Deep AI in Ableton Live & MaxMSP

From using to creating

Pr. Philippe Esling
esling@ircam.fr

Overall introduction

Goals

1. Understanding the essential theory of (deep) machine learning
2. Being able to both *use existing models and* ***develop your own approach***
3. Training and interacting with these models in MaxMSP / Ableton Live

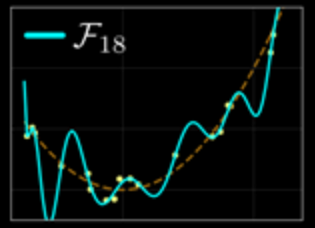
Global program

1. **Creative machine learning**
   Mathematical theory, properties of deep models, using GitHub code
2. **Using and training RAVE and AFTER for Max4Live**
   Using, training and creating with pre-existing models
3. **Embedding any model from Github and Huggingface**
   Developing your own approach to AI composition
4. **Controlling deep models … with deep models**
   How to push the boundaries of generation

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE, AFTER and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

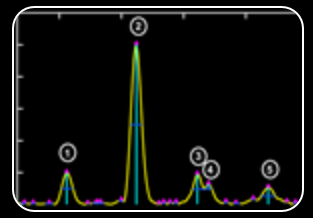
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models ... with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above



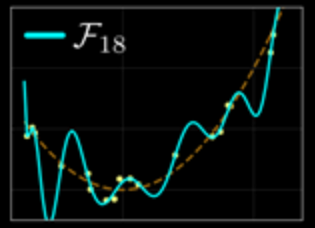
4

The present course is only an introduction to get you started on these complex topics.

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE, AFTER and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

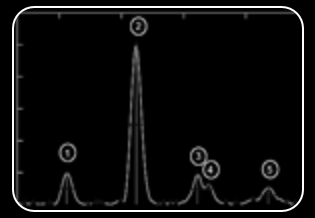
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models ... with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above



4

The present course is only an **introduction to get you started on these complex topics.**

General introduction

Artificial *intelligence* ?

- Requires to first understand what intelligence is !
- Must have something to do with *thinking*
- But in a broader sense, might also be about *perception* and *action*

Why bothering at all ?

- In terms of philosophy, we would talk about problems involving these
- However, we *try to understand the mechanisms of our thinking itself*
- *We seek models* targeted at thinking, perception and actions

Overarching goal of AI

Construct *mathematical models reproducing human behaviors*
Hopefully helping us in understanding our own thinking process

Artificial *intelligence ?*

Hard to define intelligence and thinking

Creation ex-nihilo does not exist
(*Laplace*)

> « *Man can only desire what he has already perceived* »
> **William Blake**

Try to invent a new animal right now

Intelligence: *the capacity to assemble two ideas that seemed heterogeneous*

We exist (morphologically) since ~200 kYears, and around ~50 kYears

> « *Take two concepts and create a third one without impairing the two first* »
> **Noam Chomsky**

So thinking is finding similarity, discriminating and seeing common patterns

NB: Frames of Mind: the Theory of multiple intelligence – **Howard Gardner**
(logico-mathematic, spatial, inter-personal, corporal, linguistic, intra-personnal, musical, ecologist, existential)

To have a model, we will need a *representation* that exposes *constraints* on thinking

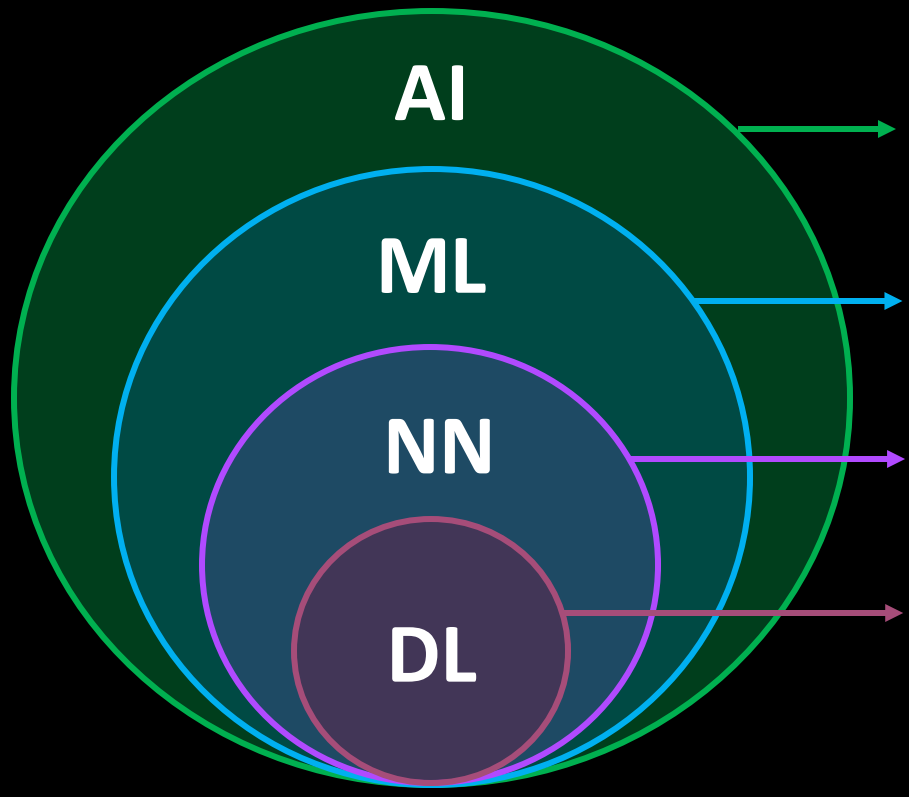
Families of Artificial Intelligence

Solving confusion around machine *intelligence*

There is wide controversies about AI (*weak vs. strong, feasible vs. fake*)

Avoiding any debate, we will only discuss here *machine learning*.

- Can we automatically learn parameters of an approximation
- The most active field recently, and also very wide topics inside it



Artificial Intelligence (AI)
Any technique allowing machines to solve human tasks

Machine Learning (ML)
Learning inference models from examples

Neural Networks (NN)
Brain-inspired ML models

Deep Learning (DL)
Building (deep) hierarchies of NN representations

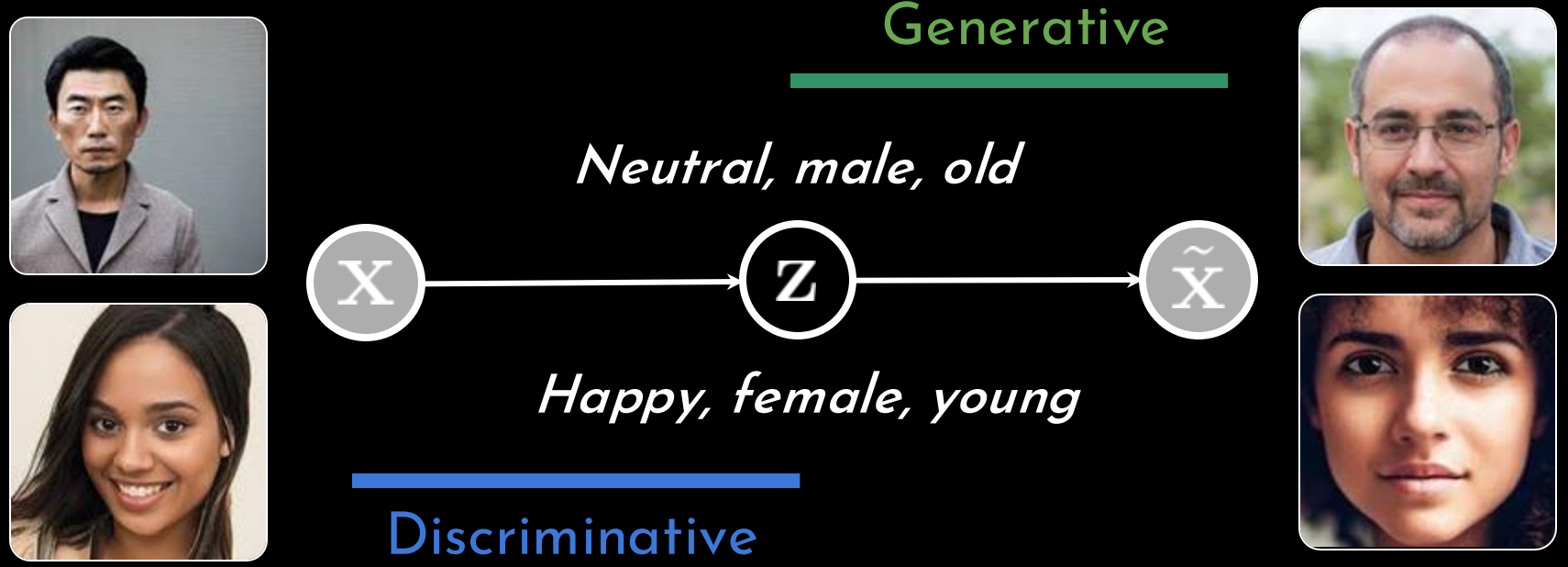
Why creative ?

3 major types of learning

- **Supervised learning** is inferring a function from labeled training data
- **Unsupervised learning** is trying to find hidden structure in unlabeled data
- **Reinforcement learning** is acting to maximize a notion of cumulative reward

Self-supervised learning is using the data itself directly to define tasks to solve

Discriminative or generative ?



Two sides of the same coin

Major players in the generative field

ChatGPT

AI language model (OpenAI) based on Transformers (GPT)
Trained on vast corpus of text to generate chat answers
Impressive question answering and reasoning abilities

Model: Transformers

Midjourney - StableDiffusion

Image-generation AI by Stability based on GANs and diffusion
Trained to generate HQ images based on prompts
Impressive capabilities used in art, entertainment, and fashion

Model: Generative Adversarial Networks (GANs)
(Diffusion models)

What about musical data ?

Creep by Radiohead
... but by The Beatles

Jukebox

Generative model by OpenAI creates original music with lyrics and singing.
Combination of VQ-VAE and Transformer for raw audio at 44.1 kHz.
Can generate music in various styles, and even mimic specific artists

MusicLM

Generative model by Google to perform text-to-music
Based on AudioLM and CLIP for generative audio at 44.1 kHz.
Allows to directly prompt what music you want and generates waveform

A fusion of reggaeton and electronic dance music, with a spacey, otherworldly sound. Induces the experience of being lost in space [...]

Voice cloning

Text-to-speech allows to clone any voice
Cyberduck app

Thank you Barack !

Reverse Karaoke

Generate music to accompany vocals
SingSong project

Course program

How to do all of this by yourself ?

- Ability to construct (math) and develop (code) your own *generative models*
- True and deep understanding of the mathematical theory behind those models
- Constant flow between theory, implementation and creative application

How to understand all

Discriminative

1. Machine learning
2. Neural networks
3. Advanced networks
4. Deep learning
5. Probabilities and Bayesian inference

Generative

6. Latent models, EM and GMM
7. Approximate inference
8. Variational Auto-Encoders (VAEs) and flows
9. Adversarial learning (GANs)
10. Diffusion models

Here comes a quick course summary for those who want to go further

Setup and course follow-up

Full course on Creative ML

github.com/acids-ircam/creative_ml

Github page

Setup guide

For those who want to **truly develop their own models**

- Highly recommended to follow one full course of deep machine learning
- Even if you do not understand the whole mathematics behind
- Develop your own intuition of models inner workings

Installing the course

1. **Star the repo** to get updates on new courses and corrections
2. **Fork the repo** to have your own copy (for exercises)
3. **Clone the repo** to your local machine

Pedagogy aspects

Recent years = Huge focus on pedagogical resources

- 2022+ - Extensive new course on ML for creative purposes (Math / code)
- 2024 - New training course offered at IRCAM pedagogy (RAVE, nn~)

Creative Machine Learning - Complete course on deep learning for creation

- Open-source interactive slides heavy on the mathematical aspects
- Online exercises (Google Colab) to develop your own models

https://github.com/acids-ircam/creative_ml

Sets of deep tutorials - Notebooks to understand deep concepts (PyTorch, JAX)

Normalizing flows /pytorch_flows Diffusion models /diffusion_models

---

Deep learning in MaxMSP and AbletonLive @IRCAM Pedagogy

Ongoing research state

Very active GitHub
github.com/acids-ircam/

FlowSynth Synthesizer control with normalizing flows
https://github.com/acids-ircam/flow_synthesizer
https://www.youtube.com/watch?v=UufQwUitBIw

RAVE High-quality (48kHz) deep audio synthesis on CPU
https://github.com/acids-ircam/RAVE
PureData / MaxMSP https://github.com/acids-ircam/nn_tilde

AFTER / NN~ Neural audio processing in MaxMSP
https://github.com/acids-ircam/nn_tilde
Today's presentation focus on extensions of this ecosystem

Mirrored on IRCAM Forum https://forum.ircam.fr/topics/detail/676-ACIDS/

ACIDS Project

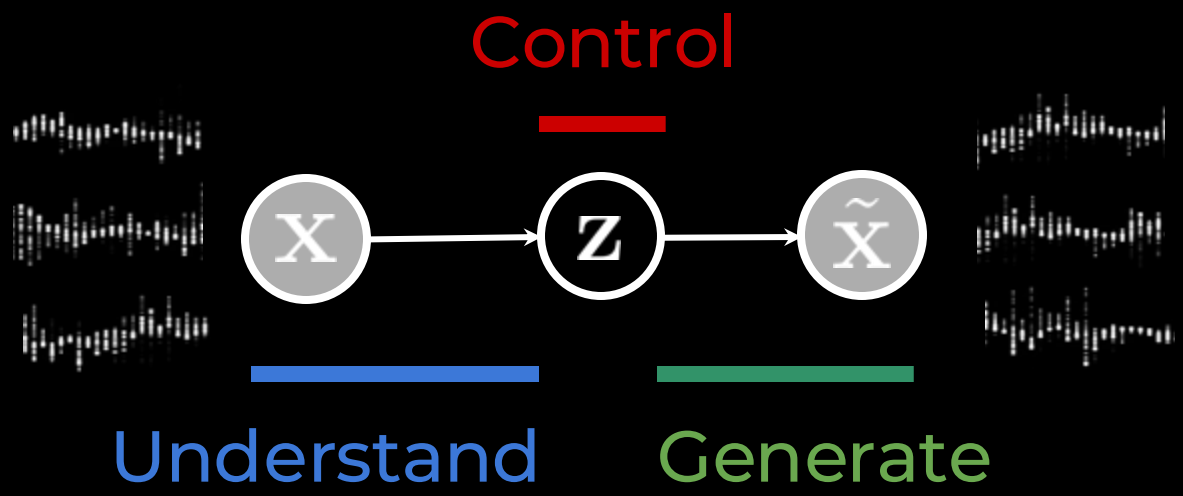
Artificial Creative Intelligence and Data Science

Research label at IRCAM, Sorbonne, Paris - Analysis / Synthesis team

Academic machine learning research since 2013

Theoretical and applied research in generative models for musical creation

General approach



Open-source usable outputs

- [**FlowSynth**] - Wav to preset
- [**VSChaos**] - Vintage neural audio
- [**RAVE**] - Real-time synthesis
- [**NN~**] - Neural audio in MaxMSP
- [**AFTER**] - Streamable diffusion
- [**TorchBend**] - Neural bending
- [**RaveTable**] - Latent wavetables

[Theory time]

Link to the course given at Todai and IRCAM

Github page

github.com/esling/creative_ml

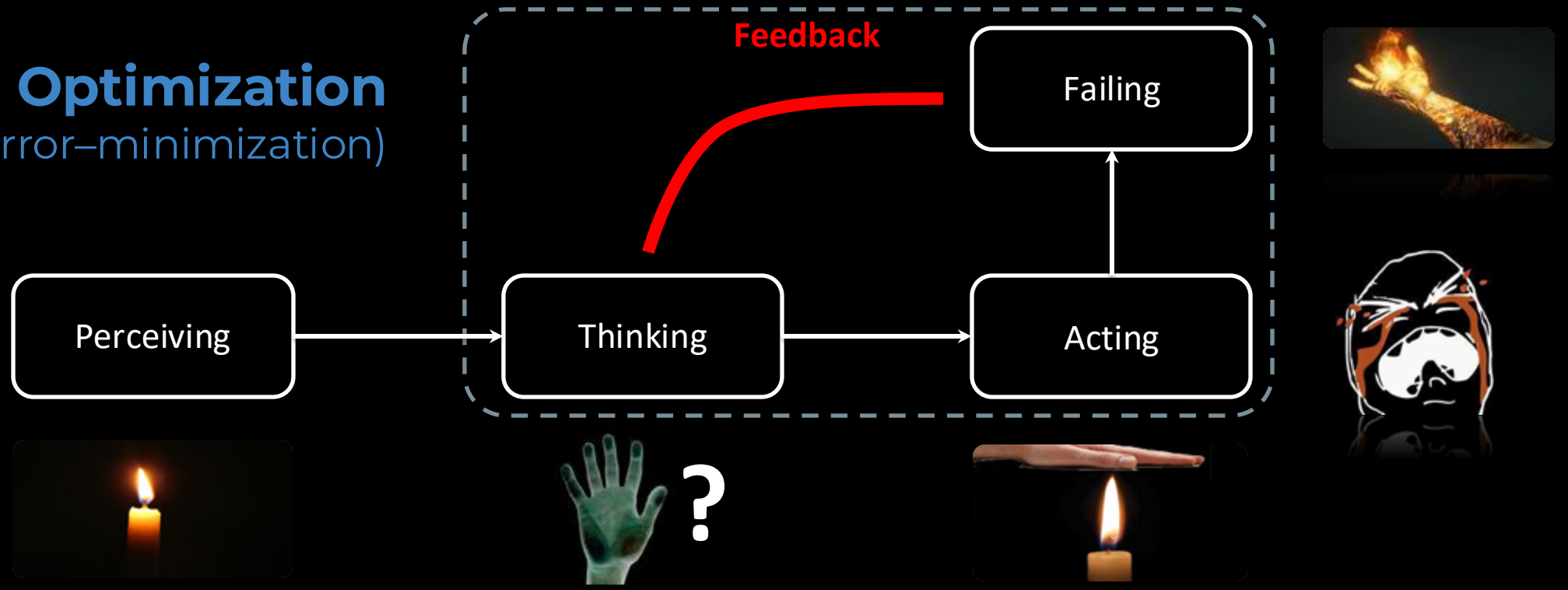
Installation guide

creative_ml#setup

Learning as error minimization

Why it makes sense in real-life

Optimization
(error–minimization)



- **Goal-directed learning** = acquiring, modifying or reinforcing knowledge
- Most problems can be summarized by a **trial-error process**
- Is a nickname for **optimization** (through **error minimization**)
  - So what can we do with ML ? Some examples from our team (ACIDS)

Machine learning

Problem statement

We witness a phenomenon
Through a set of observations

$$(x_1, y_1)\ (x_2, y_2)\ \cdots\ (x_n, y_n)$$

We know there exists a relation between $\mathcal{X}$ and $\mathcal{Y}$

But we do not know its nature

$$f : \mathcal{X} \to \mathcal{Y}$$

Machine learning constructs approximations

$$f_\theta : \mathcal{X} \to \mathcal{Y}$$

such that

$$y_i \approx \bar{y}_i = f_\theta(x_i)$$

Regression and classification

The two most classical machine learning problems

Machine learning

Formal problem statement

We aim to model the relationship between $\mathcal{X}$ and $\mathcal{Y}$

$$f : \mathcal{X} \to \mathcal{Y}$$

We collect a **dataset** that is representative of that relation

$$\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1) \cdots (\mathbf{x}_n, \mathbf{y}_n)\}$$

We will construct a **parametric approximation** $f_\theta \in \mathcal{F}_\Theta$

$$f_\theta : \mathcal{X} \to \mathcal{Y} \qquad \bar{\mathbf{y}} = f_\theta(\mathbf{x})$$

We want to find the best f_θ^* so that $\bar{\mathbf{y}} \approx \mathbf{y}$

So we need to find the best parameters $\theta^* \in \Theta$

Need to compute the errors (**loss**) of our model

$$\mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$$

And minimize (**optimize**) this amount of errors

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ \mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$$

Choice of approximation family (model)

Quality depends heavily on family $\mathcal{F}_\theta$

Notion of *model capacity*

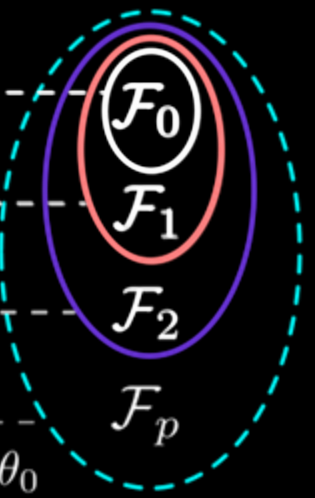
Approximation families

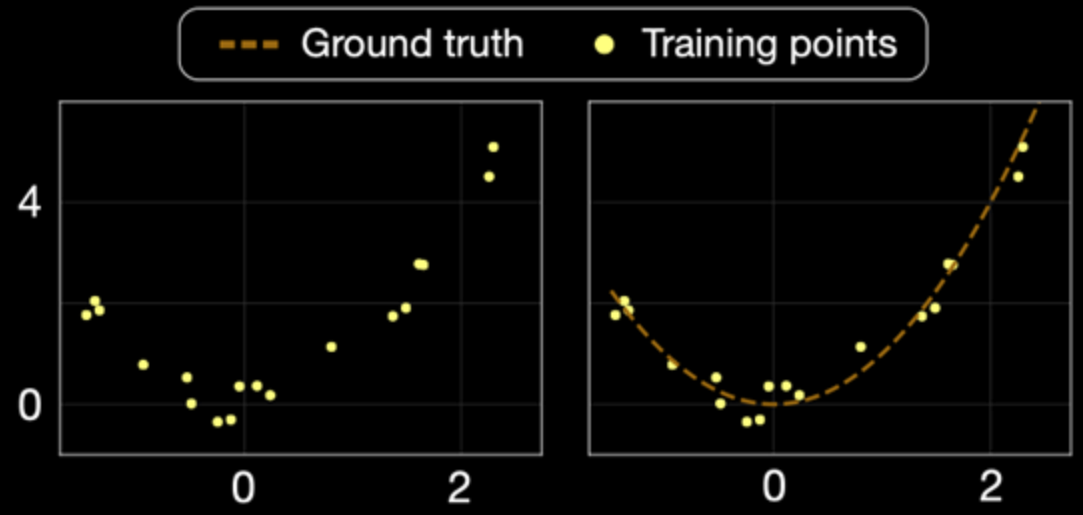
Constant
$\mathcal{F}_0(x) = \theta_0$

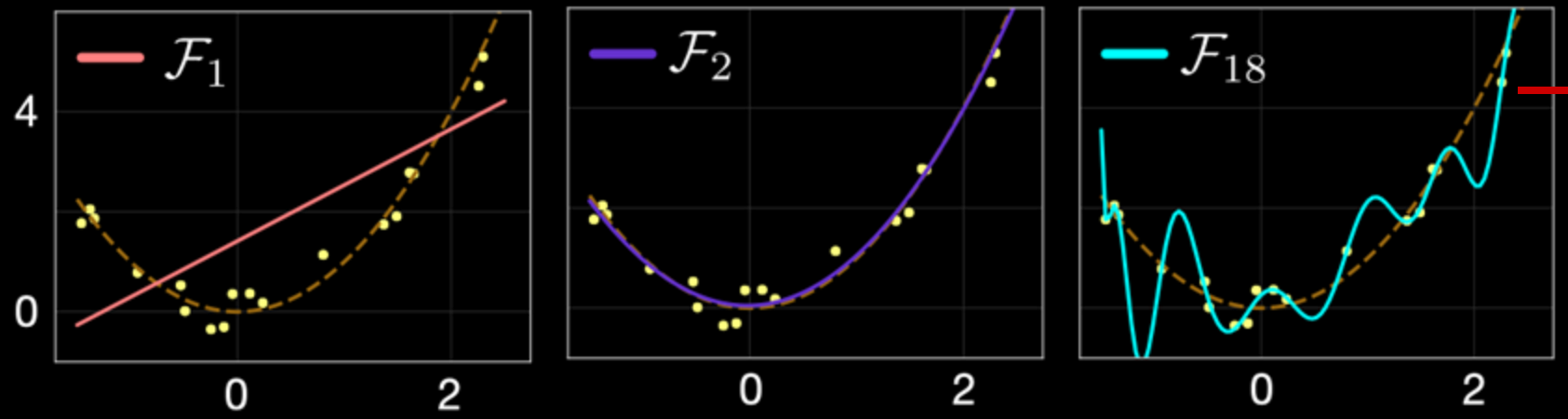
Linear
$\mathcal{F}_1(x) = \theta_1 x + \theta_0$

Square
$\mathcal{F}_2(x) = \theta_2 x^2 + \theta_1 x + \theta_0$

Polynomial
$\mathcal{F}_p(x) = \theta_p x^p + \cdots + \theta_1 x + \theta_0$







Overfitting

("Because it can")

We will see how we can solve this through **regularization**

Linear regression

Supervised learning algorithm

Model the relationship between **one** output (target) and one or more inputs (features).

$$\mathbf{x} \in \mathbb{R}^n \qquad \mathcal{X} \longrightarrow f_\theta \longrightarrow \mathcal{Y} \qquad y \in \mathbb{R}$$

Input (feature) *Output (target)*

Regression

« Find an approximation that gives an answer in terms of continuous values »

Linear regression

Supervised learning algorithm

Model the relationship between **one** output (target) and one or more inputs (features).

$$\mathbf{x} \in \mathbb{R}^n \qquad \mathcal{X} \xrightarrow{f_\theta} \mathcal{Y} \qquad y \in \mathbb{R}$$

Input (feature) *Output (target)*

Goal Find best-fitting *hyperplane* (*line*) to predict output.

Equation $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + \epsilon$

- y : Output (*target*)
- x_i : Input (*feature*)
- w_i : Coefficients (*weights*)
- w_0 : Intercept (*bias* term)
- ϵ : Residual error

~~**A note on the residual error**~~ ϵ ~~Modeling the observation noise always present~~

Model $f_\theta(x) = \bar{y} = w_0 + w_1 x_1 + \cdots + w_n x_n$

Modify parameters $\theta = \{w_0, \cdots, w_n\}$ so that $y \approx \bar{y}$

Best-fitting line ? *Minimize differences between real and predicted output*

Various application in inference and forecasting.

Key elements of our problem definition

Dataset

$$\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1) \cdots (\mathbf{x}_n, \mathbf{y}_n)\}$$

The **dataset** has to be **representative** of the relation $f : \mathcal{X} \to \mathcal{Y}$

Model

$$f_\theta : \mathcal{X} \to \mathcal{Y} \qquad \bar{\mathbf{y}} = f_\theta(\mathbf{x})$$

The choice of **parametric family** $f_\theta \in \mathcal{F}_\Theta$ is critical

Loss

$$\mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$$

How do we measure the errors (**loss**) of our model

Optimization

Search for best $\theta^* \in \Theta$ so f_{θ^*} minimizes the loss ($\bar{\mathbf{y}} \approx \mathbf{y}$)

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ \mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$$

How to choose the parametric family ?

Simple linear regression

Principle

Linear regression with a **single input** variable.
Model relationship of *single* input and output (*target*).

$x \in \mathbb{R} \longrightarrow y \in \mathbb{R}$

Examples: *Predicting house prices based on sizes, predicting temperature based on year*

Model $\bar{y} = w_0 + w_1 x \quad \theta = \{w_0, w_1\}$

Based on our collected dataset of examples $\mathcal{D} = \{(x_1, y_1), \cdots, (x_n, y_n)\}$

We can compute the errors made by our model **on each example**

$$\epsilon_i = |y_i - \bar{y}_i| = |y_i - (w_0 + w_1 x_i)| \quad \text{—— } L_1 \text{ loss}$$

L_2 loss function
Mean-Squared Error (MSE)

$$\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^{n} |y_i - \bar{y}_i|^2 = \sum_{i=1}^{n} |y_i - (w_0 + w_1 x_i)|^2$$

whole dataset

—— **More sensitive to outliers (square)**

Goal $\underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\bar{\mathbf{y}}, \theta)$ Minimize the MSE based on parameters $\theta = \{w_0, w_1\}$

Gradient descent

Intuition How to find the parameters that minimize our loss function ?

$$\min_{\theta} \mathcal{L}(\bar{\mathbf{y}}, \theta) \longrightarrow \frac{\partial \mathcal{L}}{\partial \theta} = 0$$

A minimum exists where the gradient of our function is null

Problems

1. Explicit solution ? Might **require prohibitive computation**
2. Finding minima ? There might be **multiple local minima**
3. **Gradient can change** widely depending on the examples

Gradient descent Optimization algorithm to minimize the loss iteratively.
Compute partial derivatives of the loss w.r.t each parameter.

$$\nabla \mathcal{L}(\mathbf{w}) = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \cdots, \frac{\partial \mathcal{L}}{\partial w_n} \right]$$

Represents the **direction of the steepest increase** of the loss function.
Adjusting parameters in that direction is the steepest decrease of the loss

Gradient descent

Algorithm

1 - Initialize parameters

Start from random point $\mathbf{w} \sim \mathcal{N}(\mu_0, .01)$

2 - Compute partial derivatives

Derive the loss based on all parameters

$$\nabla \mathcal{L}(\mathbf{w}) = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, ..., \frac{\partial \mathcal{L}}{\partial w_n} \right]$$

3 - Iterative updates

In each iteration update the weights

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla \mathcal{L}(\mathbf{w}^t)$$

Move in the opposite direction of the gradient

Learning rate η controls the step size of the updates.

