

## Allg. Architektur von Vol. Prototyp GAN

### Discriminator:

#### Input:

$$\begin{matrix} n \\ b \end{matrix} \quad \boxed{\text{pianoroll}} = \mathbf{P} \in \{0,1\}^{n \times b} \quad \text{M: n.b. anzahl Features}$$

or  $I_D$  = input for discriminator

n = mögliche Noten  
b = anzahl Beats

#### Output:

$$\text{Jugement: } p(\text{"input is real"}) \quad \xleftarrow{\text{noch keine Genre-Klassifikation}} \quad J \in [0,1]$$

(mit Augmented) or  $O_D$  = output of discriminator

noch keine velocity, etc. Info

### Generator

#### Input:

$$\text{zufälliger Input vector: } \hat{\mathbf{P}} \in \mathcal{N}(0,1)^L$$

L = Länge vom input random input or  $I_G$  = input generator

noch keine Gene, etc. Spezifikationen

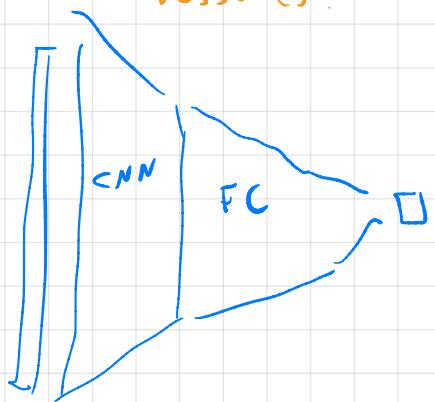
#### Output:

$$\hat{\mathbf{P}} = \text{generated pianoroll}$$

### Aufbau:

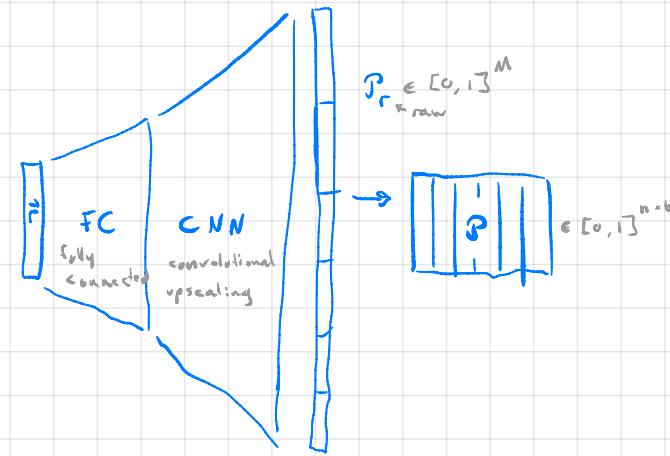
ungefähr spiegel verkehrt zu Generator?

oder gibt es was besseres?



### Aufbau:

wie VAE-Decoder (siehe ex 04)



guter Ansatz?

→ z.B. sind conv. layers ein

### Training ↓ nächste Seite

## Training

### Loss-Funktion

$$\mathcal{L} = \mathbb{E}_{\mathbf{x} \sim p_d}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_z}[D(G(\mathbf{z}))] + \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}[(\nabla_{\hat{\mathbf{x}}} \|\hat{\mathbf{x}}\| - 1)^2]}$$

from MuseGAN paper bc. better & faster convergence  
than original

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_d}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z} \sim p_z}[1 - \log(D(G(\mathbf{z})))]$$

$\mathbf{x} = \vec{P}^*$  = real piano rolls

$\mathbf{z} = \vec{r}$  = random vector

$\hat{\mathbf{x}} := \tilde{\vec{P}}$  = mix between real and fake piano rolls, see \*

\* gradient penalty more precisely: (from gradient penalty paper)

$$\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2]$$

$$\hat{\mathbf{x}} = \tilde{\vec{P}} = \varepsilon \cdot \vec{P}^* + (1-\varepsilon) \cdot \vec{P}$$

one real piano roll

↑

one generated piano roll

w/  $\varepsilon \in U[0, 1]$  for each sample

$\lambda = 10$  in paper (found to be robust for many cases)

Can a discrete generator output lead to worse training?

How to discretize best?

Trainingsablauf:

✓  $\mathcal{T}$  enthält  $N$  piano rolls  $\vec{P}^*$

Benötigt: a) vorbereitetes Trainingsset  $\mathcal{T}$  an piano rolls (einen Teil zum Testen abgespalten)  
b) jew. einen zufällig initialisierten Gen. und Dis.

1.  $\mathcal{T}$  aufteilen in Batches  $\vec{P}_B^*$  (enthalten  $B$  piano rolls  $\vec{P}^*$ )

2. Forward-pass:

a)  $\vec{J}_B^* = D(\vec{P}_B^*) \in [0, 1]^B$  berechnen

b)  $\vec{r}_B \in \{\vec{r}\}^B \mathcal{N}(0, I)^{B \times L}$  berechnen

c)  $\hat{\vec{P}}_B = G(\vec{r}_B) \in \{\hat{\vec{P}}\}^B$  berechnen

d)  $\hat{\vec{J}}_B = D(\hat{\vec{P}}_B) \in [0, 1]^B$  berechnen

Discriminator-Output

is it check paper  
 $N(0, I)$

CAN paper:  
 $W(0, 0.02)$

Yellow circle = variable benötigt im nächsten Schritt

3. Lossfunktion:

a)  $\mathbb{E}_{\mathbf{x} \sim p_d}[D(\mathbf{x})] = \text{mean}(\vec{J}_B^*)$

b)  $\mathbb{E}_{\mathbf{z} \sim p_z}[D(G(\mathbf{z}))] = \text{mean}(\hat{\vec{J}}_B)$

c) i)  $\hat{\mathbf{x}} = U[0, 1]^B$

ii)  $\tilde{\vec{P}}_B = \hat{\mathbf{x}} \cdot \vec{P}_B^* + (1-\hat{\mathbf{x}}) \cdot \hat{\vec{P}}_B \in [0, 1]^{B \times M}$   
element-wise

iii)  $\tilde{\mathbf{G}}_B := \nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}) = \text{backprop}(\tilde{\vec{P}}_B) \in \mathbb{R}^{B \times M}$   
calc. gradient

iv)  $\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2] = \lambda \cdot \text{mean}((\underbrace{\|\tilde{\mathbf{G}}_B\|_2 - 1}_\text{norm over cutting \mathbf{G} individually})^2)$

$\Rightarrow \mathcal{L} = a) + b) + c)$

↓ weiter

→ Does Loss need to be calculated explicitly in pytorch?

→ Batch normalization?

if yes, how?

4. Backprop