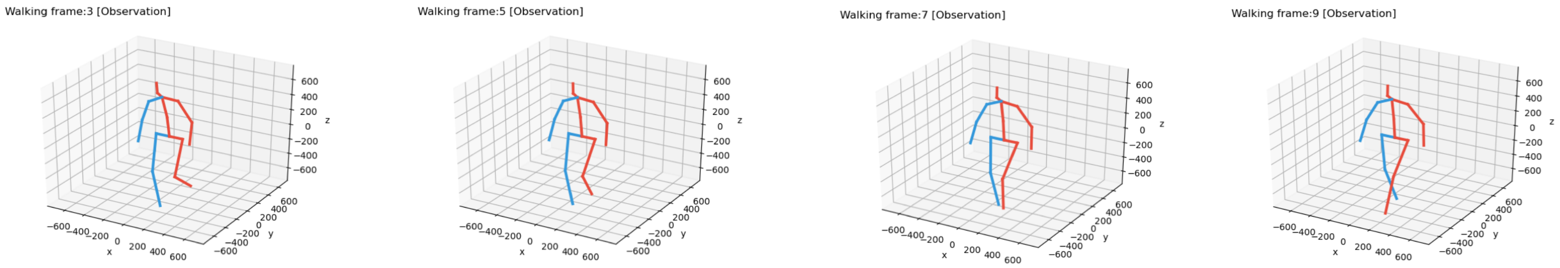
- *Note: Here, we regress the future human pose as opposed to estimating a probability distribution and then sampling a pose from that.*

Motion Prediction (**`data`**)

- A sequence of pose: `data` in the form of `B x T x J`
  - Input during training: `data[:, :-1, :]`

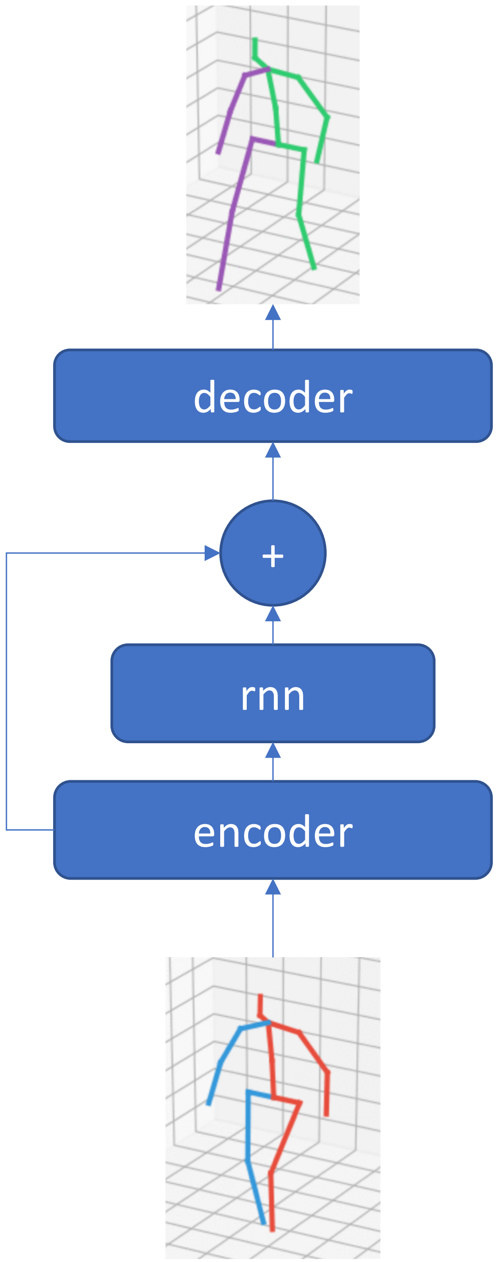


  - Output during training: `data[:, 1:, :]`



Motion Prediction (`model`)

```python
import torch
from torch import nn


class MotionModel(nn.Module):
    def __init__(self, data_dim=96, embedding_dim=256, hidden_dim=256):
        super(MotionModel, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(data_dim, embedding_dim),
            nn.ReLU())
        self.rnn = nn.GRU(embedding_dim, hidden_dim, num_layers=2)
        self.decoder = nn.Sequential(
            nn.Linear(hidden_dim, embedding_dim),
            nn.ReLU(),
            nn.Linear(embedding_dim, data_dim))
```



Motion Prediction (`training` and `forward`)

```python
def train(model, trainloader, test_seq):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(100):
        model.train()
        for batch_idx, (_, _, data) in enumerate(trainloader):
            data = data.cuda()
            inputs = data[:, :-1, :]
            outputs = data[:, 1:, :]

            optimizer.zero_grad()
            predictions = model(inputs)
            loss = criterion(predictions,
                             outputs)
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), 0.1)
            optimizer.step()

```

```python
    def single_forward(self, x, h=None):

        x_e = self.encoder(x)
        x, h = self.rnn(x_e, h)
        x = x + x_e.clone()
        return self.decoder(x), h

    def forward(self, data):

        rec, h = self.single_forward(data)
        return rec

```

Motion Prediction (`generation`)

```
    def sample(self, observation, T=50):
        output = []
        obs_len = observation.shape[1]

        rec, h = self.single_forward(observation[:, 0:1, :])

        for t in range(1, obs_len):
            rec, h = self.single_forward(observation[:, t:t+1, :], h)

        output.append(rec)

        for t in range(T-1):
            rec, h = self.single_forward(rec, h)
            output.append(rec)

        return output
```