4 - Convergence

Stopping criteria

- $\mathcal{L}(\mathbf{w}) \to 0$
- $\nabla \mathcal{L}(\mathbf{w}) \to 0$
- $t > n_{epochs}$

Understanding the learning rate

Gradient descent optimization

$$\nabla \mathcal{L}(\mathbf{w}) = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_n}\right]$$

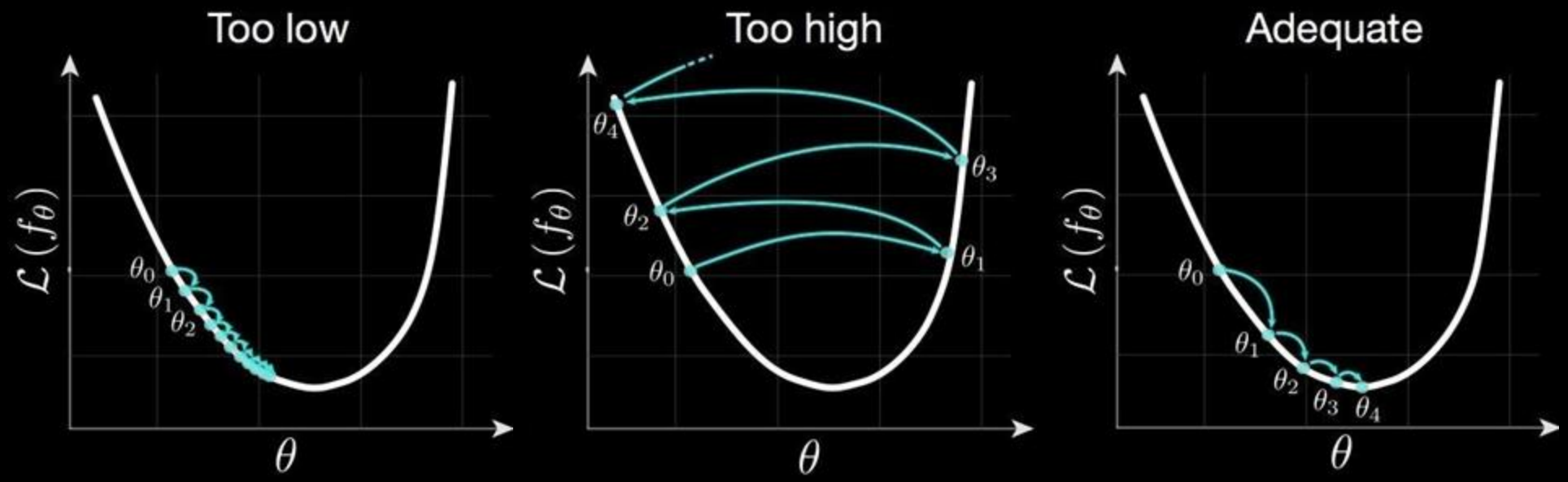
In our previous update equation

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla \mathcal{L}(\mathbf{w}^t)$$

Learning rate η

Impact of the learning rate

Problem is that we do not know the **magnitude** of the gradient



Slow convergence

Divergence

Applying gradient descent for regression

Goal

We want to minimize our loss $\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^{n} (y_i - \bar{y}_i)^2 = \sum_{i=1}^{n} (y_i - (w_1 x_i + w_0))^2$

Compute $\nabla \mathcal{L}(\mathbf{w}) = \left[\frac{\partial \mathcal{L}}{\partial w_0}, \frac{\partial \mathcal{L}}{\partial w_1}\right]$

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial}{\partial w_0}\left(\sum_{i=1}^{N}(y_i - (w_1 x_i + w_0))^2\right) = -2\sum_{i=1}^{N}(y_i - (w_1 x_i + w_0))$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial}{\partial w_1}\left(\sum_{i=1}^{N}(y_i - (w_1 x_i + w_0))^2\right) = -2\sum_{i=1}^{N} x_i(y_i - (w_1 x_i + w_0))$$

Complete algorithm

Initialize parameters $w_1^0, w_0^0 \sim \mathcal{N}(0, 1)$

Loop

1. Compute our current **approximation** $\bar{y} = w_1^t x + w_0^t$
3. Compute **gradients** of the loss function
2. Compute the current value of the **loss function** $\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^{n} \left(y_i - (w_1^t x_i + w_0^t)\right)^2$

$$\nabla \mathcal{L}(\mathbf{w}) = \left[\frac{\partial \mathcal{L}}{\partial w_0^t}, \frac{\partial \mathcal{L}}{\partial w_1^t}\right] = \left[-2\sum_{i=1}^{N}(y_i - (w_1^t x_i + w_0^t)), -2\sum_{i=1}^{N} x_i(y_i - (w_1^t x_i + w_0^t))\right]$$

4. **Update our parameters** based on the gradients

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla \mathcal{L}(\mathbf{w}^t)$$

Gradient descent variants

So far discussed **batch gradient descent** - compute gradient for the entire dataset

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{N} \sum_i \nabla \mathcal{L}_i(\mathbf{w}_t)$$

Can be slow (or even impossible) for large datasets.

Variants can improve computational efficiency and even convergence properties.

Stochastic GD

Computes gradient with **single data point** each time
Faster than batch-GD, but high variance in updates

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \mathcal{L}_i(\mathbf{w}_t)$$

Mini-batch GD

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{b} \sum_{i=1}^{b} \nabla \mathcal{L}_i(\mathbf{w}_t)$$

Use a **random subset (mini-batch)** of the dataset.
Trade-off for balancing speed and variance

Adaptive Moment Estimation (ADAM)

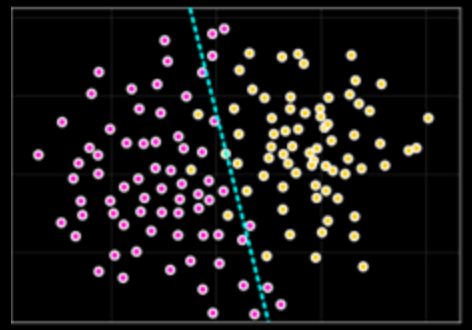
(cf. deep learning)

ADAM adapts the learning rate for each parameter

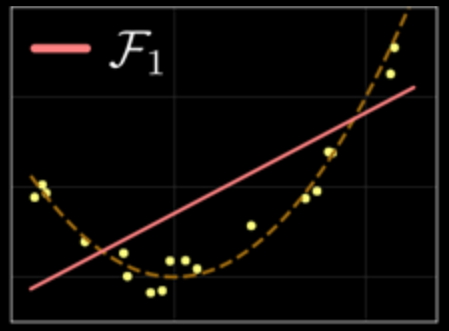
Maintains decaying average of past and squared gradients.
More robust and efficient than standard Gradient Descent.

Understanding capacity

Classification

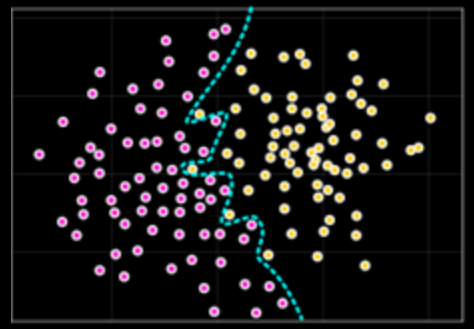


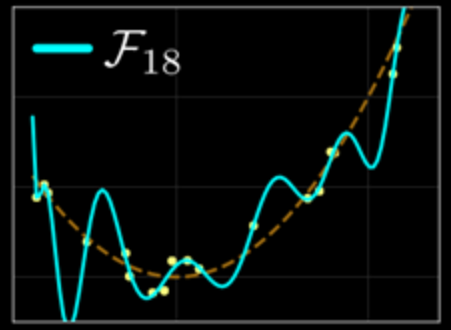
Regression



Underfitting

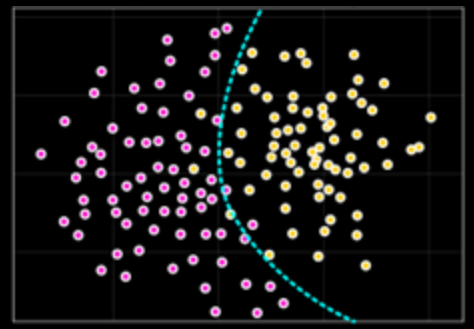
- Function family is **too poor**
- Capacity is **too low** for this problem

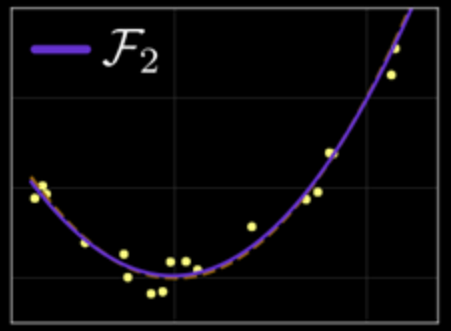




Overfitting

- Function family is **too rich**
- Capacity is **too high** for this problem



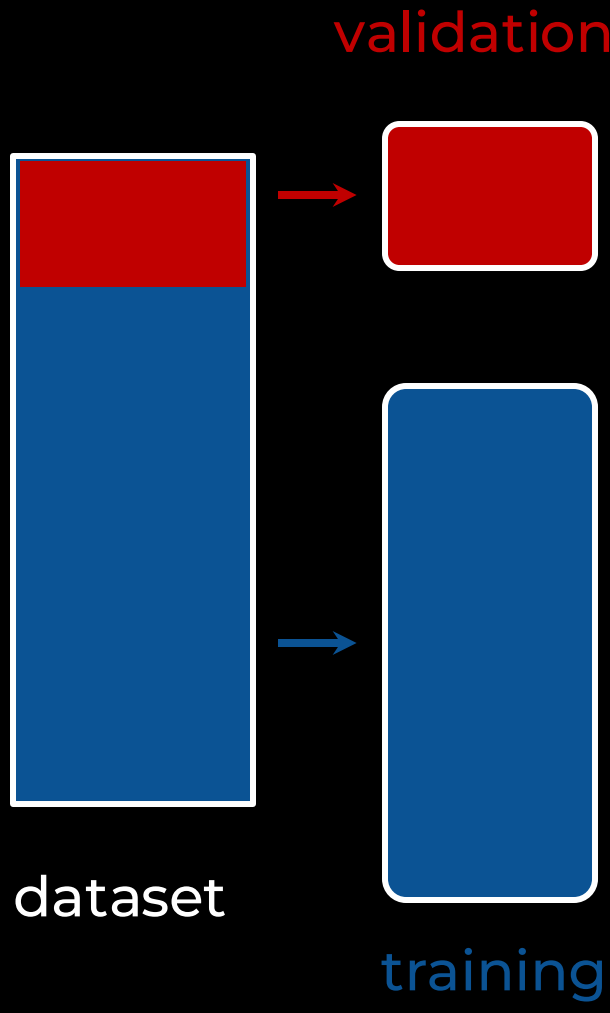


Optimal capacity

- Function family is **adequate**
- **Optimal capacity** for this problem

Cross-validation for overfitting

How do we aim to avoid overfitting ?



Split your complete dataset into two distinct set

- **Training** set (~80%) for learning your model
- **Validation** set (~20%) for evaluating overfitting

The validation set emulates an unknown dataset

Allows to evaluate the quality on unseen data

Final performance evaluation

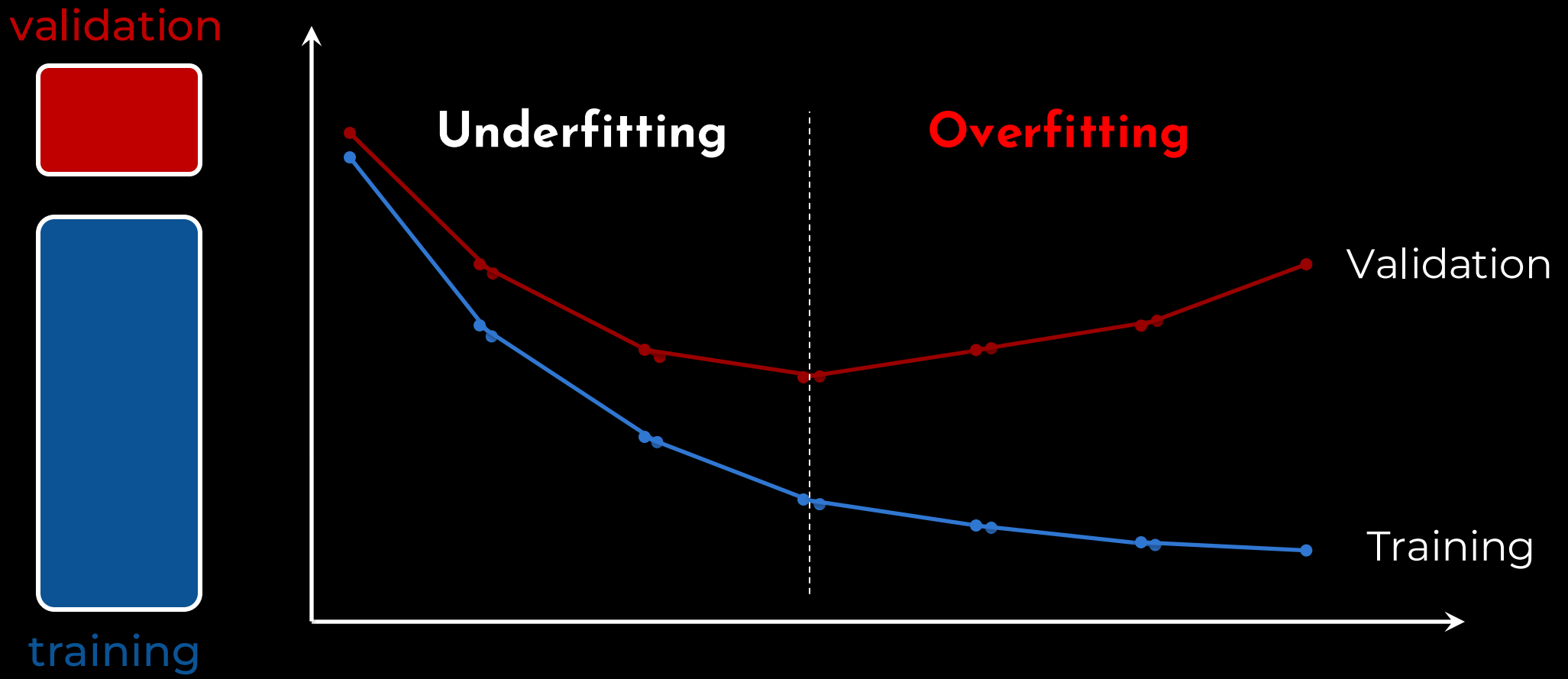
Performed on a *separate independent dataset*

Ideally this set is *out-of-distribution (o.o.d)* to prevent bias

Early stopping

Stop the learning when the error increases on the validation set

(Approximates the generalization power)



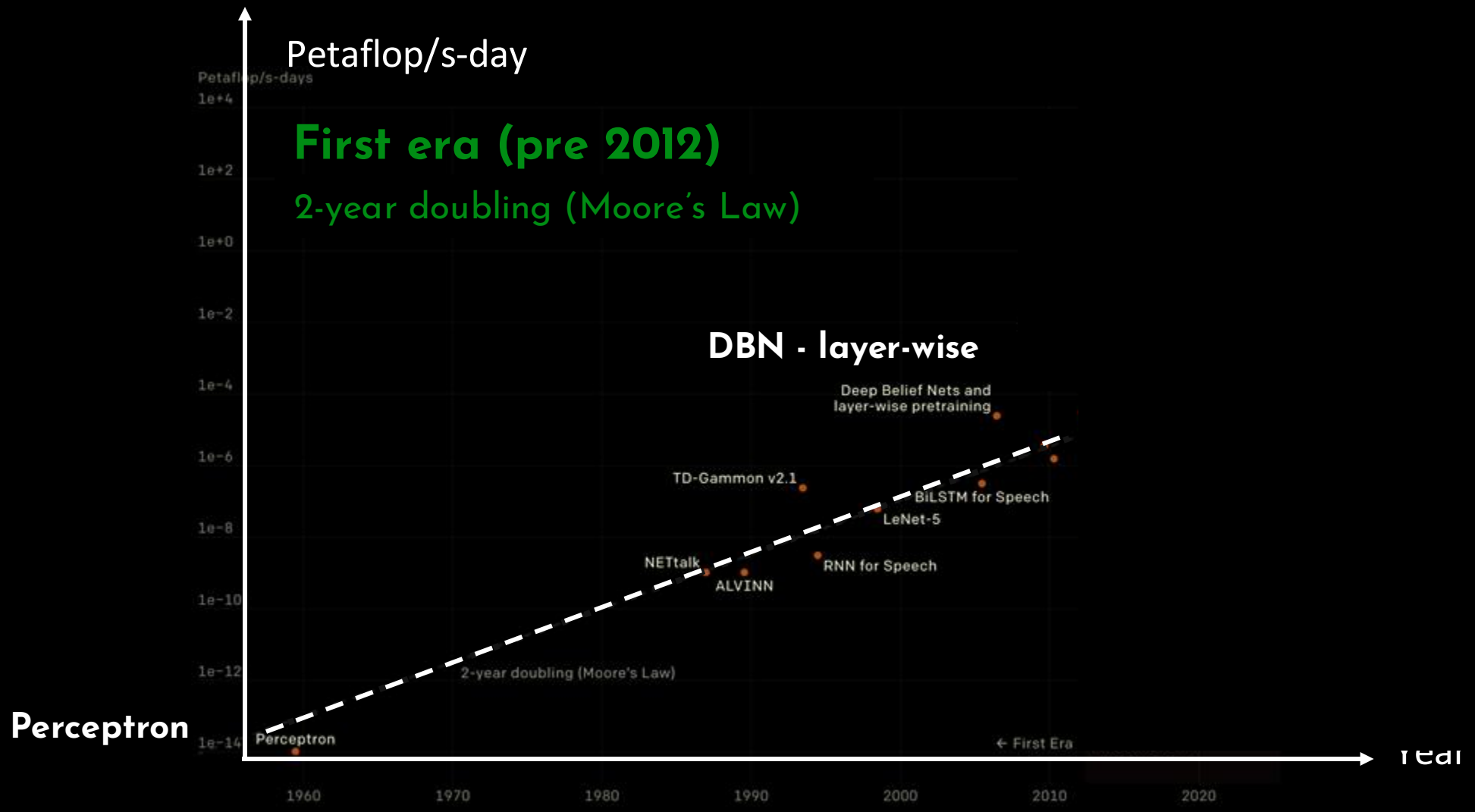
Scaling laws and data

How data naturally regularizes

Eras in machine learning

Two distinct eras in ML

Dario Amodei and Danny Hernandez. AI and compute (2018)



Consequences and observations

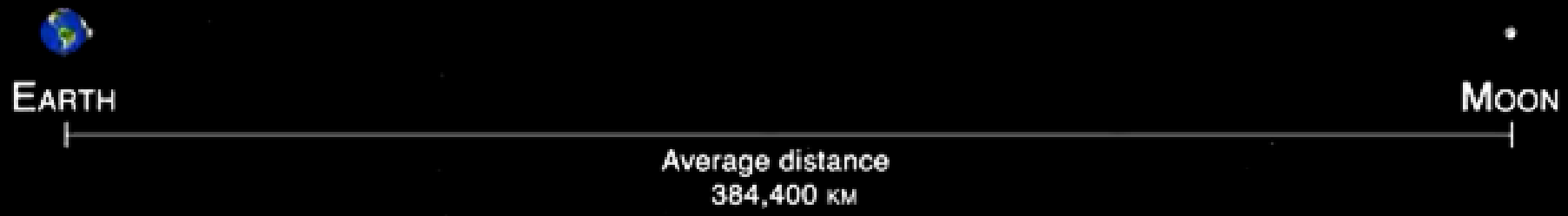
Direct consequences of this accuracy race

- **Huge environmental issues**
- **Precludes the use in non-specialized (user-side) hardware**
- **Even less possible to embed such systems**

GPT-3

175 billon parameters, *takes 355 years on a single GPU to train*

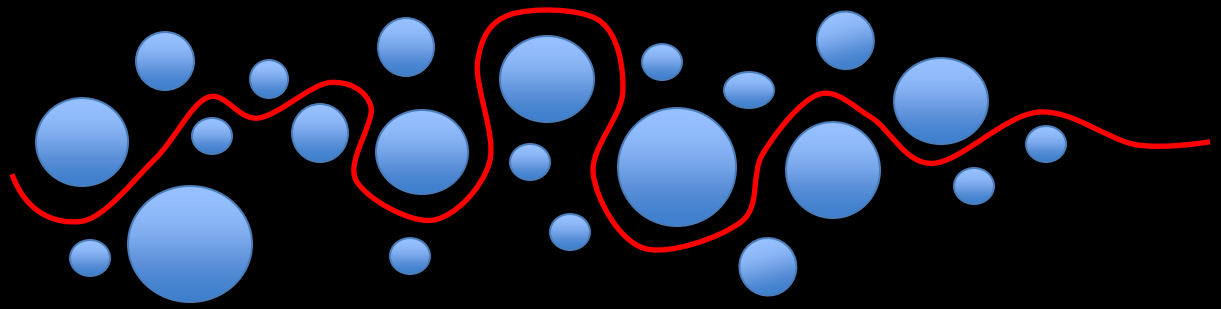
Carbon footprint for training equivalent to driving to the moon and back [3]



Dr. James O'Donoghue using NASA imagery.

Observation #1 - Complexity of the problem

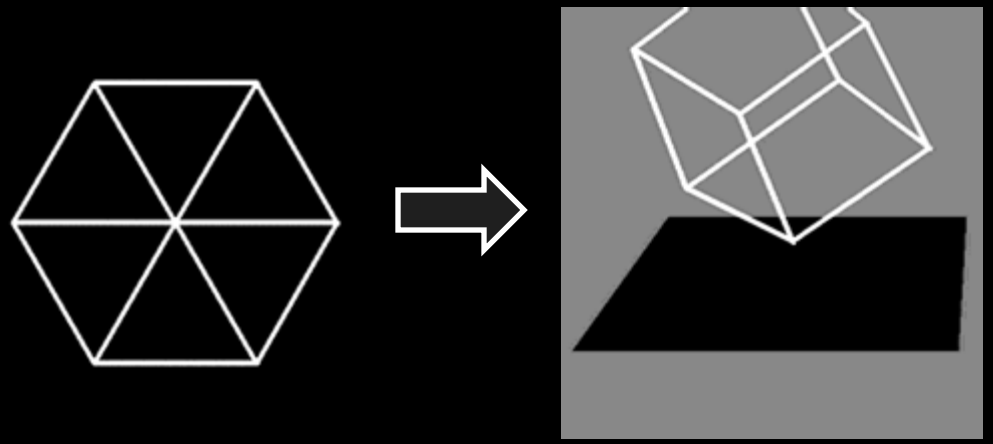
Imagine you observe the path of an ant



Seems complicated … you could say the ant is extremely smart
You look closer, there are rocks that the ant is just avoiding

Complexity can be consequence of the environment not the program

Now imagine you are trying to understand this behavior

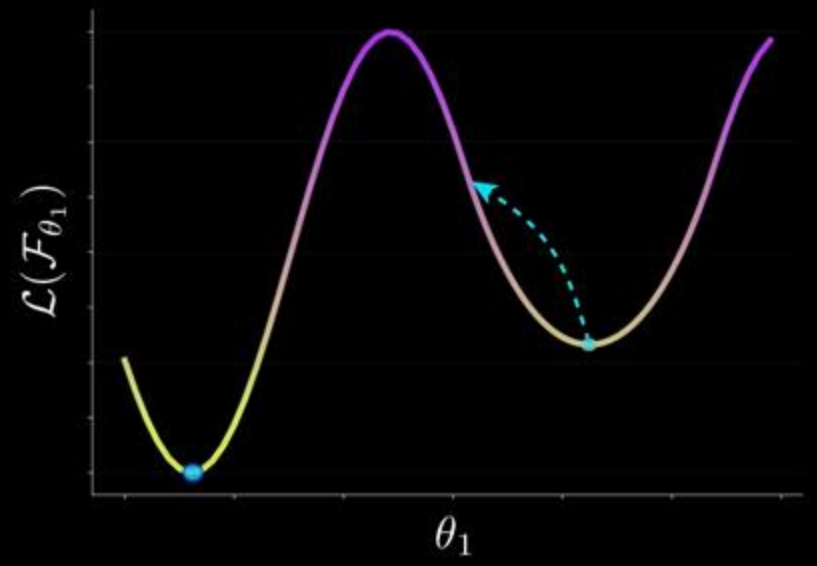


Notion of problem dimensionality

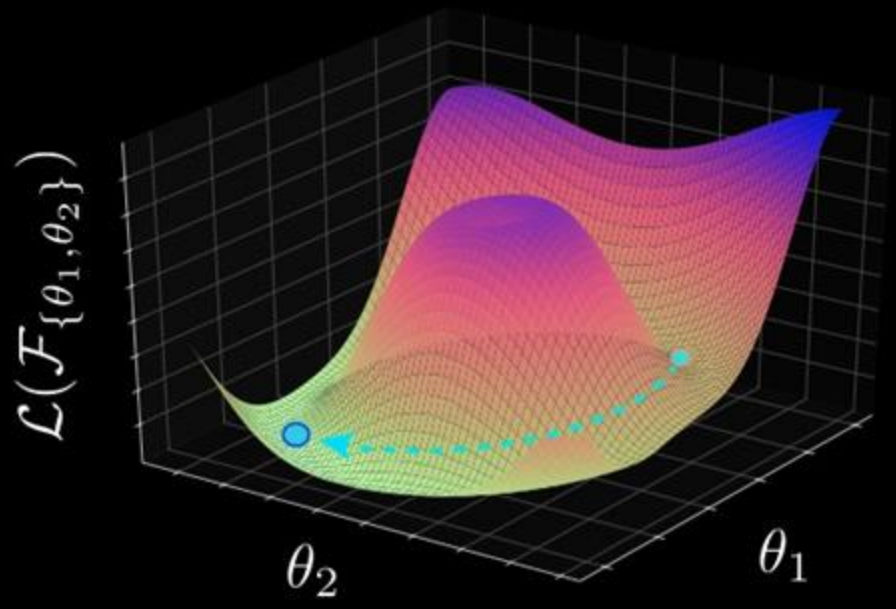
Overparametrization

One potential explanation on why overparameterization works

Imagine the loss landscape (error given parameter values)



With a single parameter, hard to escape minima



Extra parameter, we can « go around » (idealized)

Overparametrization **simplifies the training** … allowing to find a **better solution faster**

Does not mean that the final solution *needs* all these parameters

[10 minutes break]

Link to the course given at Todai and IRCAM

Github page

github.com/esling/creative_ml

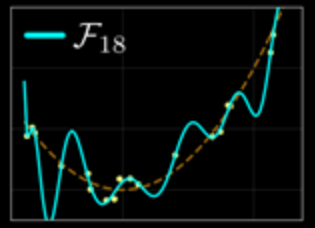
Installation guide

creative_ml#setup

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE, AFTER and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

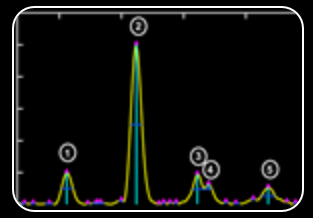
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models … with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above



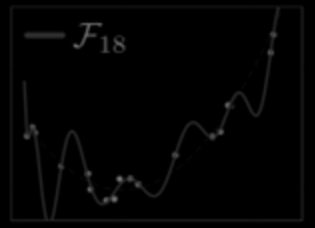
4

The present course is only an **introduction to get you started on these complex topics.**

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE, AFTER and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

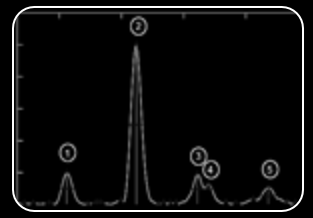
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models ... with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above



4

The present course is only an **introduction to get you started on these complex topics.**

Using existing externals (MaxMSP)

Introducing the **nn~ external** (and RAVE model)

acids-ircam / nn_tilde

nn_tilde Public

https://github.com/acids-ircam/nn_tilde

How to install it

Installation

Grab the latest release of nn~. Be sure to download the correct version for your installation.

▾ Assets 8

nn_max_msp_macOS_arm64.tar.gz	116 MB	Jun 23
nn_max_msp_macOS_x64.tar.gz	339 MB	Jun 23
nn_max_msp_windows_x64.tar.gz	148 MB	Jun 23

Installing nn~

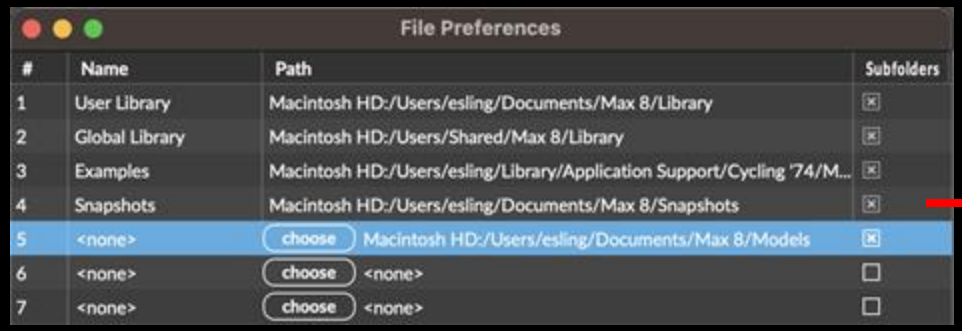
Getting models

Pretrained models

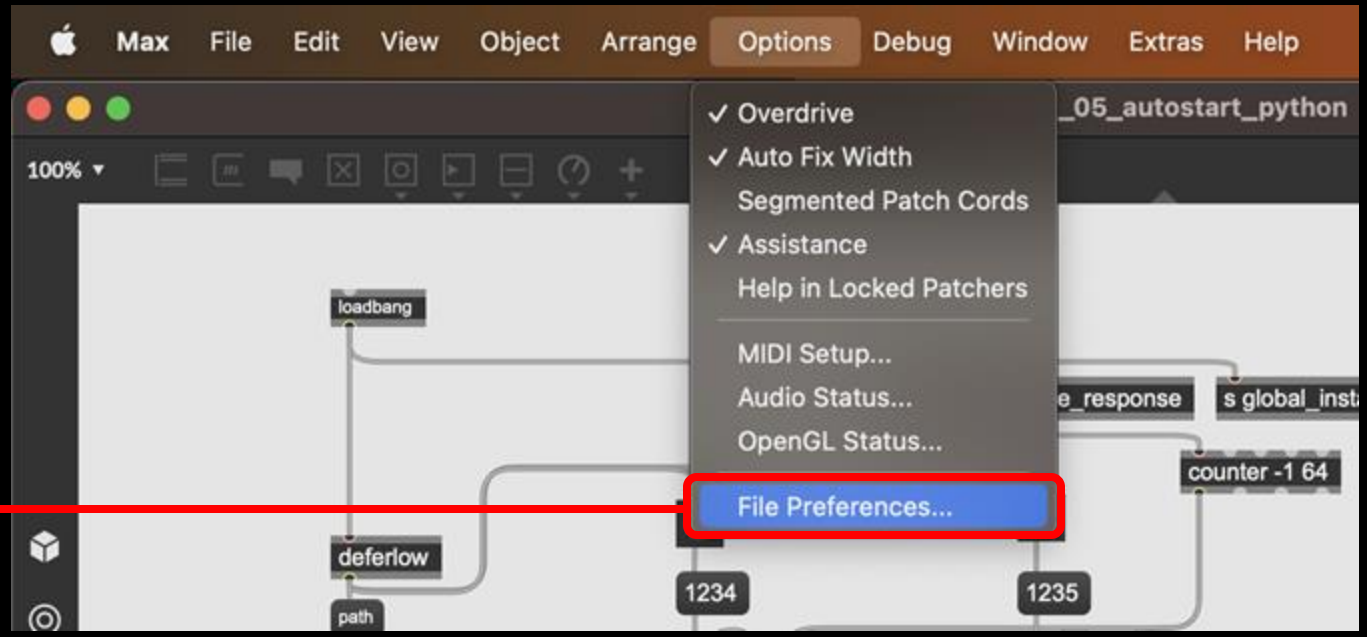
At its core, `nn~` is a translation layer between Max/MSP or PureData and the libtorch c++ interface for deep learning. Alone, `nn~` is like an empty shell, and **requires pretrained models** to operate. You can find a few RAVE models here, or a few vschaos2 models here.

Rave models download page

Model Name	Version	Model	Release Date	Author
Vintage	1	RAVE V1 - Large	21/09/2022	Antoine Caillon
Percussion	1	RAVE V1 - Default	21/09/2022	Antoine Caillon
Nasa	1	RAVE V1 - Default	21/09/2022	Antoine Caillon
Darbouka_onnx	1	RAVE V2 - Onnx	21/09/2022	Antoine Caillon
VCTK	1	RAVE V1 - Default	11/05/2022	Jb Dupuy

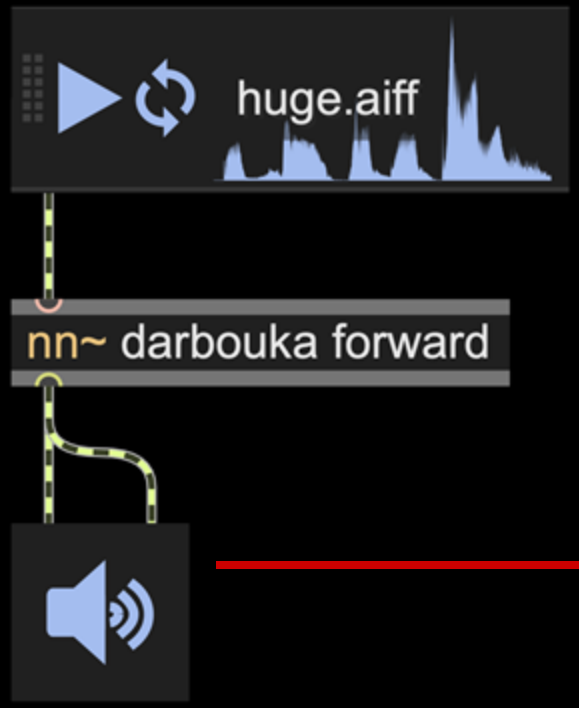


Adding models directory to the path



Audio timbre transfer (nn~)

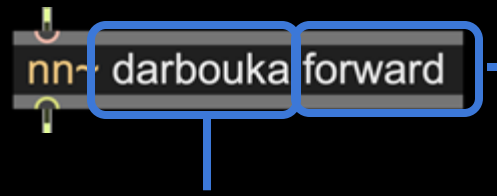
Direct usage of **audio timbre transfer**



Input signal

Transferred audio

Model arguments



Type of operation

- forward
- encode
- decode
- prior

Model name

- Needs to be downloaded
- Add it to the pretrained models directory
- Define the **type of output sound**

Handling the buffer size (real-time latency)

In order to reduce latency you can change buffer size
Use 0 to maximally reduce latency (performance)

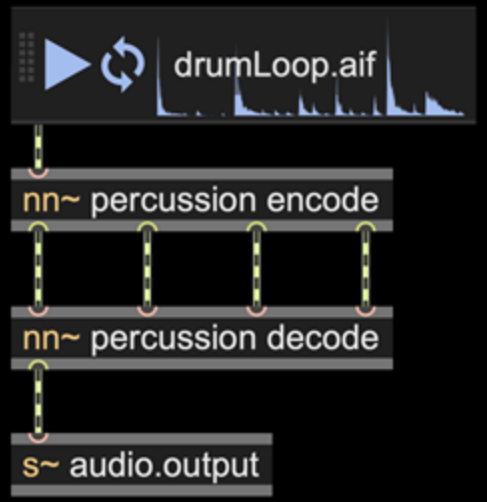
Different nn~ operations

Different types of operations

Forward

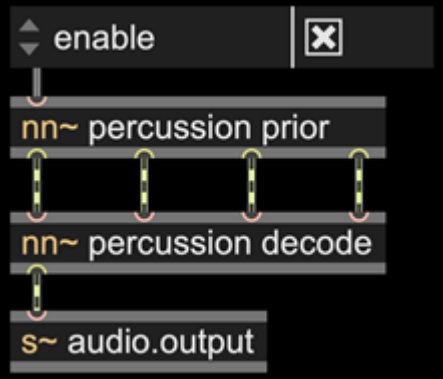
==

Encode / decode



Infinite generation

Prior (infinite) generation

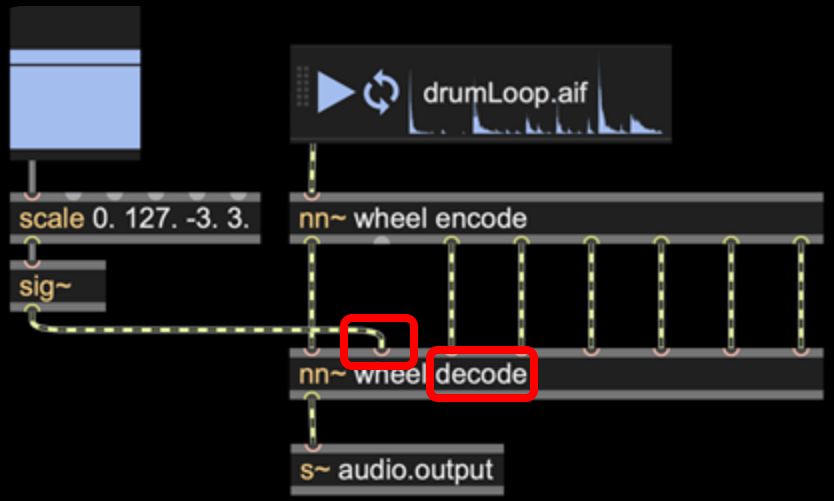


Generate signal infinitely without inputs

Method	Description	Inputs	Outputs
encode	Encodes an audio signal into a latent trajectory	audio signal	latent trajectories
decode	Decodes a latent trajectory into an audio signal	latent trajectories	audio signal
forward	Encodes *and* decodes an audio signal	audio signal	audio signal

Advanced dimension control (nn~)

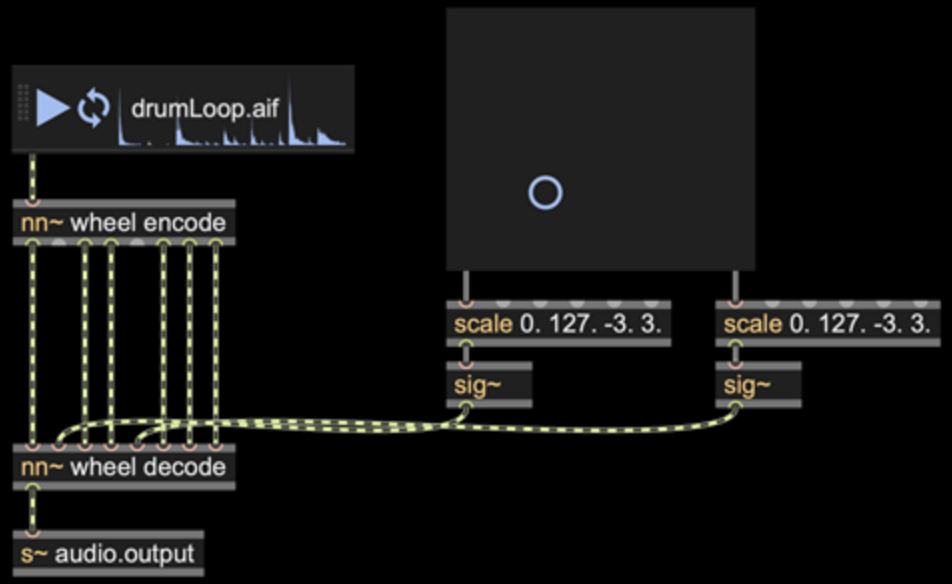
Controling dimensions of generative transfer



- All dimensions are exposed in the **decode** process
- Hence, we can mix different sources of control
  - Part of the dimensions from **encoded sound**
  - Others from **direct manual control**

Multiple dimensions control

Of course this can be extended to any dimension
Also any type of control signal input



Options for nn~

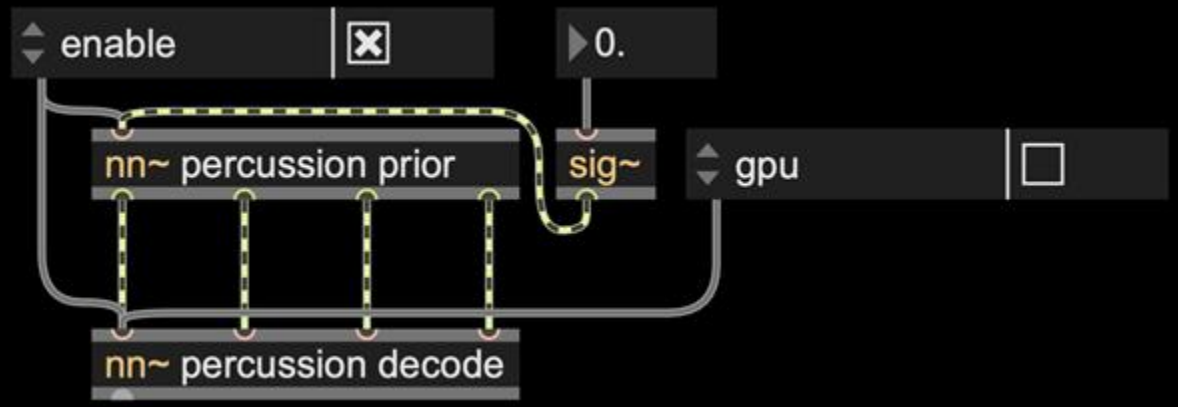
Options and messages for the external

Disable computation

Reduce computational load (save CPU)

Generation temperature

Controls randomness **of prior**



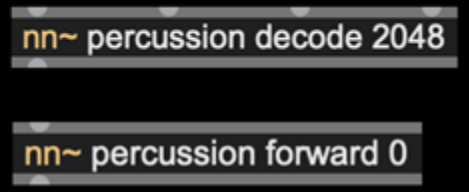
GPU Computation

Only on NVIDIA cards

Handling the buffer size (real-time latency)

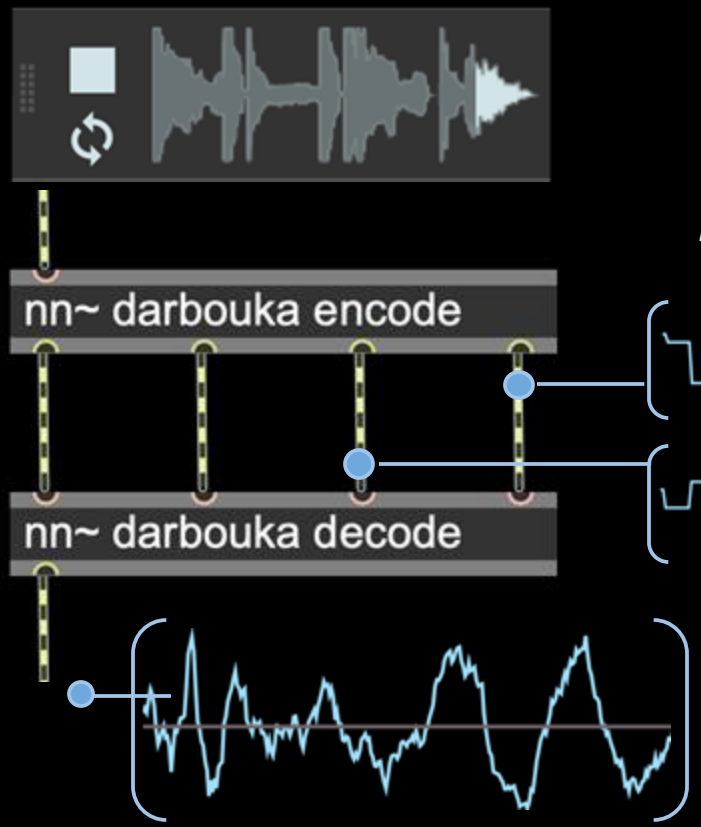
In order to reduce latency you can change buffer size

Use 0 to maximally reduce latency (performance)



Advanced uses of nn~

Case 1 - Latent signal processing



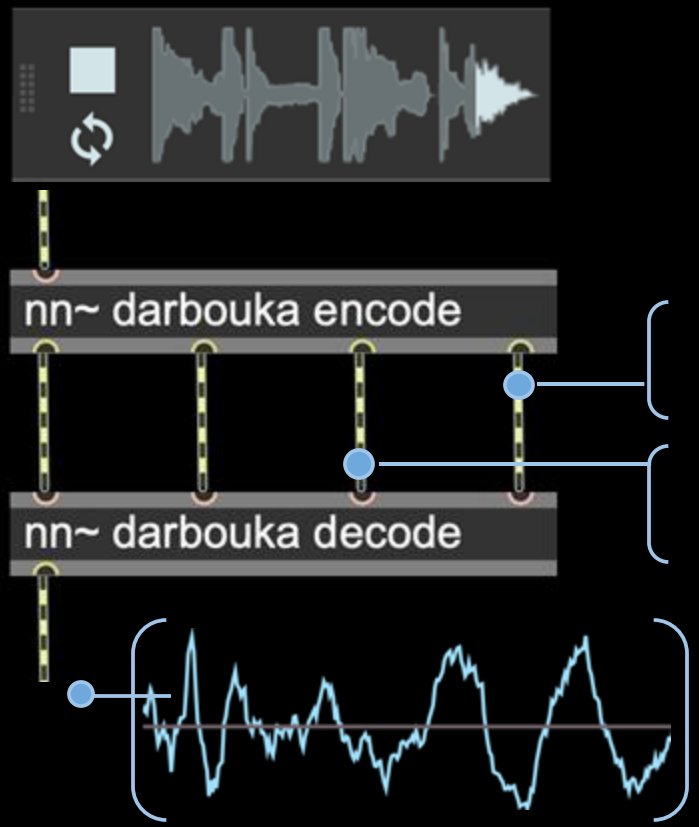
Latent variables as signals

- Outputs of the encoder are **signals**
- We can apply any operation there
- Most usual operations are
  - Multiplication (*scale*)
  - Addition (*bias*)
- But we can apply any DSP here
- Just have to think about the **meaning**
- For instance filtering is on **event speed**

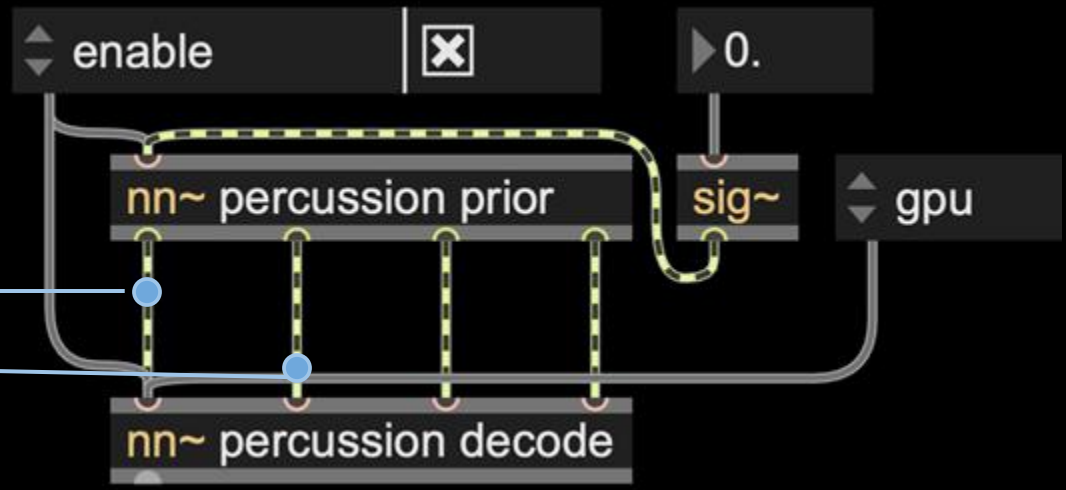
Can you think of other operations ?

Advanced uses of nn~

Case 2 - Prior signal mixing



?



How to mix those two types of signal

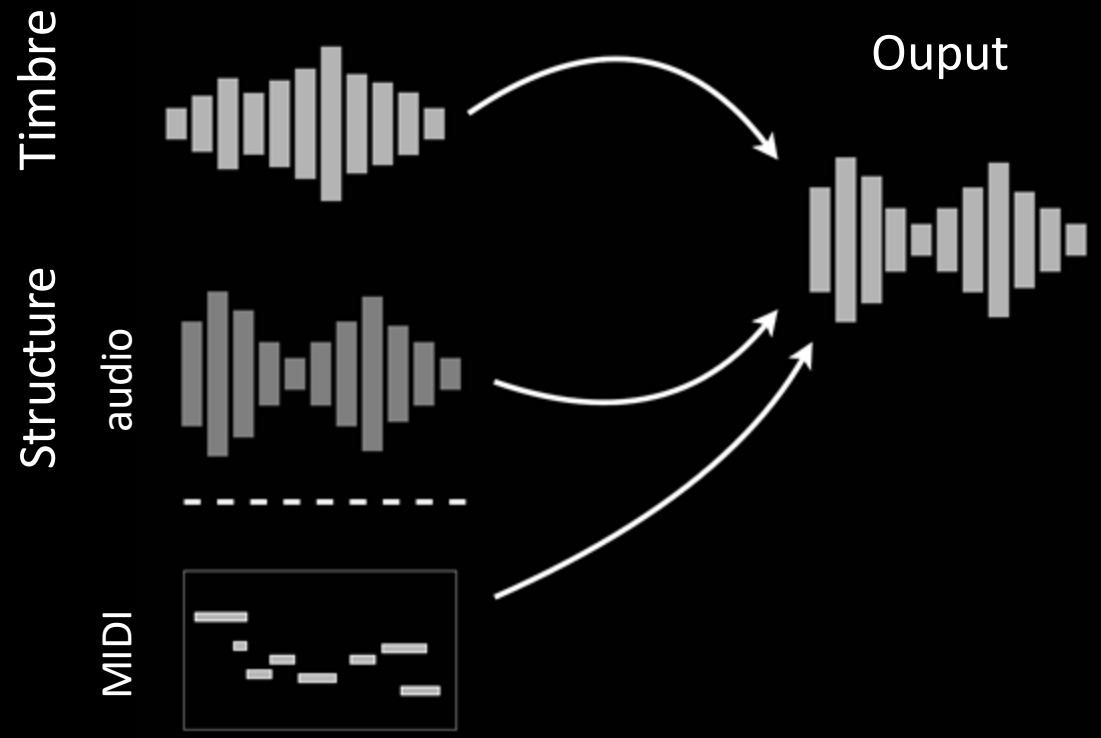
AFTER : Diffusion models for audio transfer

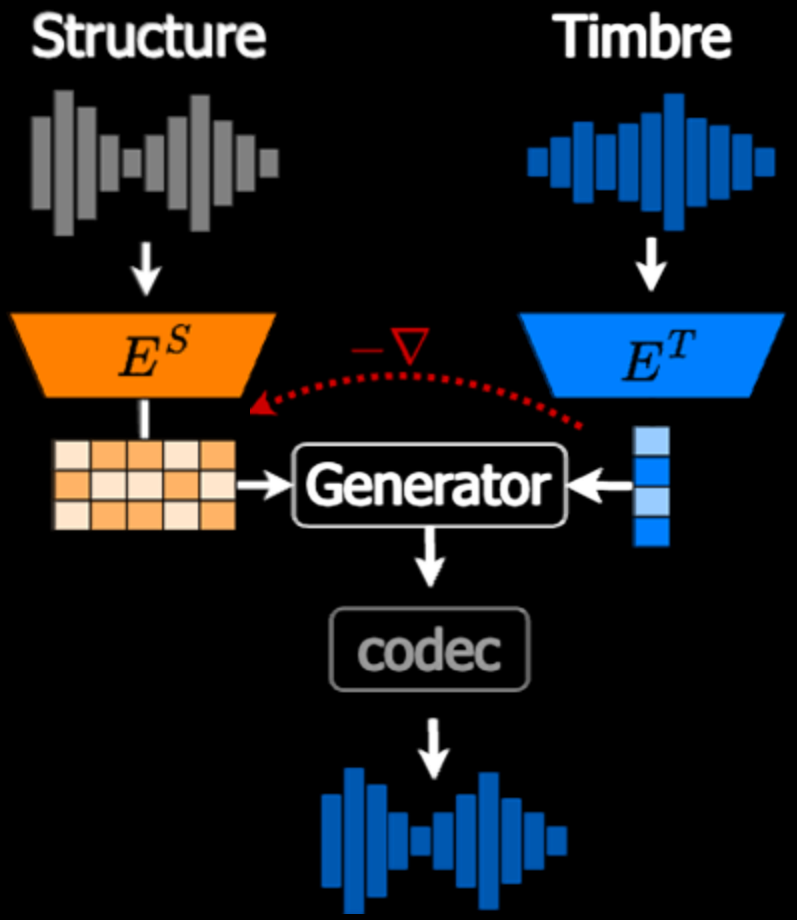
Motivation

- RAVE can do many-to-one transfer
- How to separate “structure” and timbre
- Target **timbre** is extracted from **audio**
- **Structure** input can be **audio or midi**

Model Architecture

- **Diffusion model** as generator
- Two encoders for timbre and structure
- Enforce separation between the two





Training the base model

- Needs a pretrained AutoEncoder
- Rave or from the after repo (~4days)

RAVE

Timbre encoder

- Starts by learning timbre
- Relies on random& crops of the same file
- Builds timbre space for exploring inference

Structure encoder

- Learn structure as "**everything that is not timbre**"
- Key parameter : adversarial strength

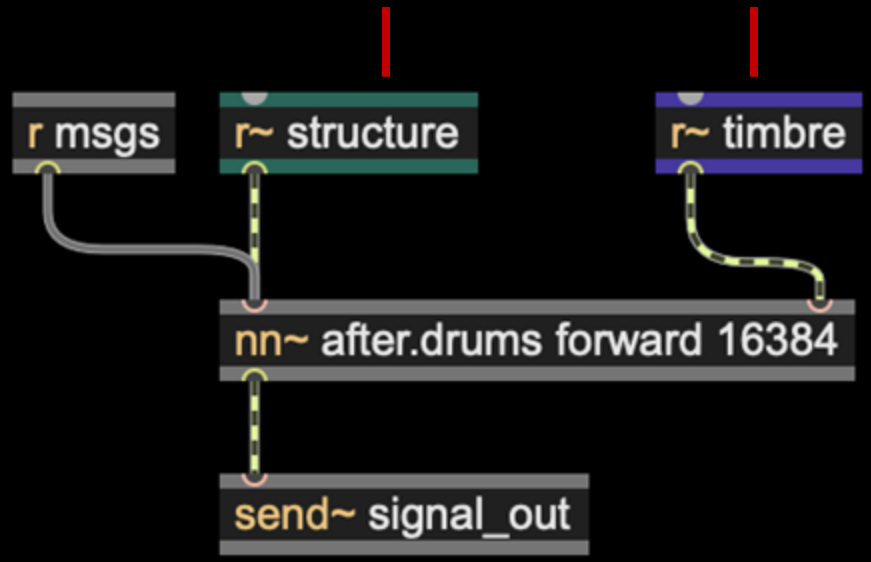
Demerlé, N., Esling, P., Doras, G., & Genova, D. Combining audio control and style transfer using latent diffusion (2024) In *International Society for Music Information Retrieval, ISMIR 2024*

AFTER : Diffusion models for audio transfer

Audio-to-audio transfer

Audio **structure input**
(defines “events”)

Audio **timbre input**
(defines “color”)

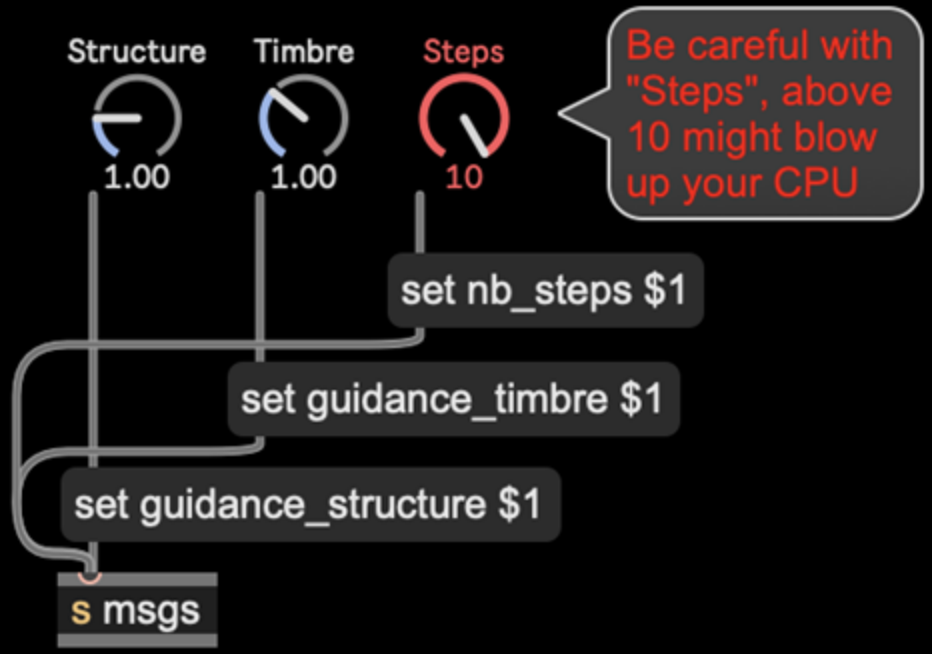


Final result will represent a mixture of

- Events from the structure input
- With a color from the timbre input

Options specific to AFTER

- Set the **weight of structure** in transfer
- Set the **weight of timbre** in transfer
- Number of **diffusion steps** (quality)