Forwarding the observation

Motion Prediction (`generation`)

```
    def sample(self, observation, T=50):
        output = []
        obs_len = observation.shape[1]

        rec, h = self.single_forward(observation[:, 0:1, :])

        for t in range(1, obs_len):
            rec, h = self.single_forward(observation[:, t:t+1, :], h)

        output.append(rec)

        for t in range(T-1):
            rec, h = self.single_forward(rec, h)
            output.append(rec)

        return output
```

We use the groundtruth motion as the input to the model

Motion Prediction (`generation`)

```python
    def sample(self, observation, T=50):
        output = []
        obs_len = observation.shape[1]

        rec, h = self.single_forward(observation[:, 0:1, :])

        for t in range(1, obs_len):
            rec, h = self.single_forward(observation[:, t:t+1, :], h)

        output.append(rec)

        for t in range(T-1):
            rec, h = self.single_forward(rec, h)
            output.append(rec)

        return output
```

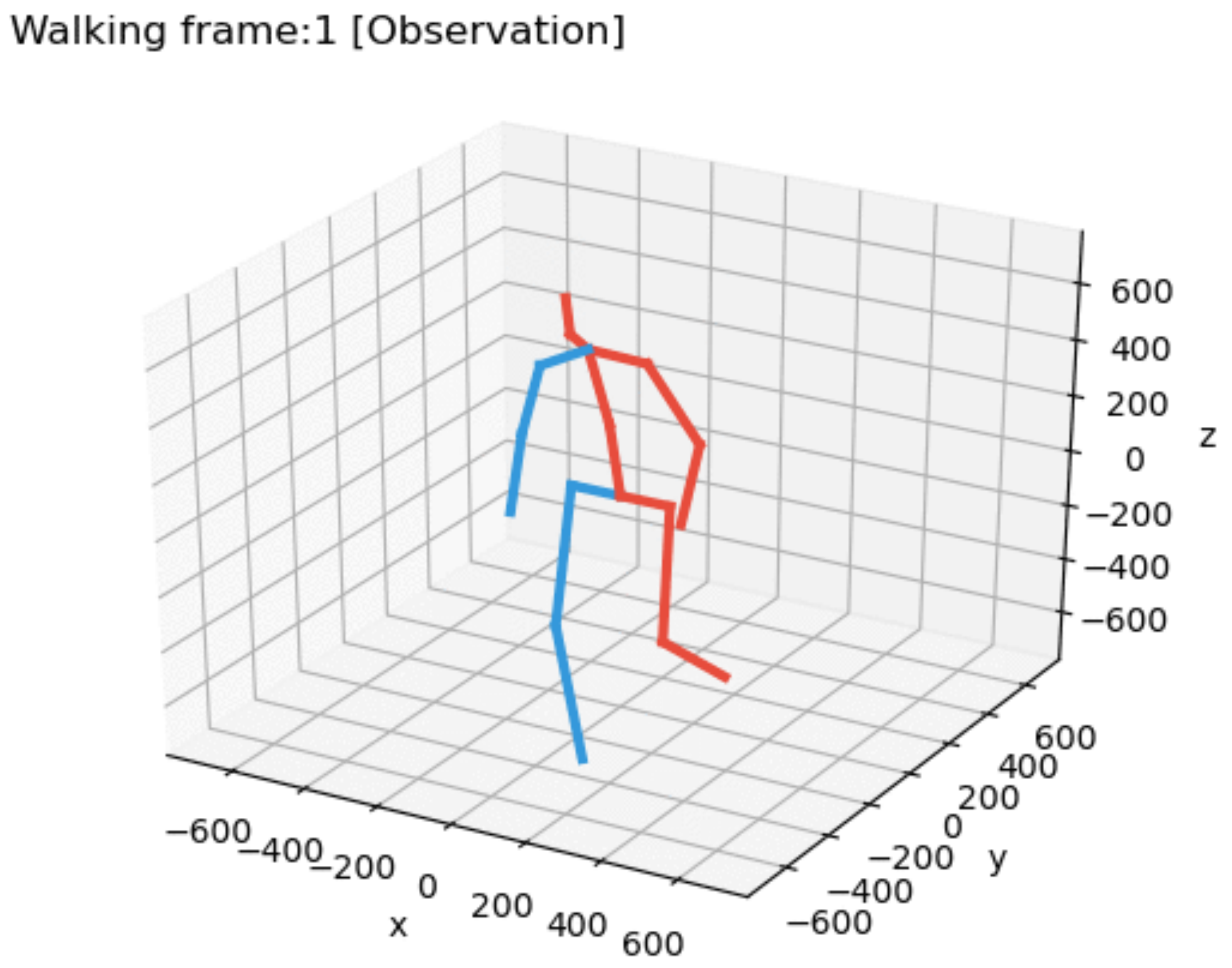
Given the last observation, we generate the first future motion

Motion Prediction (`generation`)

```
    def sample(self, observation, T=50):
        output = []
        obs_len = observation.shape[1]

        rec, h = self.single_forward(observation[:, 0:1, :])

        for t in range(1, obs_len):
            rec, h = self.single_forward(observation[:, t:t+1, :], h)

        output.append(rec)

        for t in range(T-1):
            rec, h = self.single_forward(rec, h)
            output.append(rec)

        return output
```

Autoregressively, we generate future poses; given last prediction, we generate the next motion

Motion Prediction (`results`)



Summary

- An overview of generative versus discriminative models
- Autoregressive models
  - Pros:
    - Easy to sample from
    - Easy to compute the likelihood
    - Natural to train via maximum likelihood
  - Cons:
    - Slow to generate new data
    - Does not directly learn unsupervised representations of the data.
- Materials:
  - https://deepgenerativemodels.github.io/
  - https://fleuret.org/ee559/
  - Van Den Oord, et.al. "Pixel recurrent neural networks." ICML. 2016.