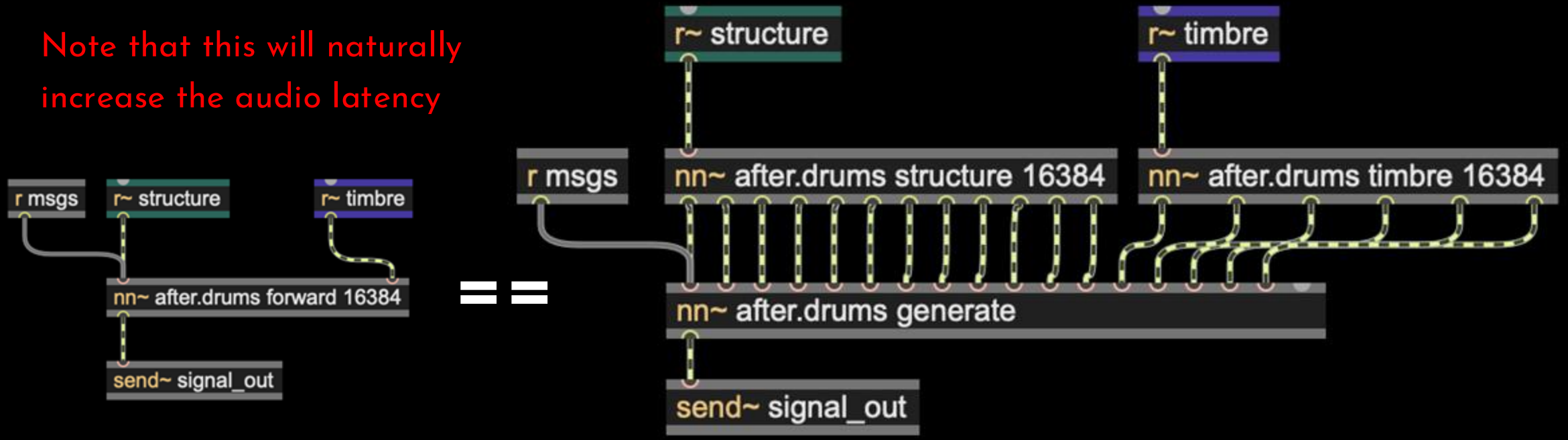


AFTER : Diffusion models for audio transfer

Finer-grained control of the transfer process

Similar to RAVE, we can expose the **internal steps of the computation**
This allows to **tweak the resulting sound** by transforming the latent signals

Note that this will naturally increase the audio latency

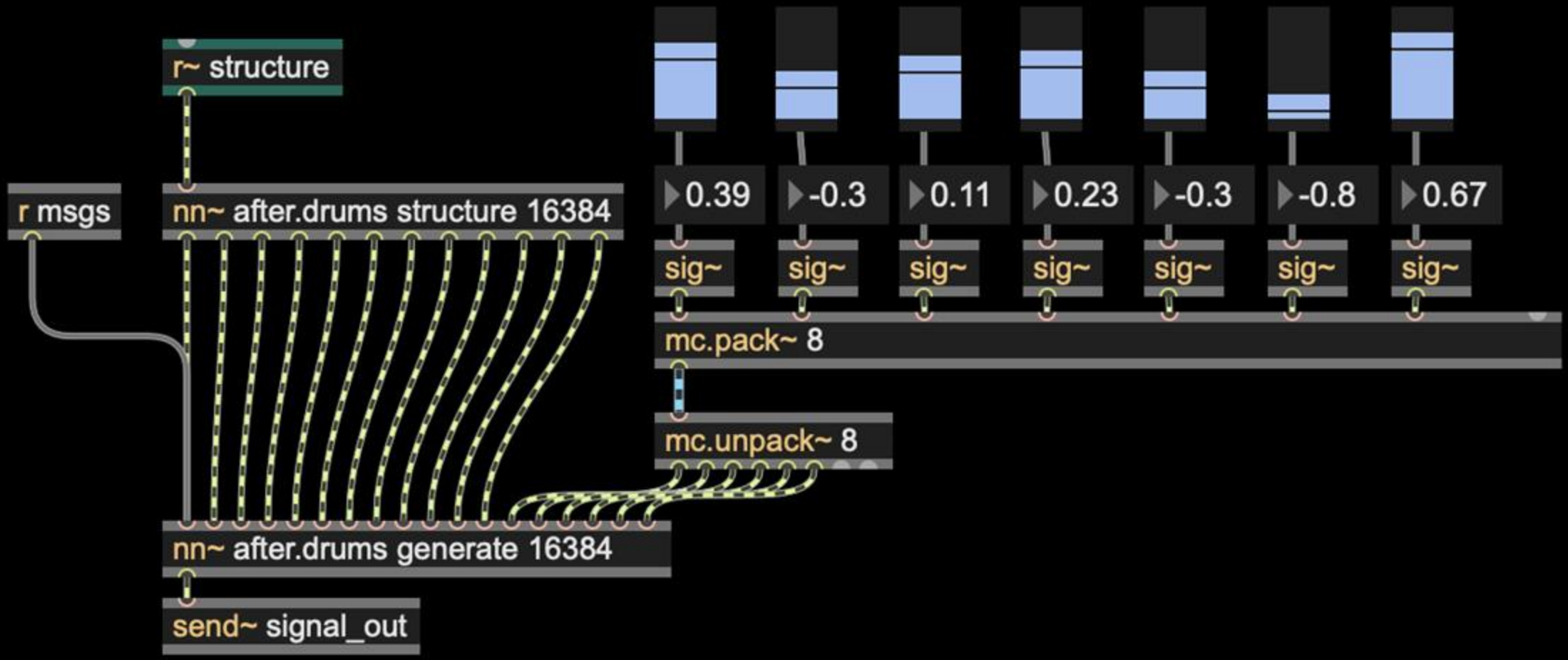


Here we separately expose the **structure encoder** and the **timbre encoder**
(note the new type of functions represented by new names)

AFTER : Diffusion models for audio transfer

Direct control of the timbre latent

Since the **timbre is considered more stable**, we can directly control it with a fixed vector

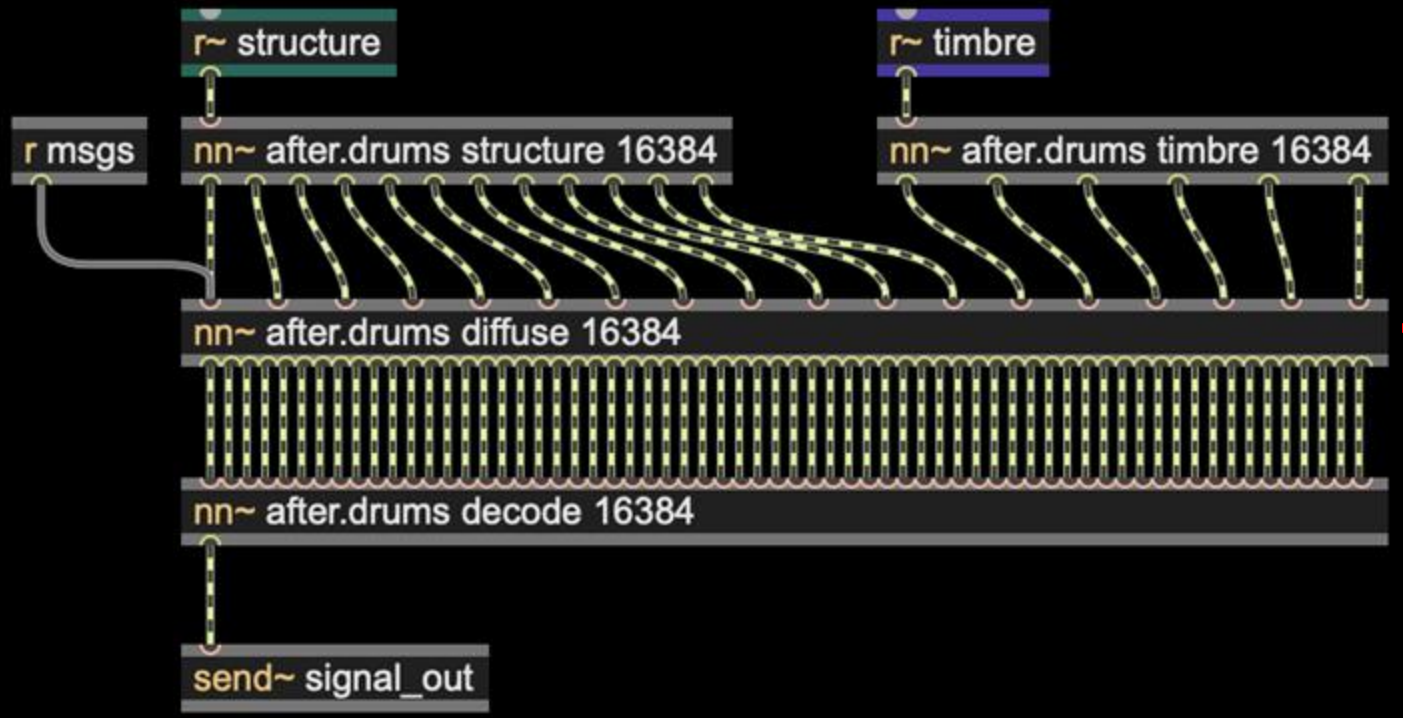


AFTER : Diffusion models for audio transfer

Fully exposed computation in the AFTER models

We can go one step further to expose all parts of the computation

Also exhibit the codec (RAVE-like) part of the computation



Timbre / structure encoders

Latent codec **diffusion model**

Final codec **decoding**

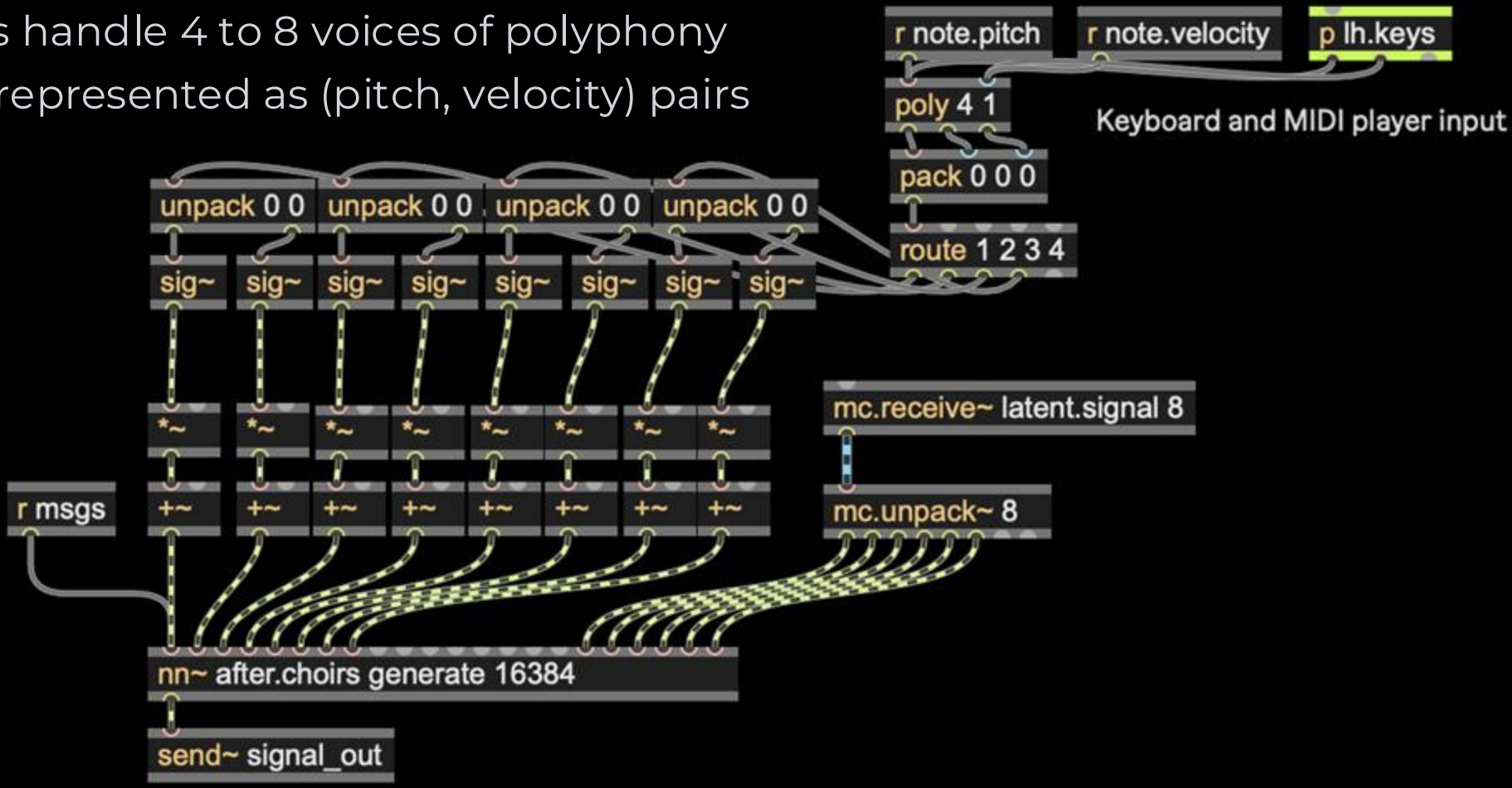
AFTER : Diffusion models for audio transfer

MIDI-to-audio transfer (specific models)

Some of the AFTER models can be **trained to receive MIDI as structure input**

Models handle 4 to 8 voices of polyphony

Notes represented as (pitch, velocity) pairs



ACIDS - Ecosystem

Musical side

MaxMSP

External

Host for deep models

Kind of a modern VST host but for deep learning models

Python

Torchscript exported (.ts)

Model side

RAVE

Timbre transfer

Descriptor control

vschaos

Vintage neural

+?

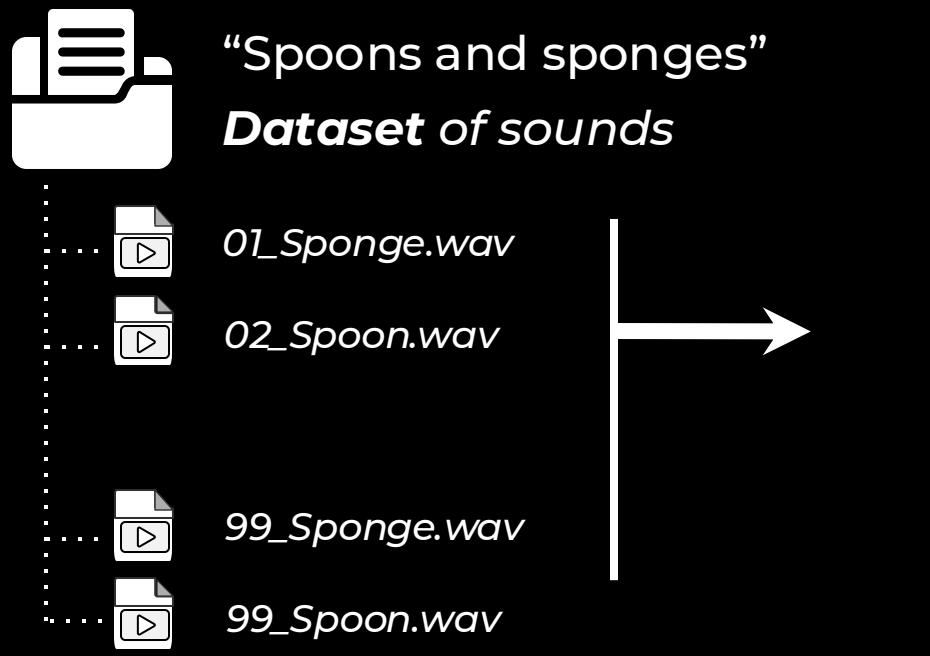
Any deep audio models exported with Torchscript can fit in nn~

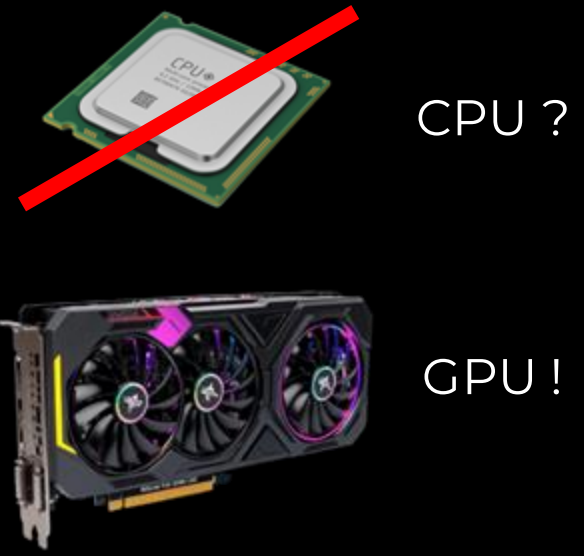
[Demonstration - The new *scripting library* of nn~]

Training the model(s)

How can you train your own model ?

Usually starts by defining the aesthetics you are targeting





80% of the battle is right here

Training functions

Need GPUs

Training the model(s)

How can you train your own model ?

Usually starts by defining the aesthetics you are targeting

80% of the battle is right here

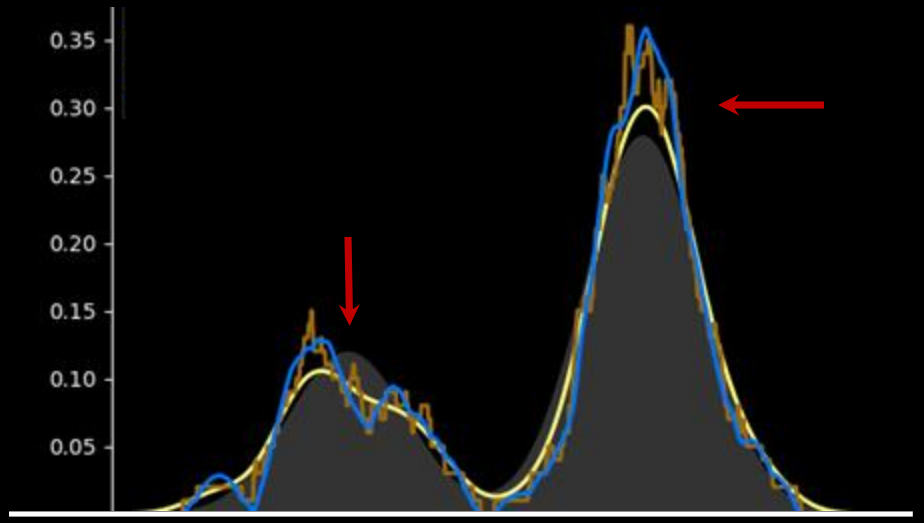
Training functions

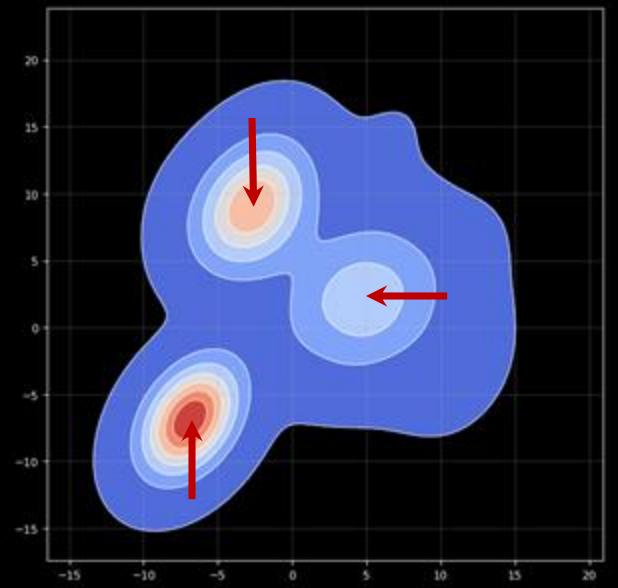
Need GPUs

Designing datasets

How to ensure a good dataset for deep audio generators

- Models like RAVE learn the **data distribution**
  - Your dataset *is the objective*.
- You have to **think in probabilistic modes** (peaks of the distribution)





One mode = one “type of sound” (represented by lots of examples)

Designing datasets - **modes**

One mode = one “type of sound” (represented by lots of examples)

Based on this, **three levers** in dataset complexity **shape learnability**

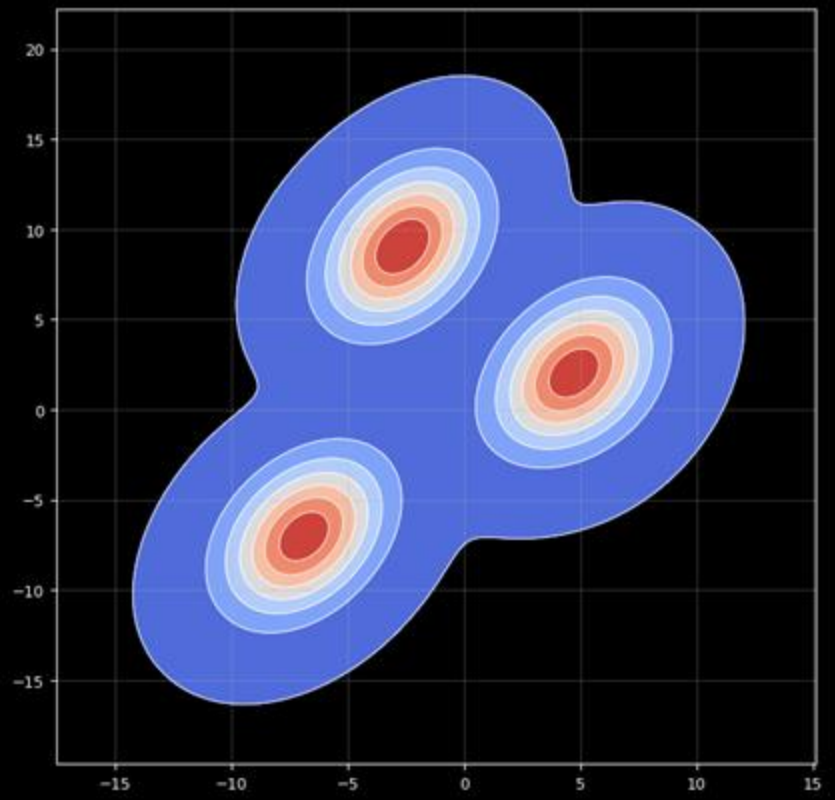
- **number of modes** in your dataset
- **mode density** (how many similar examples)
- **mode variance** (how spread) and **overlapping**

General guidance (for modes) to **set the best ground** for the model

- **Fewer modes** - easier fit (even though a less expressive model)
- **Balance density** - prevent one mode from dominating others
- **Control variance** - ensure you have enough example per sound type
- **Overlapping modes** - make sure there is a “continuous domain”
- **Avoid outliers** (make sure that the modes are clear)

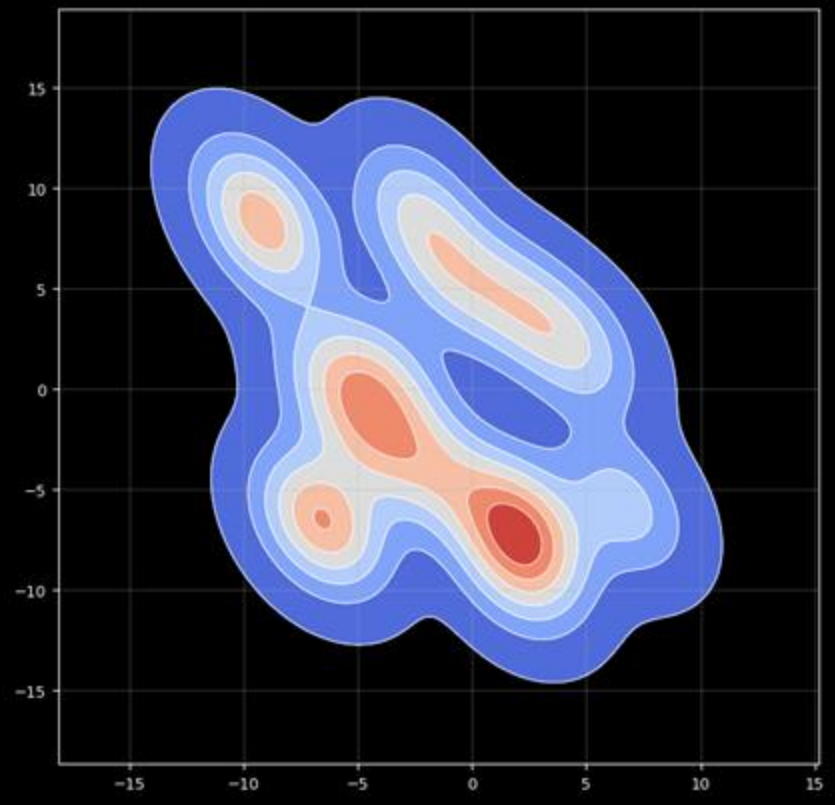
Designing datasets - **modes**

Fewer modes



- Easier to train
- Less expressive

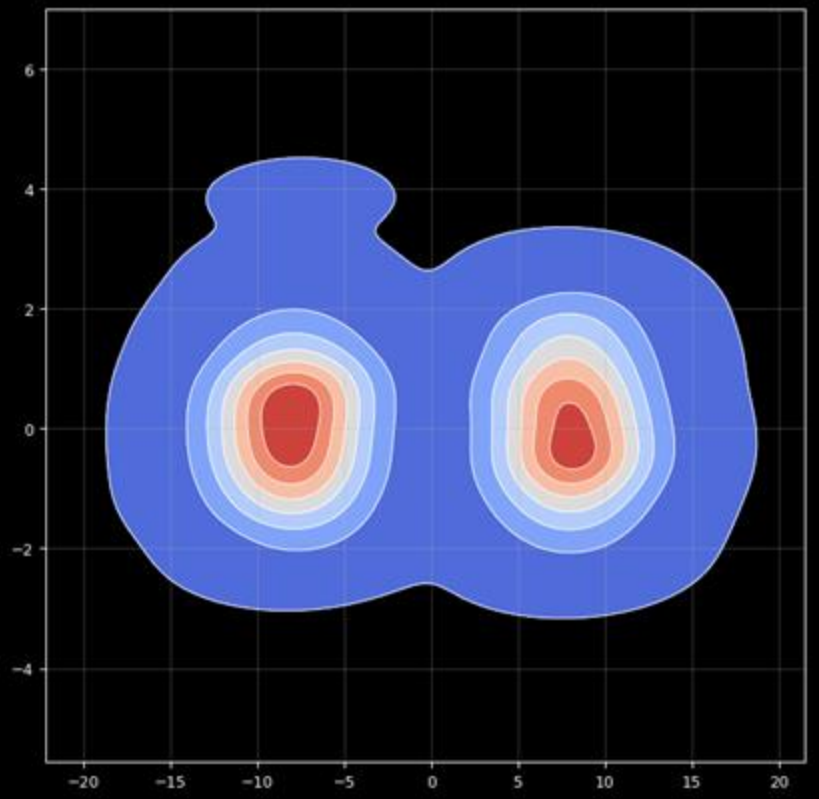
More modes



- Need more data
- Need larger model

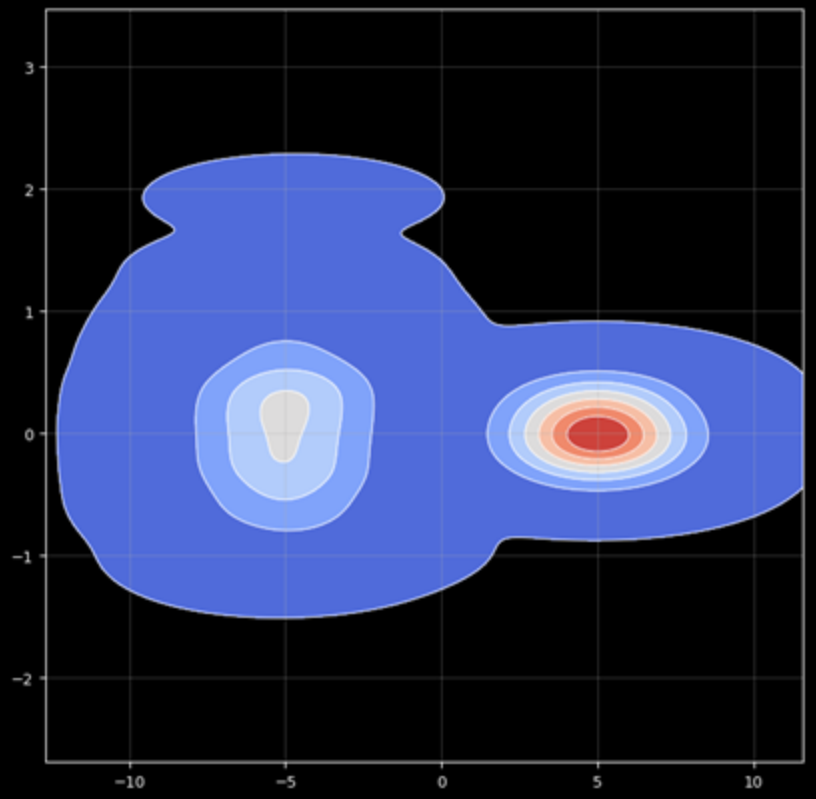
Designing datasets - **modes density**

Balanced density



- Easier to train
- Both modes learnable

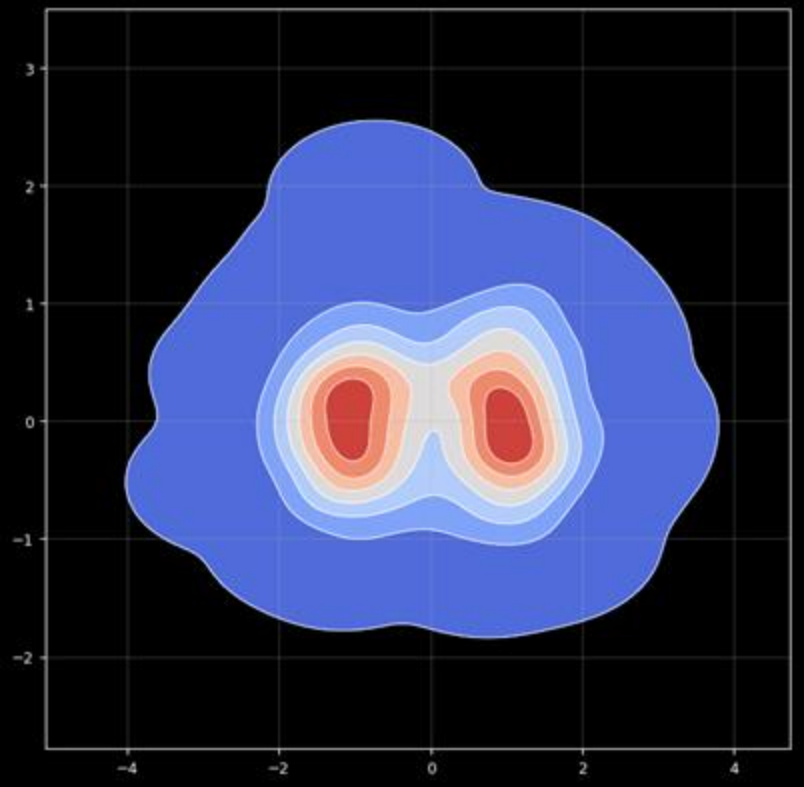
Imbalanced mode



- Left mode ignored
- Might even skew others

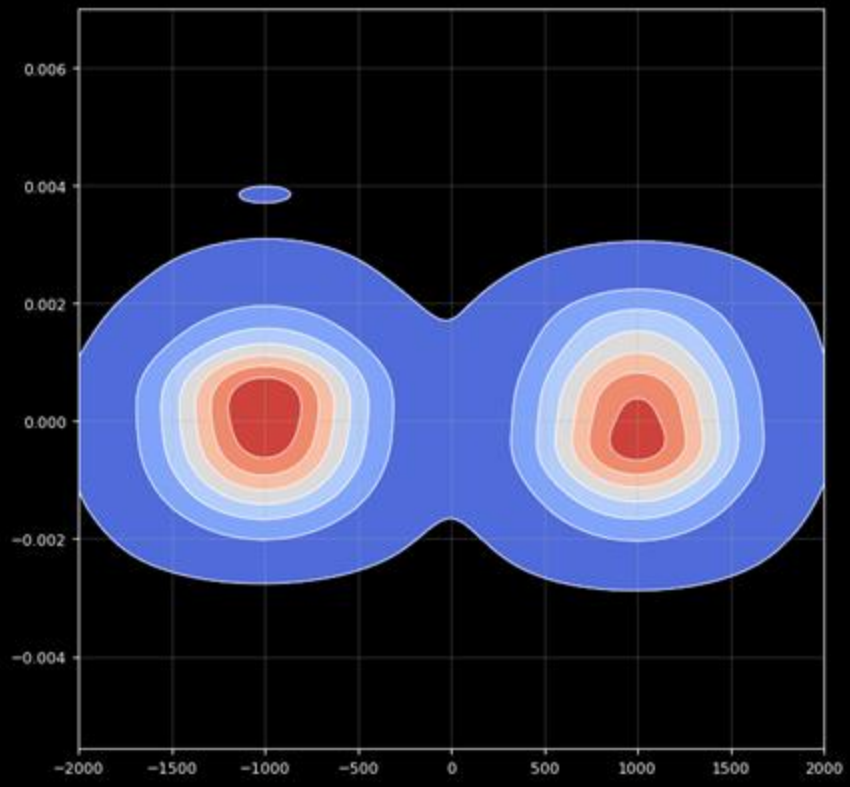
Designing datasets - **modes distance**

Overlapping modes



- Easier to train
- Both modes learnable

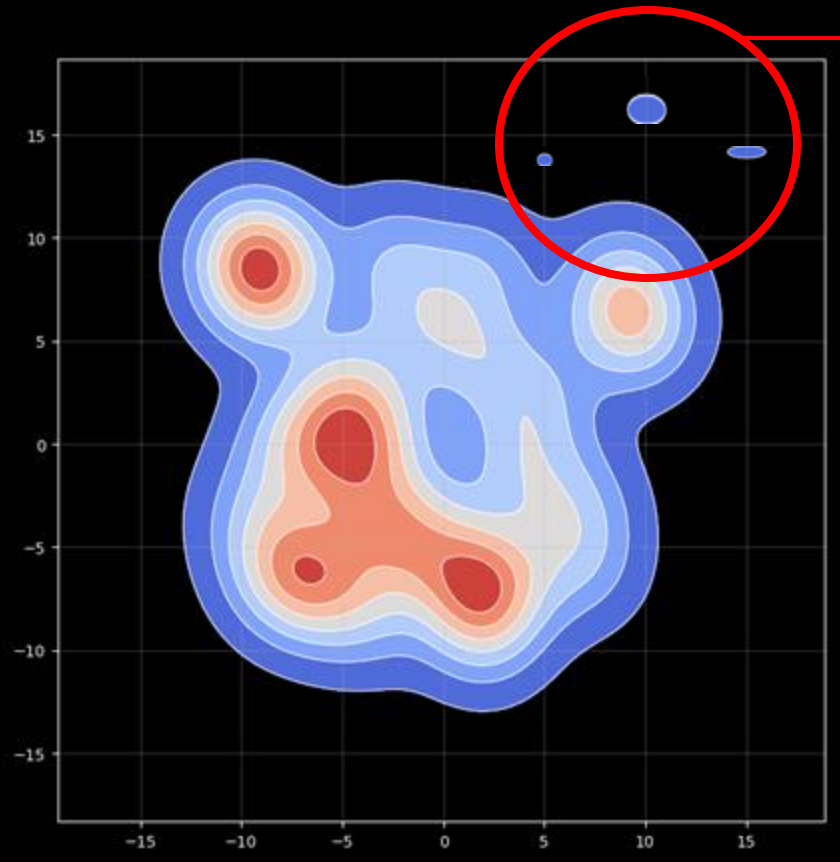
Disconnected modes



- Harder to train
- Interpolation impossible

Designing datasets - **outliers / long tail**

Outliers are point lost “outside of modes”



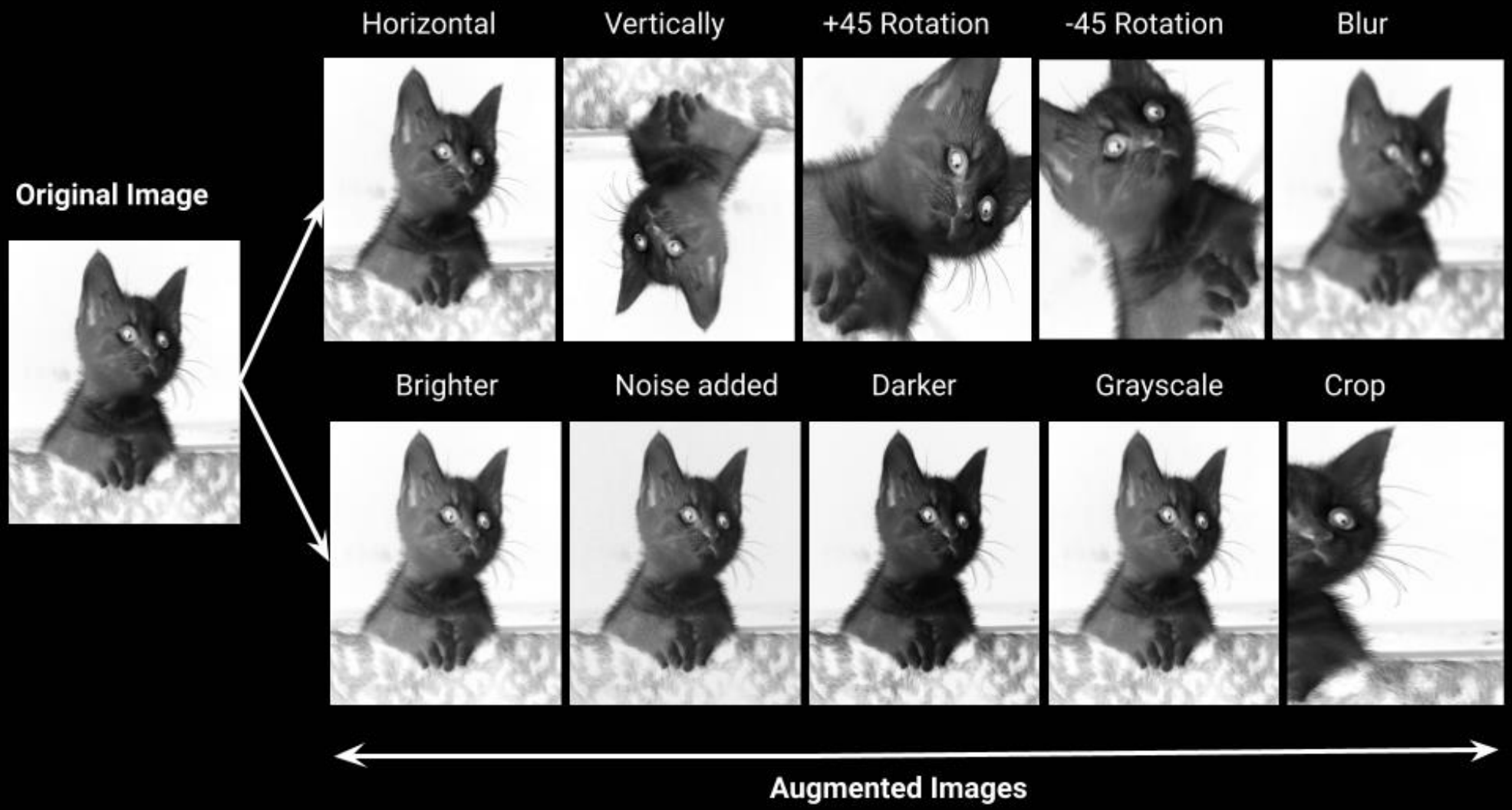
Outliers

- Will be most certainly badly modeled
- But might also skew the learning !

- Typical culprits
  - Clipped takes, noise, long silence
  - Skewed EQ, accidental exports.
  - Mislabelling
- Detect via feature space distances
- Validate with listening sweeps

Quick note on **data augmentations**

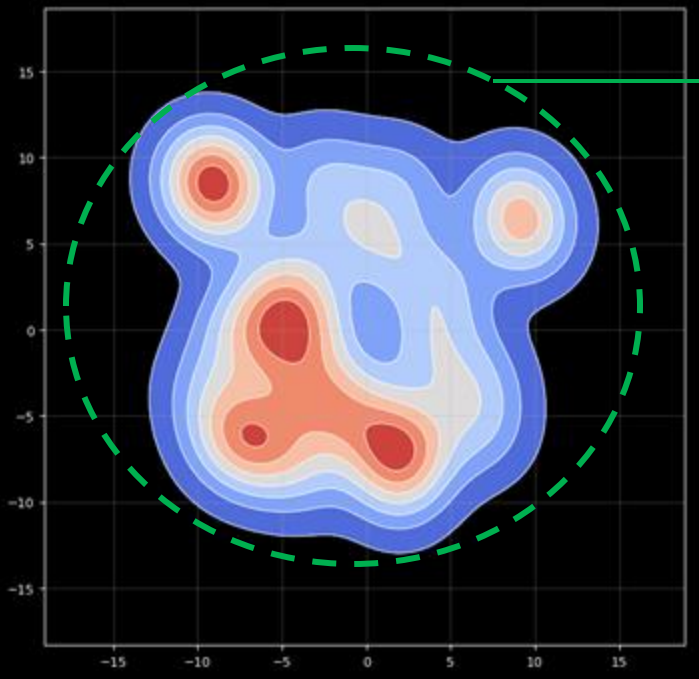
Helping the model with *data augmentations*



Quick note on **data augmentations**

Helping the model with *data augmentations*

- Add more data by pitch-shifting, adding small noise, etc...
- The only silver bullet we have in machine learning !
- Augmentations **should stay *within* a mode's perceptual basin**
  - For instance pitch-preserving for timbre models



Convex hull of your dataset

Some models (such as RAVE) include this option in their training function

Remember that generative models will learn to generate augmented data too !

Designing datasets - **practical checklist**

- **Format** - Pick a single sample rate, channels and bit depth
- **Quality** - Model learns to reconstruct, so good sample quality is crucial
  - Always resample with high-quality anti-aliasing.
- **Quantity -** Think hours not minutes (the more the better)
  - 1 to 3 h per target timbre/technique (mode) is a workable floor
  - Use more data if variance of the target is high.
- **Homogeneity / diversity -** Use the *previous discussion on modes*
  - Have a good ratio between diversity and similarity
  - Oversample rare modes, collect more or consider removing them
- **Dynamics** - Best performance with reasonable loudness
  - Try to have as much *loudness variations* as possible
  - Leave some silence otherwise the model will not shut up.
- **Avoid faults** - Remove DC, enforce max peak and prevent clipping.

Remember to **document everything** to smoke out problems

Scaling laws and data

How data naturally regularizes

Training the model(s)

How can you train your own model ?

Usually starts by defining the aesthetics you are targeting

80% of the battle is right here

Training functions

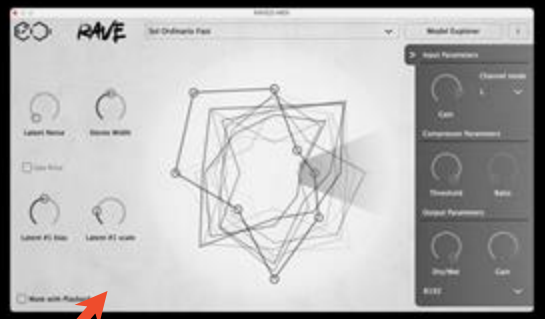
Need GPUs

Understanding different actors

Musical side

MaxMSP

nn~



VST

Torchscript

Compile & export AI models

output ⇢ weight + architecture (`.ts` file)

Python

Model side

RAVE

Timbre transfer

Pytorch → library for AI

output ⇢ weights (`.ckpt` files)

`.py`

Python → program

output ⇢ set of `.py` files

Training other models

vschaos

- Spectrogram-based
- Needs less data
- GPU-light (<1 day)
- Conditionable (e.g., pitch)
- Specific targets generation
- No VST

RAVE

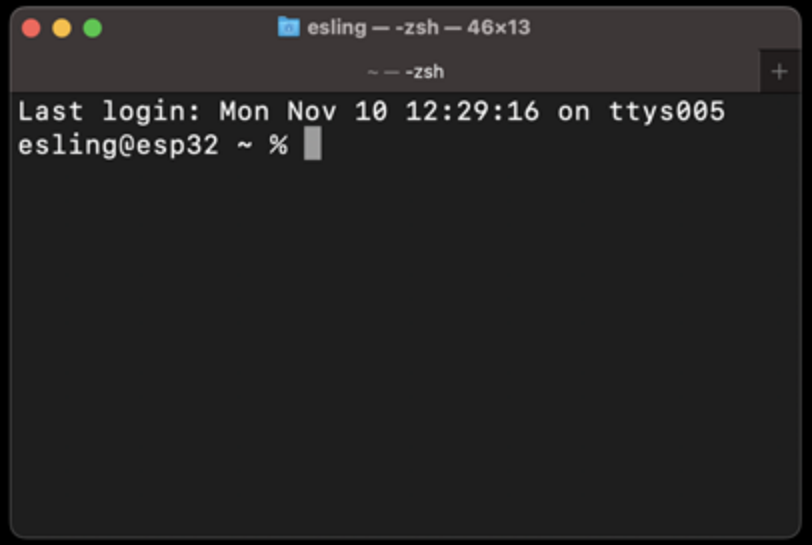
- Waveform-based
- Requires lots of data
- GPU-intensive (~5 days)
- Hard to add information
- Specifically timbre transfer
- VST

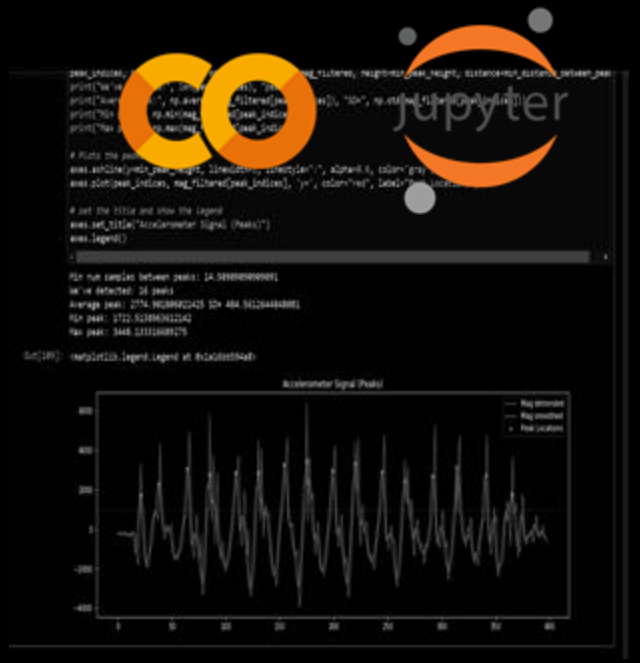
- Waveform and MIDI
- Requires lots of data
- GPU-intensive (~5 days)
- Explicit timbre separation
- Conditioned generation
- No VST

Depending on your data and needs always think upfront of the properties of the model

Interaction for training

Two major approaches for model training



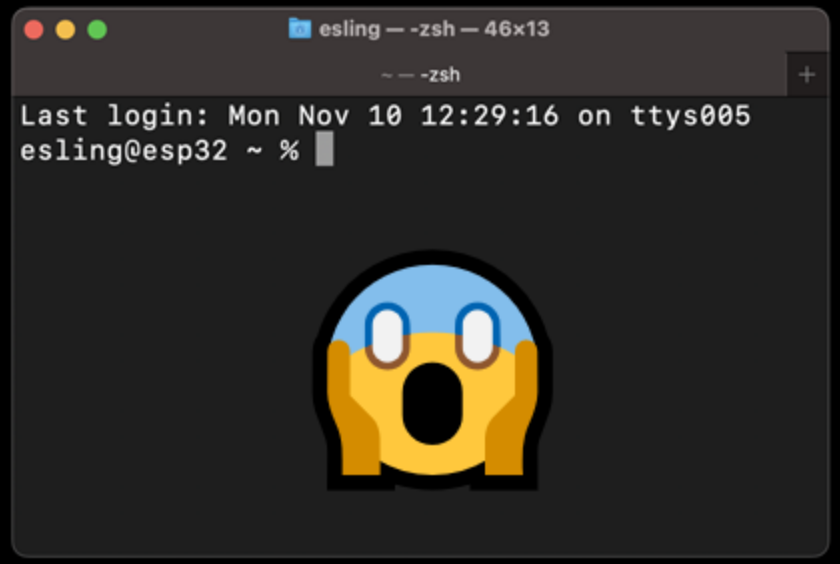


Command line (shell)

Interactive notebooks (jupyter / collab)

Model Training - shell

Almost all model training imply using the **shell** (terminal)



Minimal set of commands to understand

```
cd              : change directory
mkdir           : make a directory
open            : open a file
ls              : make a directory
screen          : create a detached process
python *f*      : execute python file f
```

Most commands accept argument preceded by double dashes

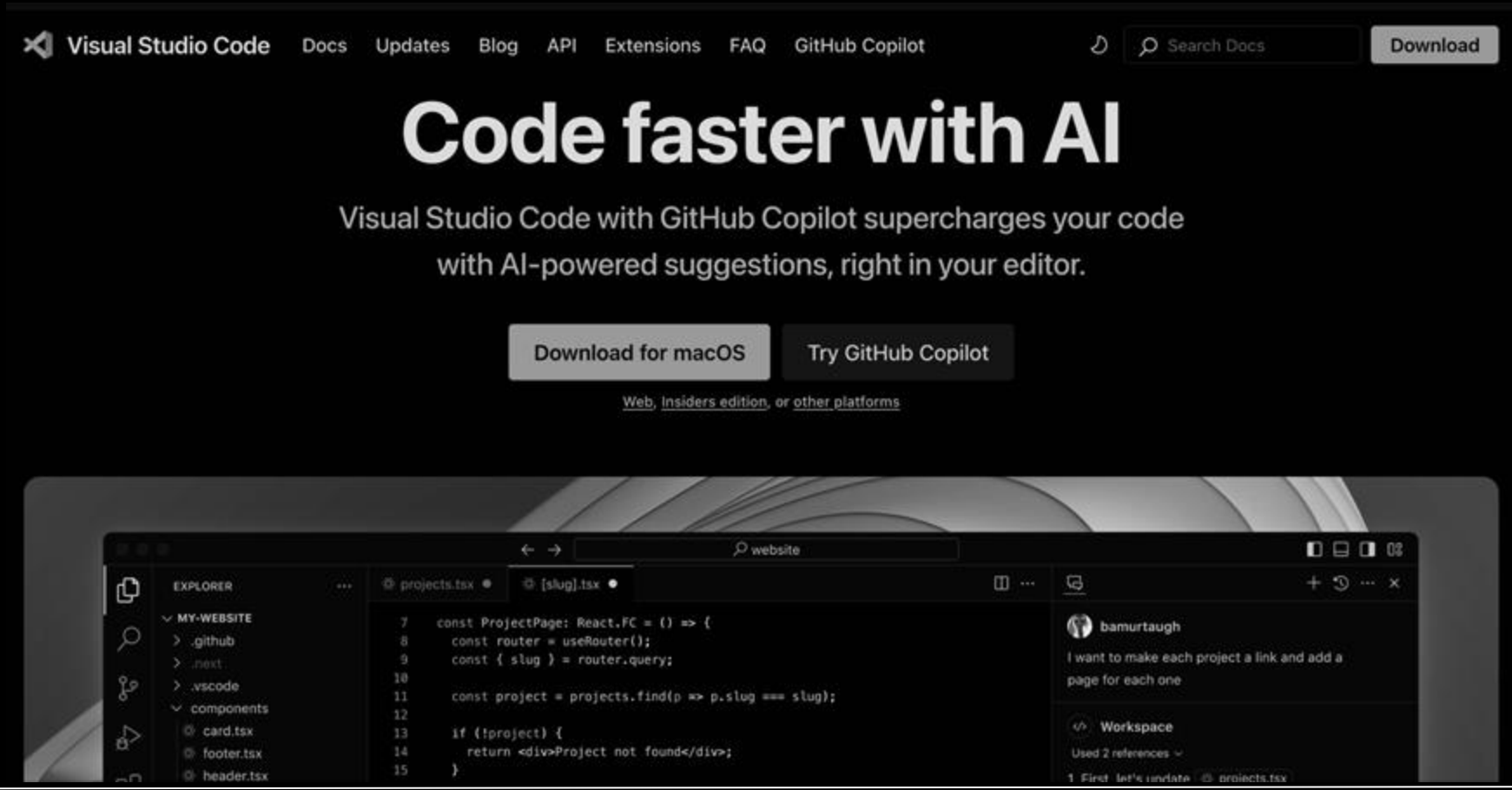
```
python3.13 train.py --n_epochs 2000 --batch_size 32 --gpu 2
```

Learn as much shell as possible & always try adding `--help` or `-h`

Python coding

Highly recommend using the Visual Studio Code (VSCode) IDE

https://code.visualstudio.com/



Using environments

Why using environments?

```
1. which python3
```

➠ global python3 interpreter

vschaos2

```
1. torch==1.12
```

RAVE

```
1. torch >= 2.1
```

➡ **conflict**

python environments are **local setups** that allow to separate dependencies for several applications

vschaos2

```
1. python=3.11
2. torch=1.12
```

RAVE

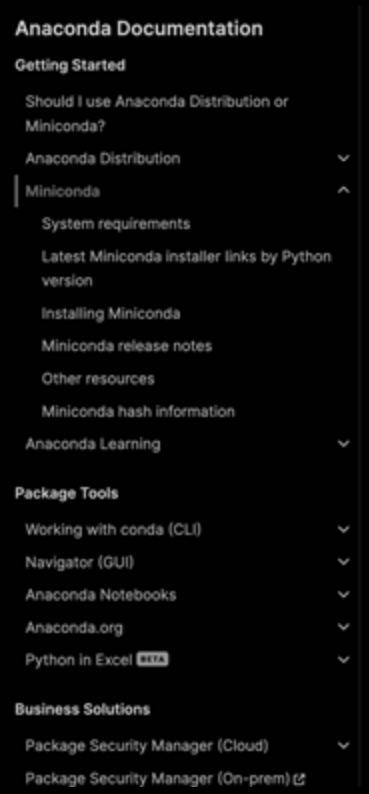
```
1. python=3.13
2. torch=2.1
```

Python environment

Installing Miniconda

https://docs.anaconda.com/miniconda/

1. **Download** Miniconda
2. **Run command**
   `bash Miniconda3.sh`
3. **Follow prompts**
4. **Close and reopen**



Platform	Name	SHA256 hash
Windows	Miniconda3 Windows 64-bit	3a8897cc5d27236ade8659f0e119f3a3ccaad68a45de45bfdd3102d8bec41
macOS	Miniconda3 macOS Intel x86 64-bit bash	ce3b440c32c9c636bbe529477fd496798c35b96d9db1838e3df6b0a80714d
	Miniconda3 macOS Intel x86 64-bit pkg	6e4495868cb82b8ae5ecb26e96aedc2400cc0fbe70288a64ebfa025faa1ec
	Miniconda3 macOS Apple M1 64-bit bash	08d8a82ed21d2dae707554d540b172fe03327347db747644fbb33abfaf07f
	Miniconda3 macOS Apple M1 64-bit pkg	d68e714ca3f433b1bea52acd050452b6bb0aaa89878869cb30bfffc729c05
Linux	Miniconda3 Linux 64-	8d936ba600300e08eca3d874dee88c61c6f39303597b2b66baee54af4f7b4

Python environment

The system **venv** is **natively present** with Python

1. **Prepare** the virtual environment

```
python -m venv /path/to/env/
```

2. **Activate** the environment

```
source /path/to/env/bin/activate
```

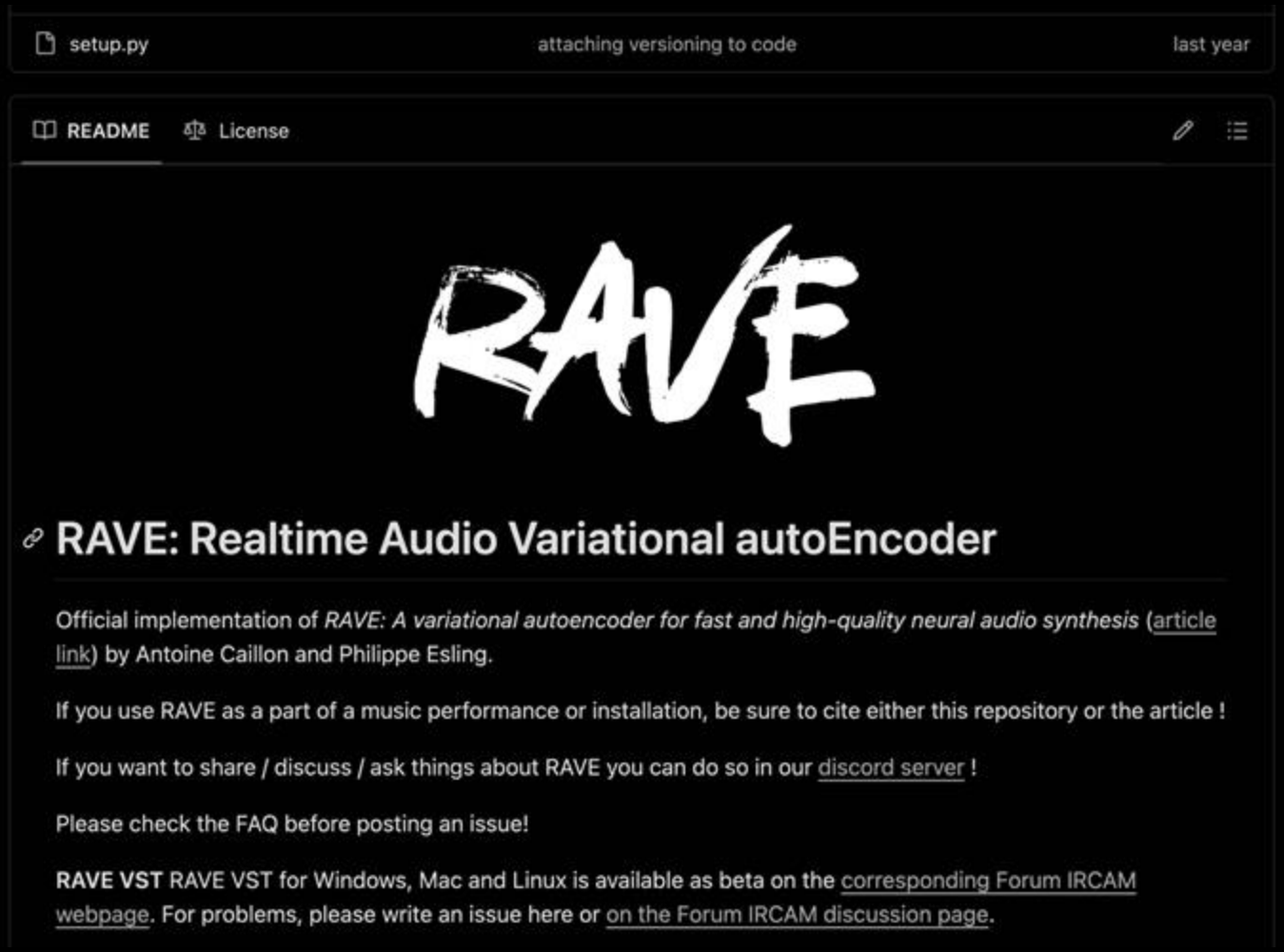
3. **Install** the libraries (check the prefix)

```
(env) pip install acids-rave
```

Training your own model

Golden advice for using models

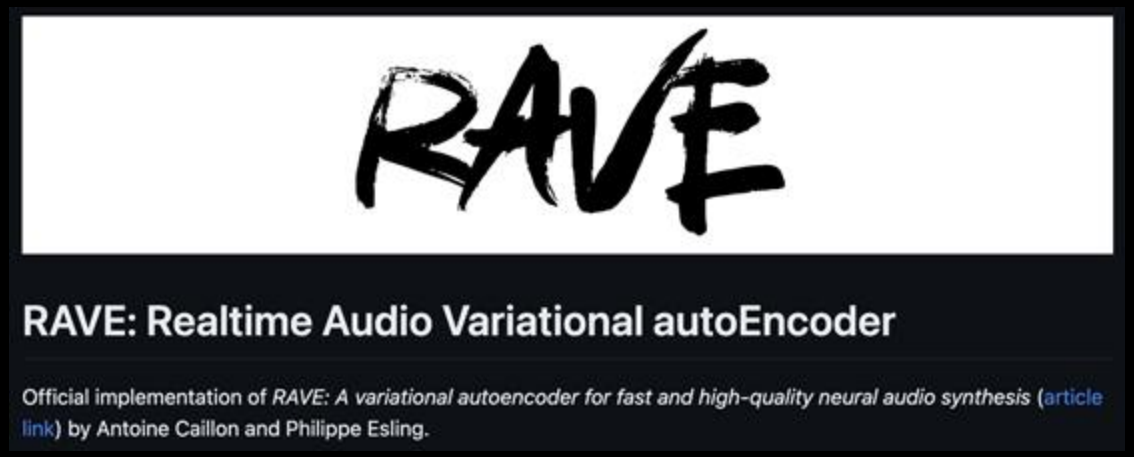
Always **read the README!** (RTFM)



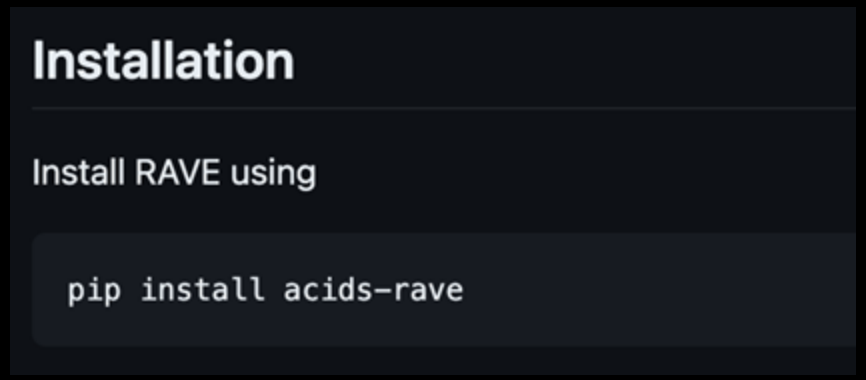
Training your own model (RAVE)

Directly using the RAVE model

https://github.com/acids-ircam/RAVE

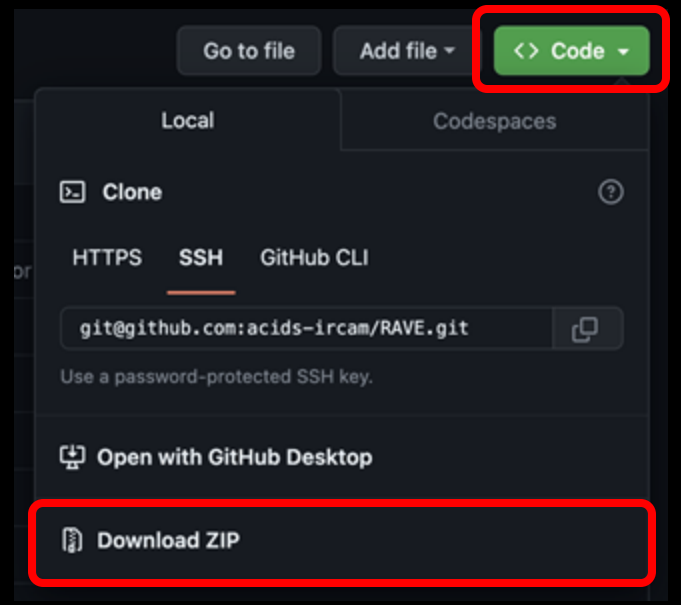


Installing the package



Direct installation

Direct code access



Training your RAVE - **functions**

Training script

Colab

A colab to train RAVEv2 is now available thanks to hexorcismos ! Open in Colab

Preprocess the audio dataset

Shell (command line)

```
rave preprocess --input_path /audio/folder --output_path /dataset/path (--lazy)
```

Train the model on our audio dataset

Shell (command line)

```
rave train --config v2 --db_path /dataset/path --name give_a_name
```

Export the model to TorchScript (for MaxMSP)

Shell (command line)

```
rave export --run /path/to/your/run (--streaming)
```

Training your own model (RAVE)

Preprocess the audio dataset

Shell (command line)

```
> rave preprocess --input_path /audio_dir --output_path /data_dir --channels 1
```

This function will go through your dataset to normalize and split it in learnable chunks

Mandatory arguments

--input_path /audio_dir	Path to your folder containing audio samples
--output_path /data_dir	Target path to store the processed dataset
--channels 1	Number of channels to train the model

Notes on the preprocessing steps

- The function **splits long audio files** so you can have whole hours in single files
- We **highly recommend to train the model in mono** (1 channel), which works best
- You need to **do this step again if you want to add** new audio files to the dataset
- You can **check the properties** of your set in the `metadata.yaml` file

Training your own model (RAVE)

Train the model on our audio dataset

Shell (command line)

```
> rave train --config v2 --db_path /data_dir --name model_name --gpu -1
```

This function starts the training on a **preprocessed dataset** (folder from previous step)

Mandatory arguments

--config v2	Type of model (configuration) to train (*see next slides*)
--db_path /data_dir	Path to the processed dataset (cf. previous slide)
--name model_name	Name of the output model
--gpu 0	Device to train the model (N or -1 = cpu)

Others

	causal	Use causal convolutions (**faster model but lower quality**)
--config	**noise**	Enables noise synthesizer (better for noisy sounds)
	(regularize)	Several regularizations available for V2

Important The training takes a lot of time, so always run it in a screen environment

Training your own model (RAVE)

Notes on the training

You absolutely need a CUDA-compatible GPU to train models
Different models need different amount of VRAM memory (check specs)

Types of architectures *Note that you can create your own configurations*

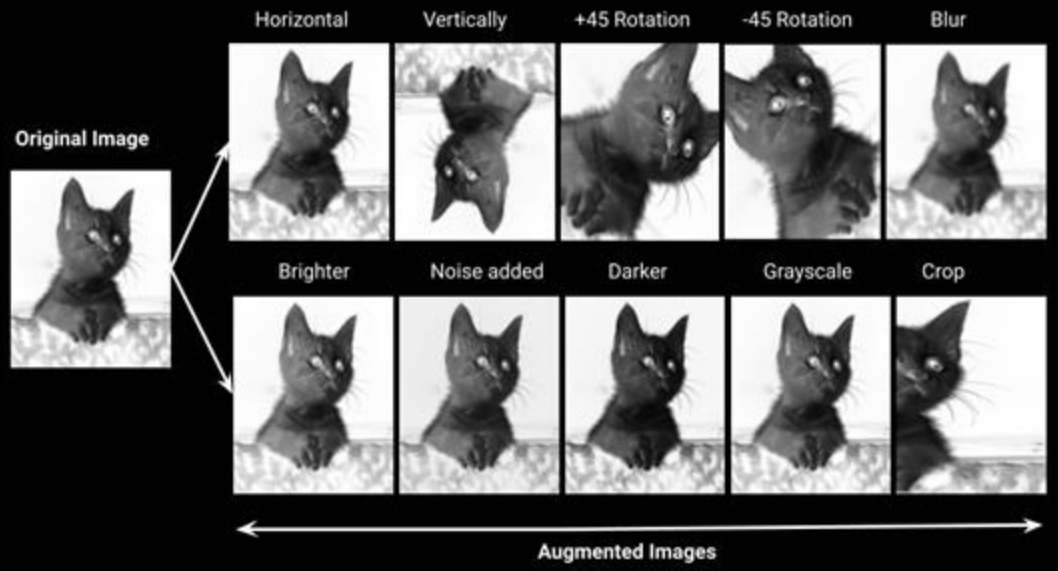
`v1`	Original continuous model
`v2`	Improved model (better for complex sounds)
`v3`	Modernized version (larger and using instance norm)
`v2_small`	Smaller receptive field, better for static sounds
`v2_nopqmf`	Directly works on waveform (adapted for bending)
`discrete`	Discrete latent (useful for using as codec to a diffuser)
`onnx`	Noiseless v1 for onnx
`raspberry`	Lightweight version for real-time on Raspberry

Recommandations

- Simple or short sounds : `v1, v2_small`
- Complex sounds or layered music : `v2, v3`
- Lightweight embedded model : `raspberry`

Using augmentations

The **only silver bullet** in machine learning



- Apply non-destructive transforms to the input
  - (image) zoom, rotate, flip, scale, ...
  - (audio) transpose, stretch, volume, ...
- Provides more input points to the model

Be careful with generative models

They will learn to generate augmented data too !

Available in RAVE

`mute`	Randomly mute some parts (helps learning silence)
`compress`	Compress the waveform (helps with loudness)
`gain`	Perform random gains (helps with loudness)

More augmentations available with `acids-dataset`

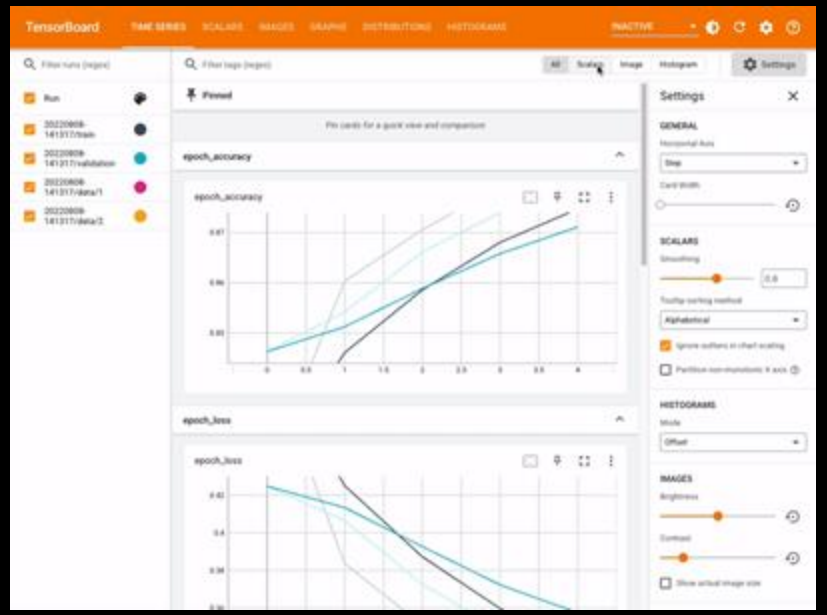
Monitoring your trainings with Tensorboard

Since training is long, you should always **monitor how the model behaves early**

```
> cd model_name_xxxxxxxxx
> tensorboard --logdir . --bind_all
```

```
TensorBoard 2.16.2 at http://7d1b6be8ce5b:6006/
```

In your browser, go to page **http://xxxxxxxxxx:6006**



Tensorboard

The only good thing in Tensorflow

- Allows to monitor how model is behaving **during training**
  - Check evolution of the loss (**scalars**)
  - Listen to some sound examples (**audio**)

Detect problems early on in the training

Resume trainings with checkpoints

You can always **resume and even finetune** your existing checkpoints !

Models are periodically saved (see `save_every` option) so you can restart training if crashed

```
> rave train ... --ckpt model_name_xxxxxxxxx ...
```

This will continue the training from the last best saved model

Notes on the finetuning process

- This is especially useful if you use GPU services (collab, cloud)
- You can of course continue training on different machines
- This also opens up the possibility to **creative finetuning**
  - Start the training on a given (usually large) dataset
  - Fine-tune on a smaller dataset to keep quality but have specific transfer
  - Also possible to **start on one set and finetune on a completely different one**

Training your own model (RAVE)

Export the model to TorchScript (for MaxMSP)

Shell (command line)

```
> rave export --run /model_path (--streaming)
```

Finally, we need to export the model to obtain a `.ts` file that can be used in MaxMSP

Mandatory arguments

`--rum` `/model_path`

`--streaming`

Notebook version

Quick introduction on notebook-based training

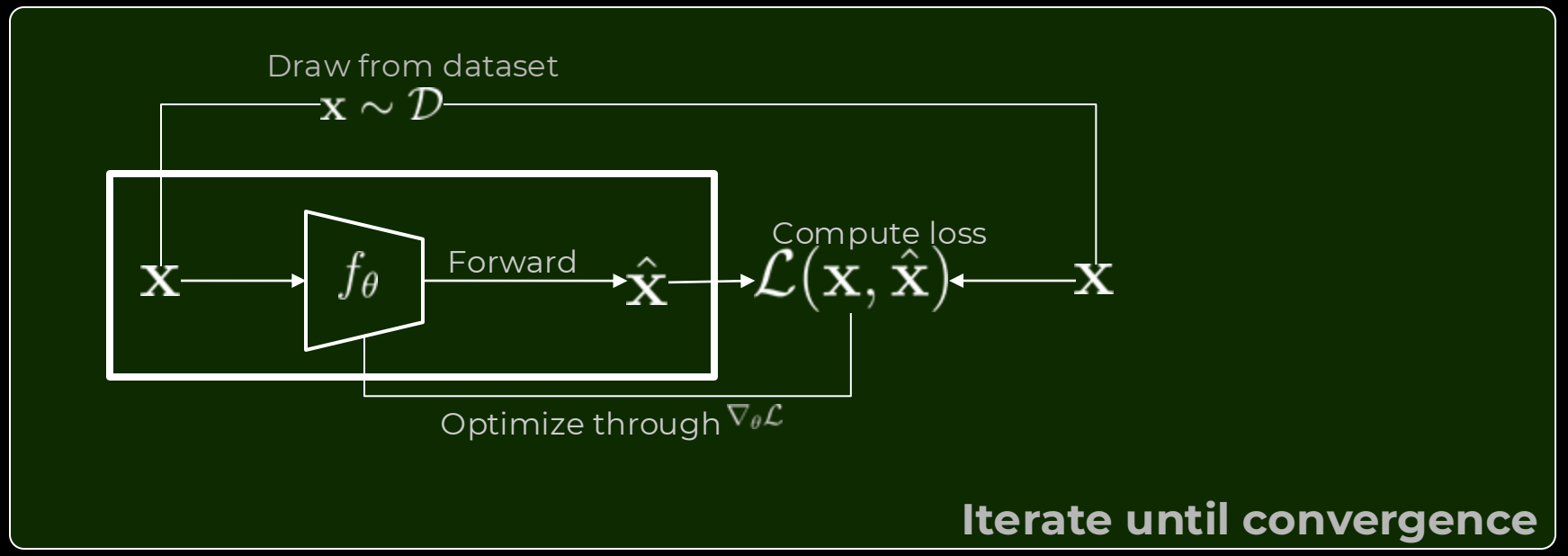
Model Training

Deep learning basic elements

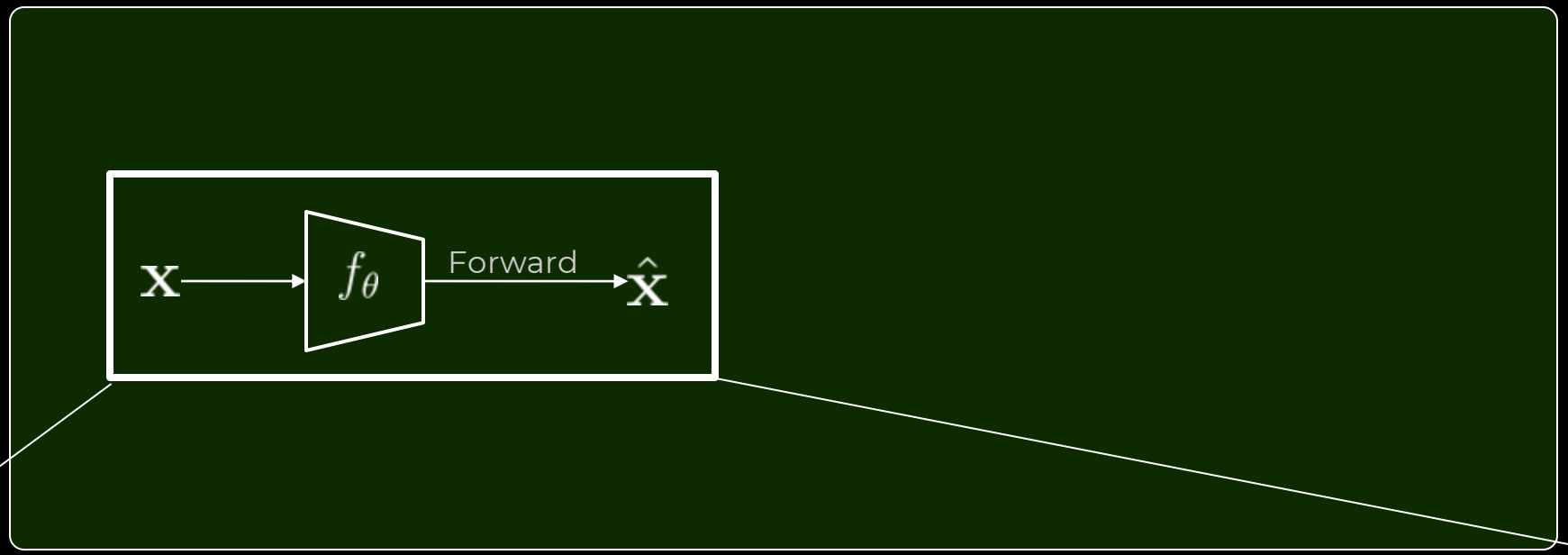
1. **Dataset**
   (torch.utils.data.Dataset) $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})_i\}_{i=1}^N$
2. **Model** (deep neural network) $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$
   (torch.nn.Module)
3. **Loss** (cost) function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$
4. **Optimizer** $\theta \mapsto \theta^*$

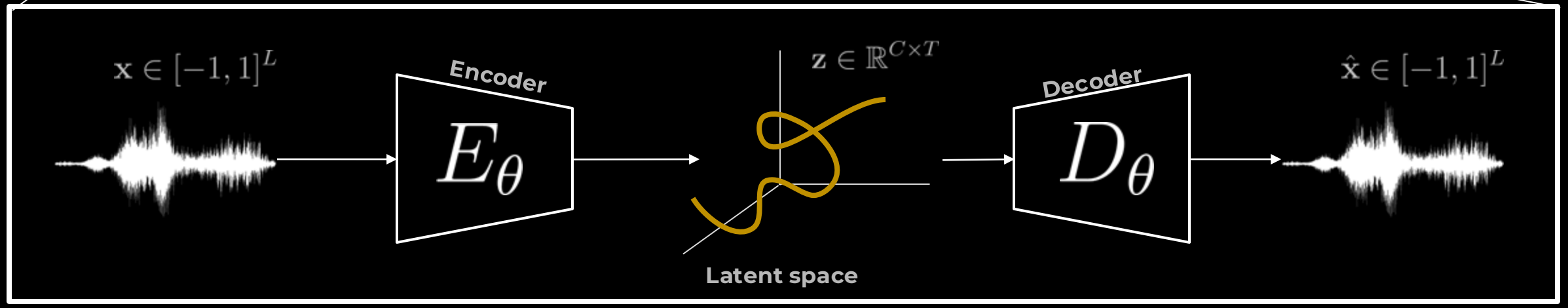
```
# Given f_theta, dataset, loss_fn, optimizer
while not converged:
  draw (x, y) ~ dataset
  y_hat = f_theta(x)
  loss = loss_fn(y, y_hat)
  loss.backward()
  optimizer.step()
return f_theta
```

Training your own model (RAVE)

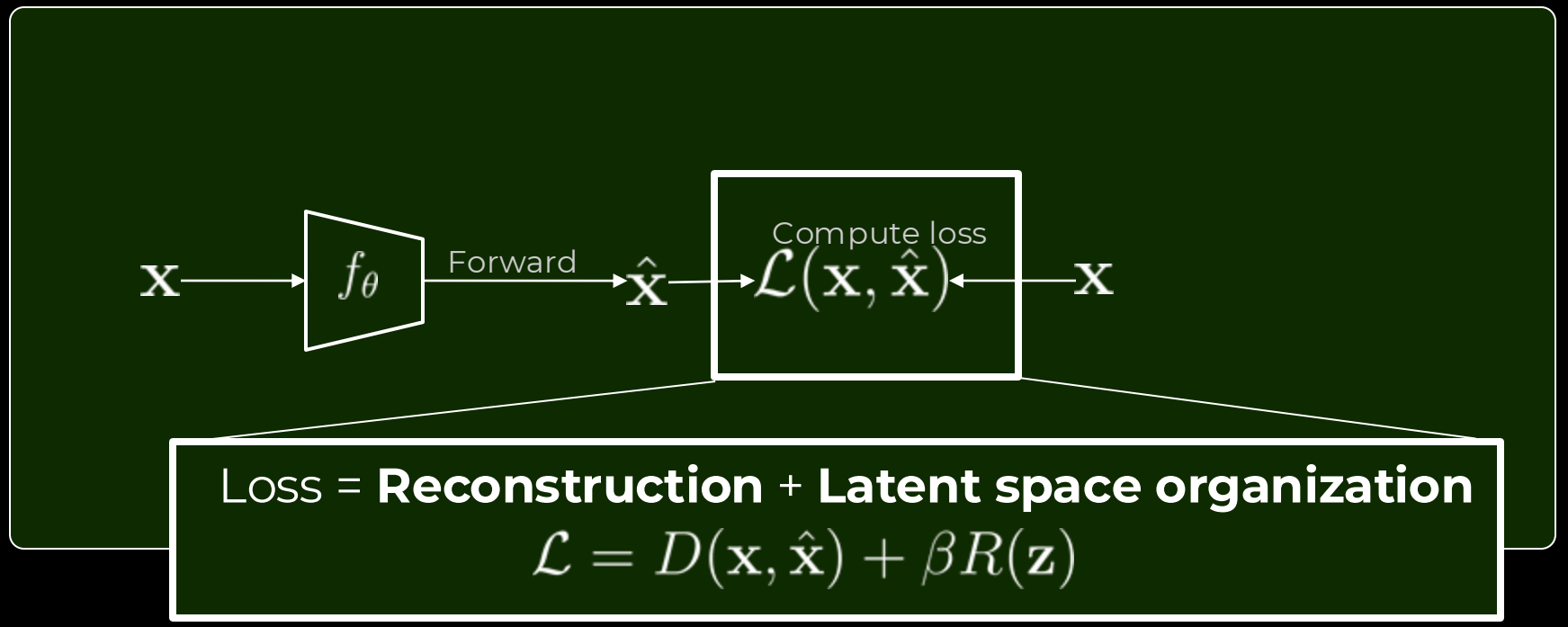


Training your own model (RAVE)

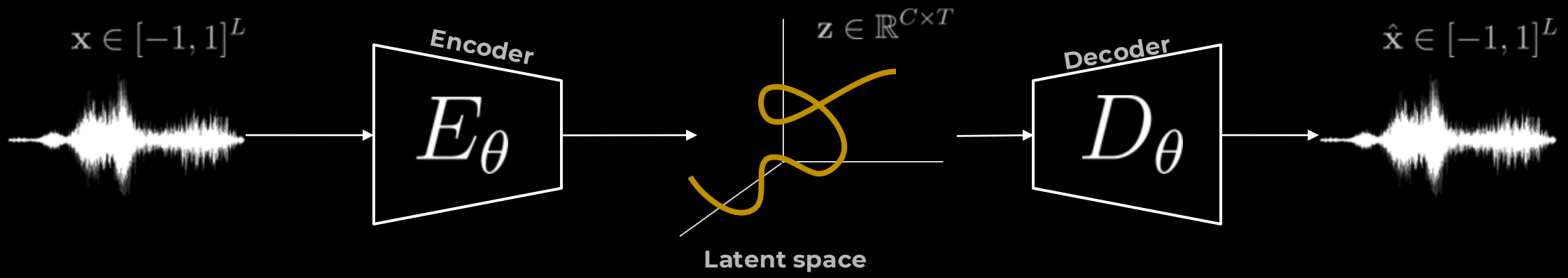




Training your own model (RAVE)



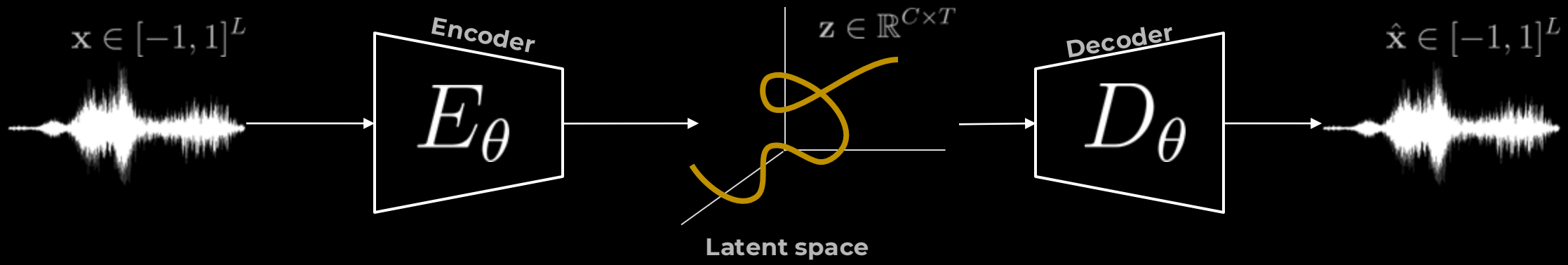
Variational Autoencoder



Training other models

[1] https://github.com/acids-ircam/vschaos2

RAVE

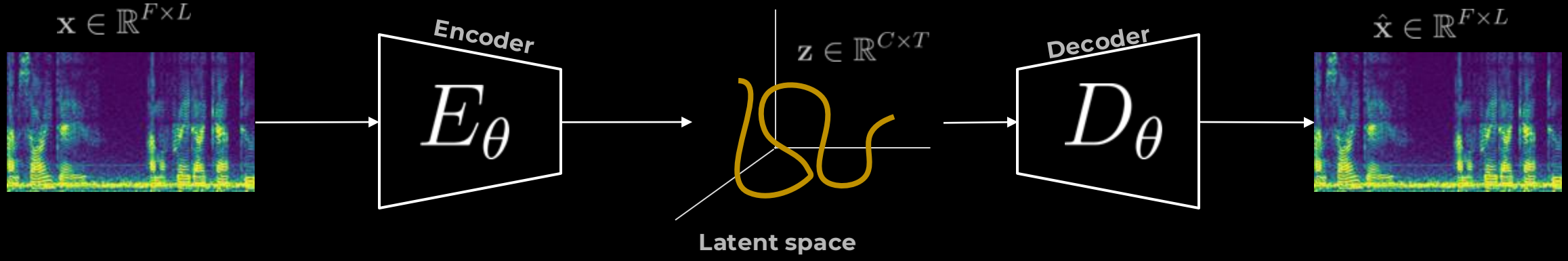


Loss = **Reconstruction** + **Latent space organization**

$$\mathcal{L} = D(\mathbf{x}, \hat{\mathbf{x}}) + \beta R(\mathbf{z})$$

Variational Autoencoder

vschaos [1]



Training your own model (AFTER)

Active GitHub
github.com/acids-ircam/AFTER

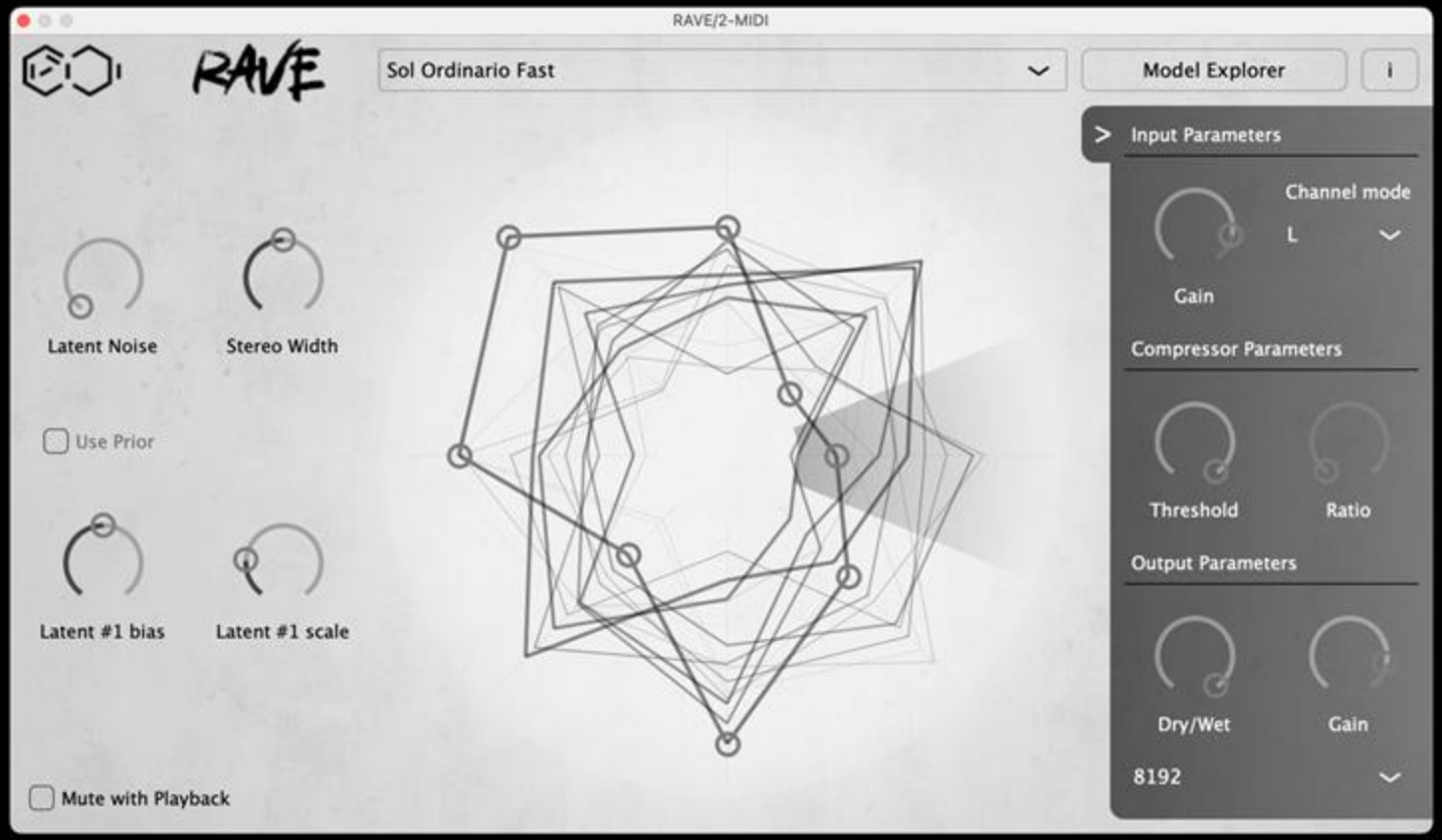
Simple training pipeline (similar to RAVE)

```
1.after prepare_dataset …
2.after train …
3.after export …
```

Identical training principles

Re-apply the same concepts as before to new context
Remember to always read the manual

RAVE VST


RAVE/2-MIDI
RAVE
Sol Ordinario Fast
Model Explorer
i
Input Parameters
Channel mode
L
Gain
Compressor Parameters
Threshold
Ratio
Output Parameters
Dry/Wet
Gain
8192
Latent Noise
Stereo Width
Use Prior
Latent #1 bias
Latent #1 scale
Mute with Playback

Other RAVE-based VSTs

Neutone - Morpho

https://neutone.ai/

Datamind - Combobulator

https://datamindaudio.ai/

ACIDS - Discord

We recently created a new discord for ACIDS Project

Join us if you have questions or want to share your own projects

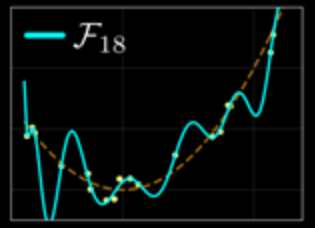
ACIDS - Discord

https://discord.gg/r9umPrGEWv

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

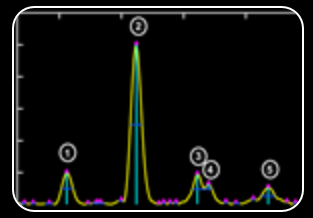
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models … with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above



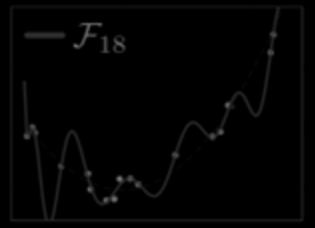
4

The present course is only an **introduction to get you started on these complex topics.**

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

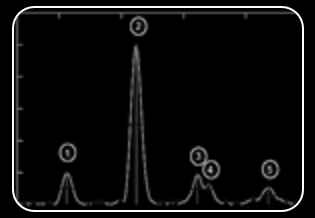
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models ... with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above



4

The present course is only an **introduction to get you started on these complex topics.**

Other applications (*discriminative*)

Genre: *Acid techno*
Tempo: *140 BPM*
Key: *B#*
Lyrics: *Avin' it*

Music Information Retrieval (MIR)

Music Genre Classification

Automatically classifying music genres from audio samples.

Nam, J et a. (2018). Deep learning for audio-based music classification and tagging. IEEE signal processing magazine, 36(1), 41-51
https://e-tarjome.com/storage/panel/fileuploads/2019-07-04/1562228921_E11422-e-tarjome.pdf

Beat and Tempo Detection

Analyzing music to detect beat patterns and tempo using deep learning.

Schreiber, H., & Müller, M. (2018). "A Single-Step Approach to Musical Tempo Estimation Using a Convolutional Neural Network." ISMIR.
https://www.tagtraum.com/download/2018_schreiber_tempo_cnn.pdf

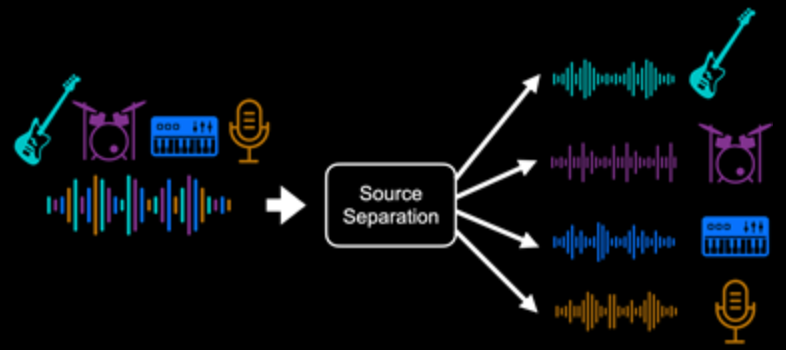
Automatic Music Tagging

Assigning genre, mood, or other descriptive tags to songs using deep learning.

Choi, K., Fazekas, G., Sandler, M., & Cho, K. (2016). "Convolutional Recurrent Neural Networks for Music Classification."arXiv:1609.04243.
https://arxiv.org/abs/1609.04243

Source Separation

Separating individual instruments or voices from a mixed audio signal.
Usual problem of frequency coverage and bandwidth
Recent models provide stunning results



Jansonn A. et al. (2017). "Singing Voice Separation with Deep U-Net Convolutional Networks." ISMIR.

Other applications (*discriminative*)

Music Recommendation

Using deep learning to create personalized music recommendations

Analyze users' preference and listening history.

Well-known *cold start* problem

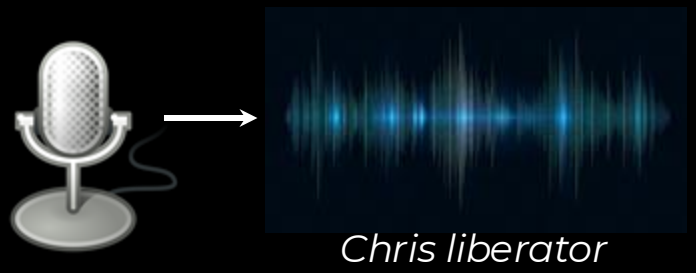
Aaron van den Oord et al. *"Deep Content-based Music Recommendation"* (2013)
https://proceedings.neurips.cc/paper/2013/file/b3ba8f1bee1238a2f37603d90b58898d-Paper.pdf

Query-by-Humming

Fingerprinting for efficient music retrieval in large databases.

Mapping the query and songs to a common reduced space

Provides efficient query-by-humming system



Chris liberator

Mostafa, N. et al (2017). A Note Based Query By Humming System Using Convolutional Neural Network. In Interspeech
https://www.isca-speech.org/archive_v0/Interspeech_2017/pdfs/1590.PDF

Audio-to-Score Conversion

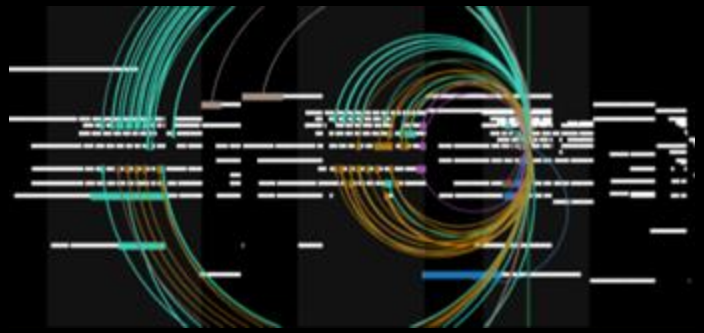
Transcribing audio recordings into sheet music using deep models.

Non-unicity of the solution and polyphonic problem

Almost solved for piano but still hard for orchestra

Hawthorne et al. (2018). "Onsets and Frames: Dual-Objective Piano Transcription." ISMIR 2018.
https://arxiv.org/abs/1710.11153

Other applications (*generative*)



Symbolic music generation

Generating symbolic music with advanced models

Handle long-range dependencies for coherent musical structures.

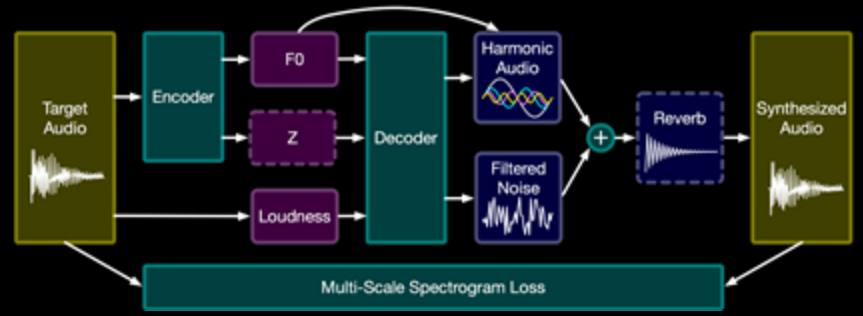
Huang, C. Z. A., & Du, S. (2020). "Pop Music Transformer: Generating Music with Rhythm and Harmony." arXiv:2002.00212.
https://arxiv.org/abs/2002.00212

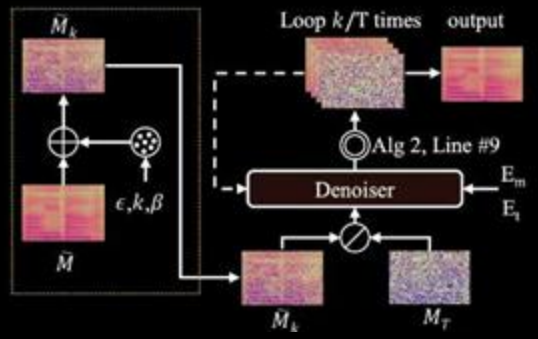
Neural Music Synthesis

Generating novel synthesized instruments and soundscapes

Current trends increasingly mixes traditional DSP elements

Engel, J. et al. (2020). "DDSP: Differentiable Digital Signal Processing." ICLR. 2020
https://openreview.net/forum?id=B1x1ma4tDr





Text-to-singing

Singing voice generation based on input text

Also needs to handle prosody and emotional response

Liu, J. et al. (2022). Diffsinger: Singing voice synthesis via shallow diffusion mechanism. In AAAI 2022.
https://ojs.aaai.org/index.php/AAAI/article/view/21350/21099

Algorithmic DJing

Analyze structure to create seamless transitions between songs

Enables automated DJing and playlist creation.

Huang, J. et al. Modeling the compatibility of stem tracks to generate music mashups. In AAAI 2021
https://ojs.aaai.org/index.php/AAAI/article/view/16092/15899

Two major proposals (for today)

We are going to see two different ways of approaching this

1. Using the nn~ scripting system (**for Pytorch models**)
2. Creating a bridge using OSC (**for any model**)

1 - **Torchscript exported (.ts)**  

Host for deep models
Kind of modern VST host for deep models

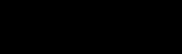
2 - **Python model**  OSC 

Interoperability
Setting up a communication bridge

Two major proposals (for today)

We are going to see two different ways of approaching this

1. Using the nn~ scripting system (**for Pytorch models**)
2. Creating a bridge using OSC (**for any model**)

1 - **Torchscript exported (.ts)**   

Host for deep models
Kind of modern VST host for deep models

2 - **Python model**   OSC 

Interoperability
Setting up a communication bridge

nn~ - Scripting

Musical side

MaxMSP

External

nn~

Host for deep models

Kind of a modern VST host but for deep learning models

Python

Torchscript exported (.ts)

Model side

RAVE

Timbre transfer

Descriptor control

vschaos

Vintage neural

+?

Any Pytorch (and basic Python) models can fit in nn~

[Demonstration - The new *scripting library* of nn~]

nn~ - Scripting

Musical side

MaxMSP

External

Host for deep models

Kind of a modern VST host but for deep learning models

Python

Torchscript **exported (.ts)**

Main steps

1. Find a **Pytorch source code** you want to use
2. Write a wrapper class using `nn_tilde.Module`
3. In that class write functions you want to call

   For instance `forward, encode, decode`

1. Register those functions using `register_method()`
2. Finally script and save the Torchscript model (.ts)

Scripting in MaxMSP - Basic example

Write wrapper (**nn_tilde.Module**)

```python
class ScriptFeature(nn_tilde.Module):

    def __init__(self, config):
        """ Initialize our object """
        super().__init__()
        # OSC library objects
        self.register_method(...)

    def forward(self, config):
        """ Main call for our model """
            ...
        return out
```

Register functions (**in init**)

```python
        # OSC library objects
        self.register_method(
            "forward",
             in_channels=2,
             in_ratio=BLOCK_SIZE,
             out_channels=1,
             out_ratio=1,
             input_labels=["pitch", "loudness"],
             input_labels=["pitch", "loudness"],
             input_labels=["pitch", "loudness"])
```

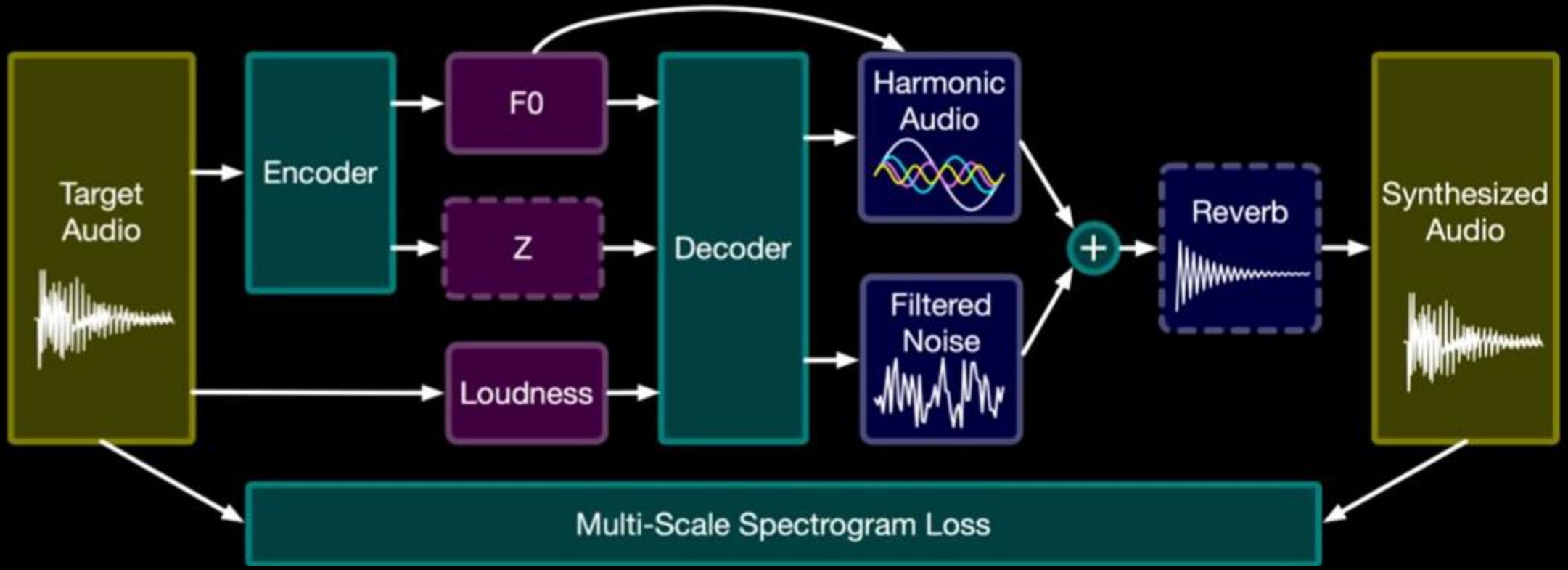
Write functions to call (**forward**)

Final script and save

```python
script_model = ScriptFeature(...)
script_model.export_to_ts("./exports/ddsp.ts")
```

Final example – DDSP

Architecture overview



Scripting in MaxMSP - DDSP

Write wrapper (`nn_tilde.Module`)

```
1.  class ScriptDDSP(nn_tilde.Module):

2.      def __init__(self, config):
3.          """ Initialize our object """
4.          super().__init__()
5.          # OSC library objects
6.          self.register_method(...)

7.      def forward(self, config):
8.          """ Main call for our model """
9.              ...
10.         return out
```

Register functions (**in init**)

```
1.          # OSC library objects
2.          self.register_method(
3.              "forward",
4.              in_channels=2,
5.              in_ratio=BLOCK_SIZE,
6.              out_channels=1,
7.              out_ratio=1,
8.              input_labels=["pitch", "loudness"],
9.              input_labels=["pitch", "loudness"],
10.             input_labels=["pitch", "loudness"])
```

Write functions to call (`forward`)

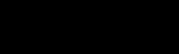
Final script and save

```
1.  script_model = ScriptDDSP(...)
2.  script_model.export_to_ts("./exports/ddsp.ts")
```

Two major proposals (for today)

We are going to see two different ways of approaching this

1. Using the nn~ scripting system (**for Pytorch models**)
2. Creating a bridge using OSC (**for any model**)

1 - **Torchscript exported (.ts)**   

Host for deep models
Kind of modern VST host for deep models

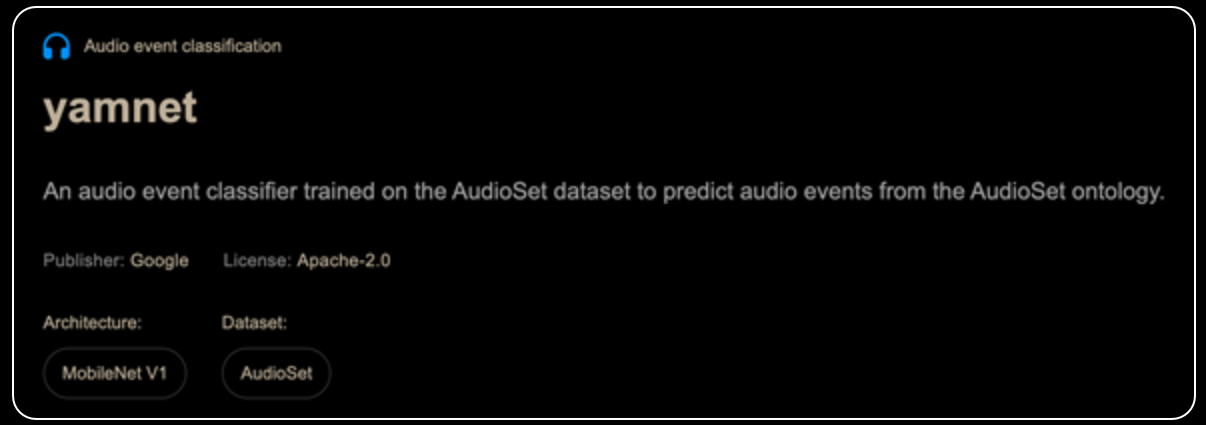
2 - **Python model**   OSC 

Interoperability
Setting up a communication bridge

Using existing code (Python)

Given existing (trained) deep model

- Lots of trained model exist online
- We can download and use directly
- Allows to create **personalized controls**
- Here we will create **audio classification**



How to use it and apply it ?

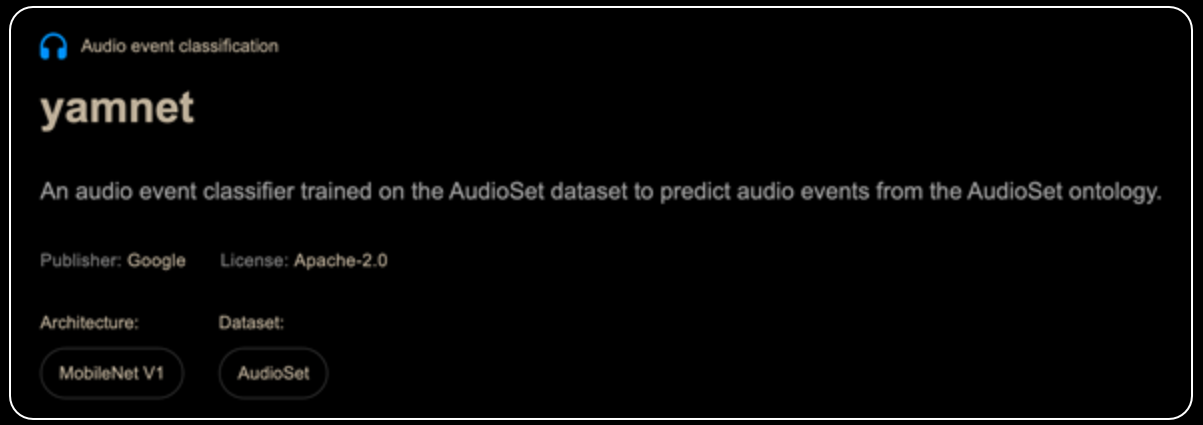
https://tfhub.dev/google/yamnet/1

1. Think about your problem and what you need
   a. Here, imagine you want to know the class (type) of a sound
2. Devise the overall processing workflow
3. Find a pre-trained model online that fits your purpose
4. Check the documentation to find the model behavior (and example code)
5. Perform a series of **testing code** on **toy data**
6. Incorporate the model inside your workflow

Using existing code (Python)

Given existing (trained) deep model

- Lots of trained model exist online
- We can download and use directly
- Allows to create **personalized controls**
- Here we will create **audio classification**



How to use it and apply it ?

https://tfhub.dev/google/yamnet/1

```
1. import tensorflow_hub as hub
2. import numpy as np

1. # Load the model.
2. model = hub.load('https://tfhub.dev/google/yamnet/1')
3. # Input: 3 seconds of silence as mono 16 kHz waveform samples.
4. waveform = np.zeros(3 * 16000, dtype=np.float32)
5. # Run the model, check the output.
6. scores, embeddings, log_mel_spectrogram = model(waveform)
```

Classification scores Embedding dimensions

[Live coding demonstration]

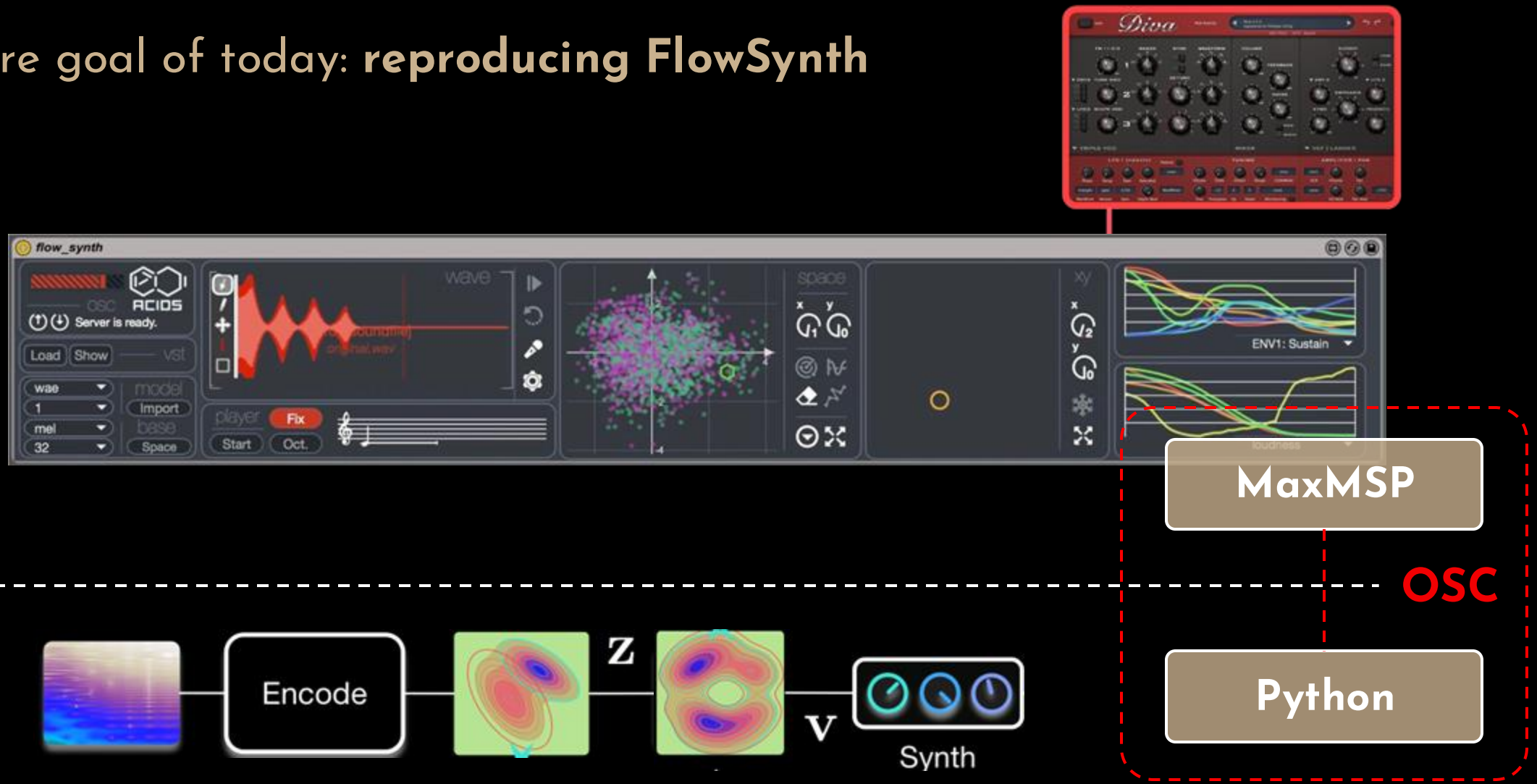
FlowSynth - Demo video

https://github.com/acids-ircam/flow_synthesizer

https://www.youtube.com/watch?v=UufQwUitBIw

Course overview

Core goal of today: **reproducing FlowSynth**



Introduction to OSC

Open Sound Control (OSC)

- Protocol for communication among computers, synthesizers, and devices.
- Modern, **open-ended**, and flexible, often used where MIDI feels too constraining.
- Supports a wide range of (standard and adaptable) **data types**.

Properties of OSC

- Transport-independent and message-based lightware protocol
  - Client / Server design (star network) following **UDP protocol**
  - Based on socket interface communication
  - Requires the IP address of the server to communicate
  - Can be used in **localhost (internal communication between programs)**
- Free implementations available for all kind of systems and languages
- Extremely useful in music, but also sensor arrays, image processing, video, …
- However, **need to define your own protocol and set of messages**
  - Implemented using the set of domain-relevant knowledge
  - **Need to know this protocol beforehand**

Properties of OSC

Why using OSC ?

- **Interoperability**: Platform-independent and supported in lots of software
  Ideal for setups involving different types of devices
- **Flexibility**: Allows to define your own types of messages (unlike MIDI).
  Create messages that are tailored to your specific needs.
- **Networking**: Works over any communication protocols (TCP, UDP).
  Well-suited for both local networks or internet.

Compared to MIDI

- No fixed hardware wire transmission format required
- Can be used over WiFi, Internet, USB, 4G, etc...
- Unlimited extensibility and **unlimited bandwidth** (Ethernet limit **100 Gb/s**)

Available datatypes

- Raw sensor data (RGB images)
- Raw audio (waveform) data
- Time series features
- Atomic types

Writing your own protocol

Decomposing an OSC message

An OSC message consists of

- OSC Address Pattern (e.g., /filter/frequency)
- OSC Argument (e.g., 440.0).

Example

OSC Address Pattern: /synth/volume

OSC Argument: 0.75

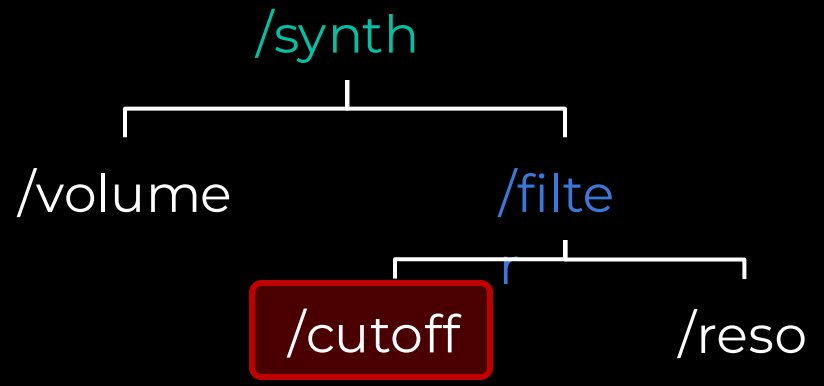
Makes OSC highly readable and easy to debug

Set the volume of a synthesizer to 75%.

You have to write your own protocol (message types)

Steps for creating your protocol

1. Decide on your address pattern and logic
   a. OSC messages follow a hierarchical taxonomy
2. List the set of messages you need to implement
3. Define the number of OSC arguments
4. Detail the behavior of each message
5. Implement corresponding functions



Full message

/synth/filter/cutoff 44100

Introduction to OSC

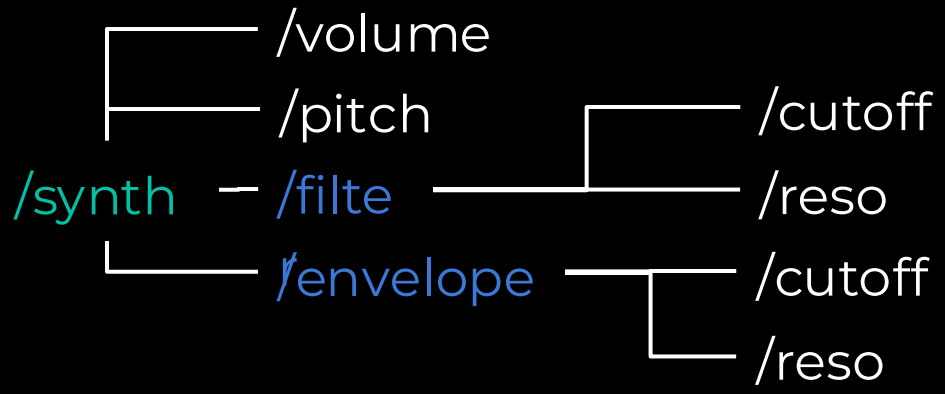
Basic datatypes

Int32: 32-bit big-endian two's complement integer
Float32: 32-bit big-endian IEEE 754 floating point number
OSC-string: Sequence of ASCII characters finished by a null
OSC-blob: A size count, followed by that many 8-bit bytes of data

Goal of today

- Leveraging OSC for using advanced Python code inside MaxMSP
- Enables more advanced and customizable control than MIDI messages.

We will start with a very simple **synthesizer control protocol**



Max handles reactive synthesizer and “asks questions” to Python for control

Python to MaxMSP

Exercise #0: Conditional message sending

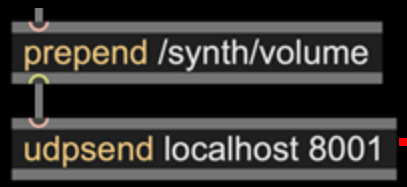
1. Write an OSC server in Python (using python-osc) to send and receive data from MaxMSP.
2. Upon reception of a `/synth/update` message
   a. Computes new values for the synth parameters
      i. At least `/synth/pitch` and `/synth/filter`
   b. **Sends the computed values back to MaxMSP**

MaxMSP Side

Python side

Send message to Python

Client



Server

```
1. dispatch = dispatcher.Dispatcher()
2. dispatch.map("/synth/volume", compute)
```

[Perform advanced computation]

```
1. def compute(addr, args):
       ...
1. client = SimpleUDPClient("localhost", 8000)
2. client.send_message("/filter/frequency", ..)
```

Receive message

Server

Distribute to objects

Client

Python and OSC

Introduction

- **python-osc** is a library for handling OSC using the UDP protocol.
- Official python-osc GitHub repository for more details: python-osc

Example code to **set up OSC server**

```
1. from pythonosc import dispatcher, osc_server

2. server = osc_server.ThreadingOSCUDPServer(("localhost", 8000), dispatch)
3. print("Serving on {}".format(server.server_address))
4. server.serve_forever()
```

Adding **message filter** (waiting messages on address **/filter/frequency**):

```
1. def print_filter_freq(unused_addr, freq):
2.     print(f"Filter frequency: {freq}")

3. dispatch = dispatcher.Dispatcher()
4. dispatch.map("/filter/frequency", print_filter_freq)
```

Function **print_filter_freq** gets called whenever a **/filter/frequency** message is received

Python and OSC

Sending OSC messages from Python

Example Python code that **sends an OSC message**:

```
1. from pythonosc.udp_client import SimpleUDPClient

2. client = SimpleUDPClient("localhost", 8000)
3. client.send_message("/synth/filter/cutoff", 440.0)
```

In this code, we create an OSC client that sends messages to `localhost` on port `8000`.
Send message with address `/synth/filter/cutoff` and argument `440.0`.

Exercise #1: Conditional message sending

1. Write an OSC server in Python (using python-osc) to send and receive data from MaxMSP.
2. Upon reception of a `/synth/update` message
   a. Computes new values for the synth parameters
      i. At least `/synth/pitch` and `/synth/filter`
   b. Sends the computed values back to MaxMSP

MaxMSP and OSC

Introduction to OSC in MaxMSP

- MaxMSP provides two objects to handle OSC communication.
- The **`udpreceive`** and **`udpsend`** objects come prepackaged.

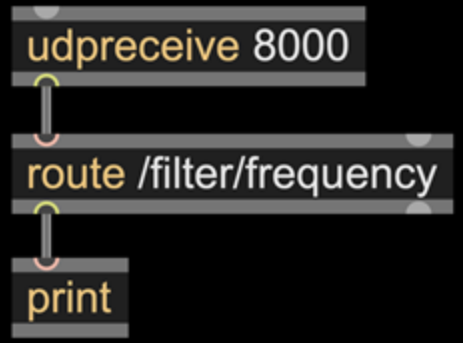
Receiving OSC messages

- Create **`udpreceive`** with specified **port**
- Use **`route`** object to filter messages.

Listens for OSC messages on 8000

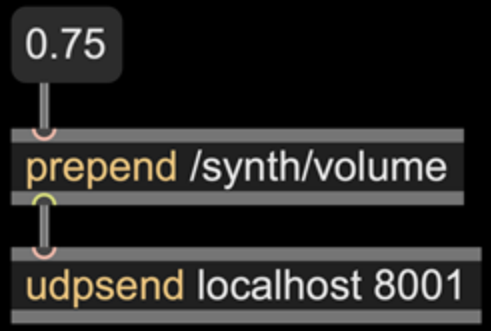
Route messages **`/filter/frequency`**

Print to Max Console.



Sending OSC messages

- Create **`udpsend`** with target IP and port
- Send messages with correct prefix



Create message **`/synth/volume 0.75`**

Send to **`localhost`** *on port 8000.*

We can now finish our exercise on controlling a synthesizer from Python

Python and OSC

Full server architecture

```
1.  class OSCServer(object):
2.      def __init__(self, in_port, out_port, ip='127.0.0.1', *args):
3.          """ Initialize our object """
4.          super(OSCServer, self).__init__()
5.          # OSC library objects
6.          self.dispatcher = dispatcher.Dispatcher()
7.          self.client = udp_client.SimpleUDPClient(ip, out_port)
8.          # Bindings for server
9.          self.init_bindings(self.osc_attributes)
10.         self.server = osc_server.BlockingOSCUDPServer((ip, in_port), self.dispatcher)
11.         # Server properties
12.         self.in_port, self.out_port, self.ip = in_port, out_port, ip
13.         ...
```

Efficiency

1. Our object **contains both client and server**
2. All of our behaviors **will be defined as class methods**
3. Also allows to have a **persistent server state**

Python and OSC

Full server architecture

```
1.  class OSCServer(object):
2.          ...
3.      def stop_server(self, *args):
4.          '''stops the server'''
5.          self.client.send_message("/terminated", "bang")
6.          self.server.shutdown()
7.          self.server.socket.close()

8.      def run(self):
9.          '''runs the server'''
10.         self.server.serve_forever()

11.     def init_bindings(self, osc_attributes=[]):
12.         '''Here we define every OSC callbacks'''
13.         ...

14.     def send(self, address, content):
15.         '''global method to send a message'''
```

Clean stop of server

Allows to avoid blocking the corresponding port

Initialize all bindings

Set all messages responses

Python to MaxMSP

Exercise #0: Conditional message sending

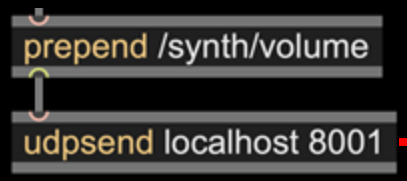
1. Write an OSC server in Python (using python-osc) to send and receive data from MaxMSP.
2. Upon reception of a `/synth/update` message
   a. Computes new values for the synth parameters
      i. At least `/synth/pitch` and `/synth/filter`
   b. **Sends the computed values back to MaxMSP**

MaxMSP Side

Python side

Send message to Python

Client



Server

```
1. dispatch = dispatcher.Dispatcher()
2. dispatch.map("/synth/volume", compute)
```

Receive message

Server

[Perform advanced computation]

```
1. def compute(addr, args):
       ...
1. client = SimpleUDPClient("localhost", 8000)
2. client.send_message("/filter/frequency", ..)
```

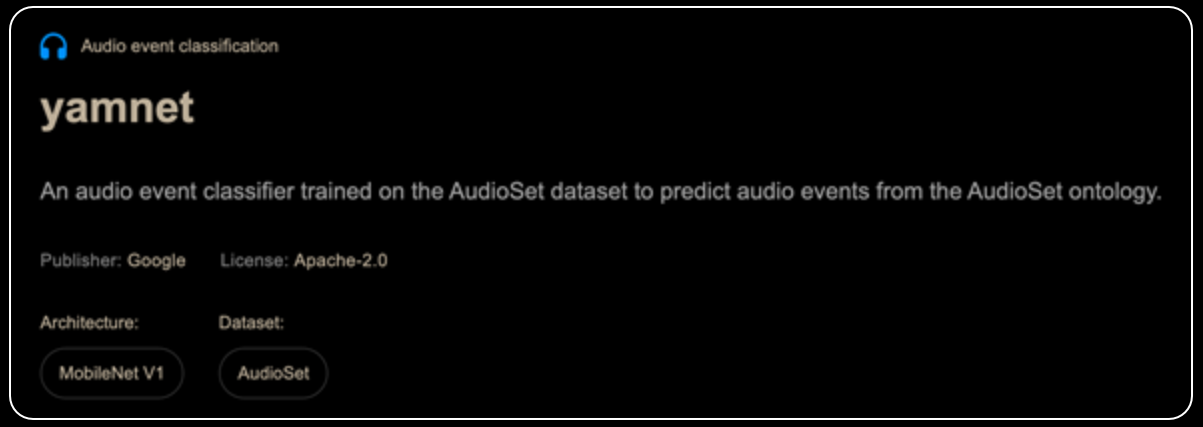
Client

Distribute to objects

Final exercise #1 - Audio classification

Putting everything that we have seen into real use

https://tfhub.dev/google/yamnet/1



- We loop back to the **Course 01 exercise**
- We will use our audio classification
- Try to connect it to MaxMSP
- Goal is to send audio from MaxMSP
- And classify it live in Python
- Then use the result for processing

Example application proposal: Smart AI-based processing

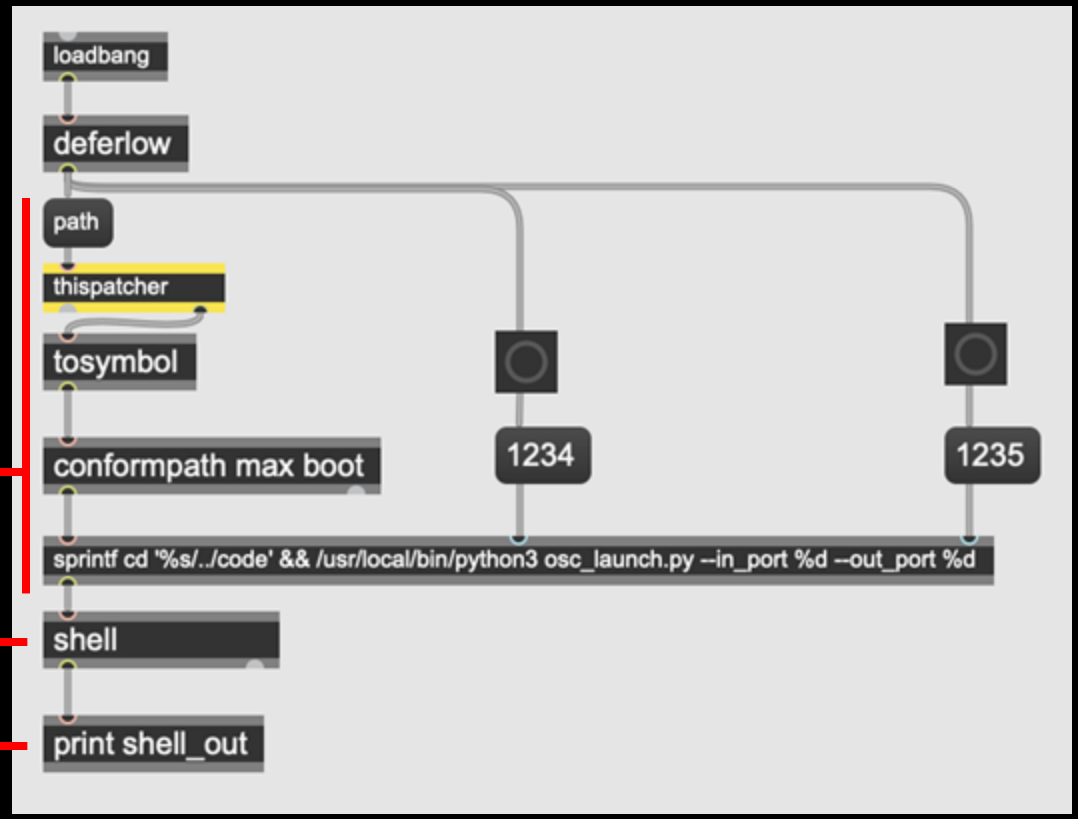
1. We will take input sounds in MaxMSP
2. Send them to Python through OSC (through file writing !)
3. Perform audio classification with YamNet
4. Then send the classification result back
5. Filter the results in MaxMSP
6. Use specific sounds as triggers
    a. Example: If drums, apply different type of audio processing

Launching the server

How to automatically launch the Python server ?

jeremybernstein / **shell**

https://github.com/jeremybernstein/shell/releases/tag/1.0b3



Create a **command-line call**

Launch it

Check the output

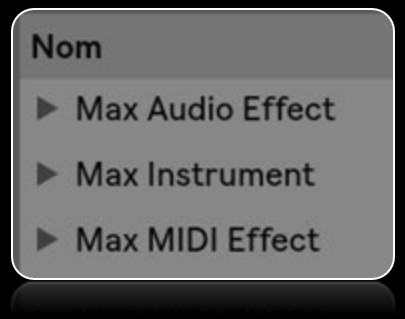
Only use when your code is ready (performance), otherwise hell to debug

Max4Live

What is Max4Live ?

- Collaboration between Cycling '74 and Ableton (before buyoff)
- Direct integration of MaxMSP possibilities inside Ableton Live
- Infinite possibilities for devices that extend the functionality of Live.

Types of Max4Live devices



Audio Effect: Processes audio signals (transform input audio in real-time).

Instrument: Generates sound signal from MIDI input.

MIDI Effect: Processes MIDI data (generate or transform MIDI notes and control).

Max for Live Essentials: Collection of audio and MIDI effects, synthesizers and control in Max4Live
Max for Live Essentials

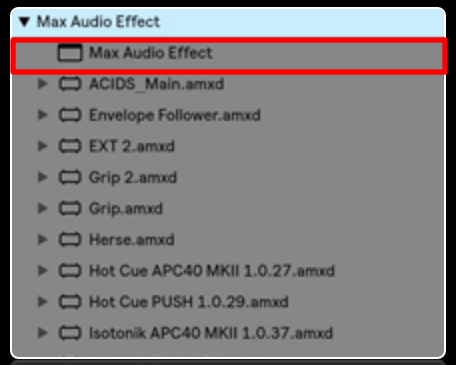
Max4Live API

- Allows to control Live's interface from within Max4Live devices.
- Move faders, change parameters, trigger clips and more.
- Accessible through 'live.object', 'live.path', 'live.observer' and other

Max4Live - Example

We will create a basic Max4Live device

Drag



Click this

First Max4Live exercise

- First a simple Max4Live device that adjusts the volume of the track it's inserted on.
- The Live API is used to get the current track and control its volume.
- We will develop devices that can leverage deep learning models.

Learning resources

- Live API: Live API overview
- Max4Live: Ableton Max4Live
- Max4Live tutorials: Learning Max with Max4Live

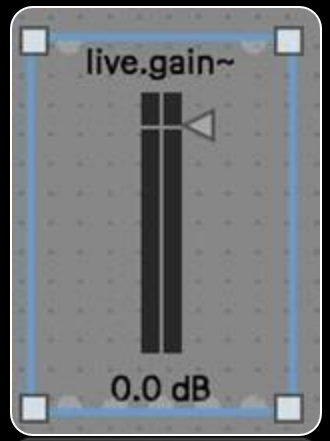
Using live objects (gain~)

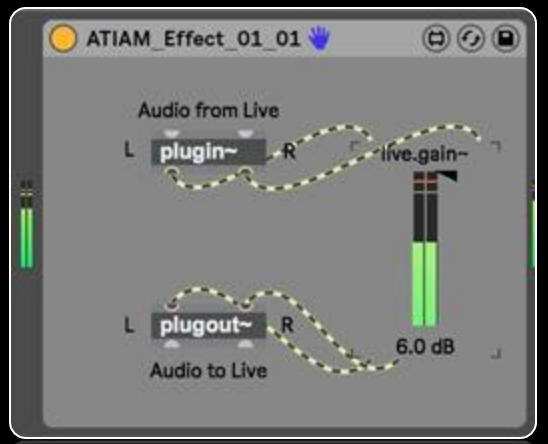
Max4Live device basics

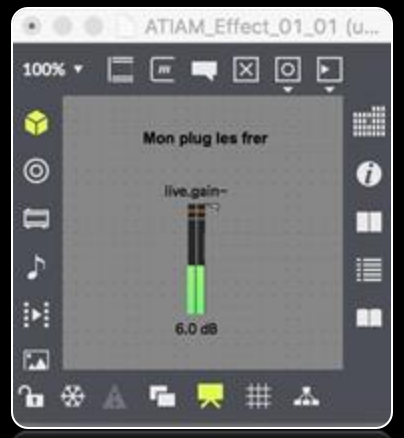
- Max4Live devices are built simply by patching objects in MaxMSP.
- Allows to extend Live with all Max functions (signals, computations, or control).

Building a simple device

- We will create a simple audio effect that alters the amplitude of incoming audio signal.
- We will connect a `live.gain~` to the `plugin~` and add a title to our amazing device







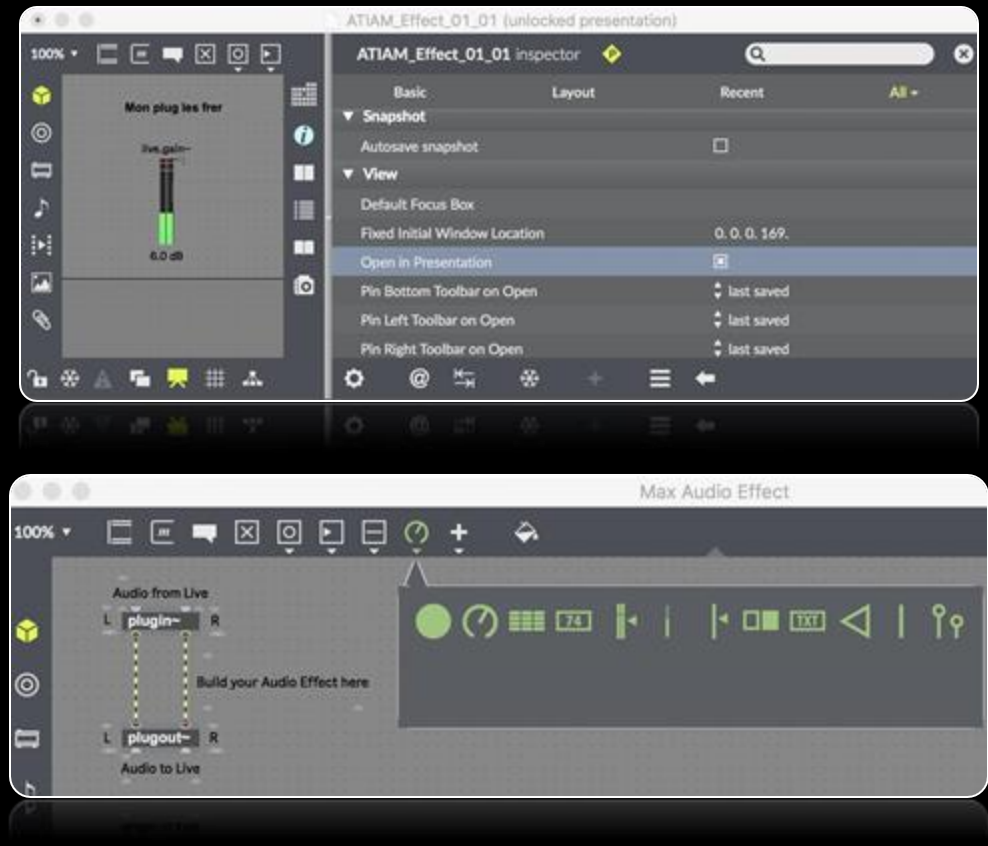
Shortcuts :

N for adding a new object (B bang, M message, C comment)
Cmd+Maj+P for adding to presentation
Cmd+Y for auto-aligning

All objects from Live

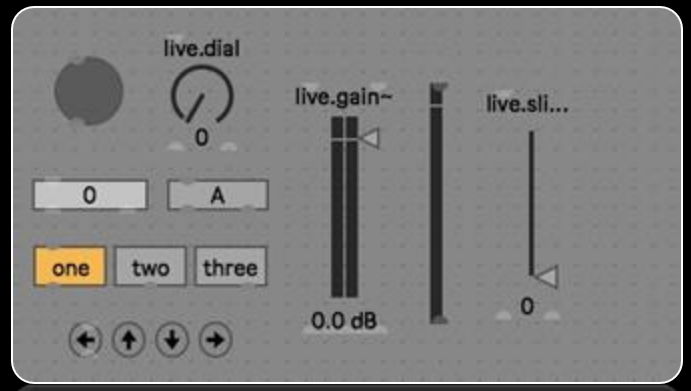
Presentation and device aesthetics

Open the device in presentation (Inspector)



Ableton Live GUI objects

Access to all of Live GUI objects in Max



All objects start with **`live.*`**

Check the documentation

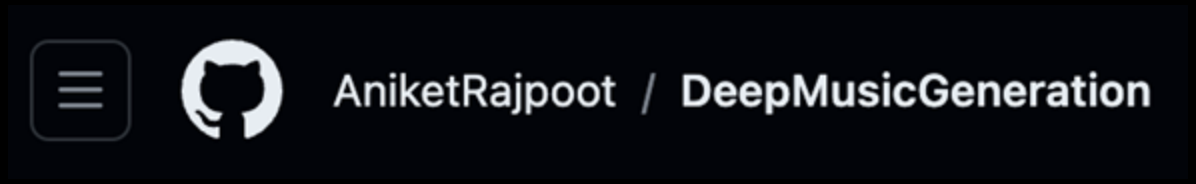
Also note that any graphical object from MaxMSP is valid !

Exercise #2 - Music generation

Another exercise for going further (symbolic generation)

- Consider a deep model that generates new melodies based on user input.
- User provides a **seed melody** through Max4Live, sent to Python via OSC.
- Python script generates a new melody using the deep model
- Sends it back to Max4Live for playback or further processing.

Example application proposal: AI-based melody generation



https://github.com/AniketRajpoot/DeepMusicGeneration

- Package for deep music generation
- Allows to generate infinite melodies
- Will be sent back to MaxMSP

Advanced topics

Advanced use of Max4Live

Max Jitter Tutorial	**Jitter**: Useful for video, 3D graphics, and large sets of numbers.
Gen~ Tutorial	**Gen~**: High-performance, sample-accurate DSP code within Max
Multichannel Audio	**Multichannel**: Max4Live devices can handle multichannel audio signals.

Advanced deep learning concepts

Generative Models	Recent trend used to generate new musical ideas.
Real-time Audio Processing	Dive deeper into the world of real-time audio processing with deep learning.

Other ideas for merging Max4Live and deep learning

Wekinator: Real-time interactive machine learning that can be connected to MaxMSP via OSC.

Madmom: audio signal processing library for MIR written in Python

Max Data Sonification Toolkit: A toolkit for real-time sonification of data in Max.

Magenta Studio: A suite of machine learning music models available as Ableton Live devices.

Beat Blender: Uses a variational autoencoder to interpolate between different drum beats.

Going further

Advanced MaxMSP

Cycling '74 tutorials

- Consider diving deeper and explore more complex objects
- Learn how to use Gen~ for more complex audio processing
- Use Jitter for video and matrix data processing.

Advanced Python

Real Python tutorials

- Versatile language with a rich ecosystem of libraries.
- Learn data analysis with pandas, machine learning with scikit-learn,
- Web applications with Flask or Django.

Advanced Max4Live

Live API overview

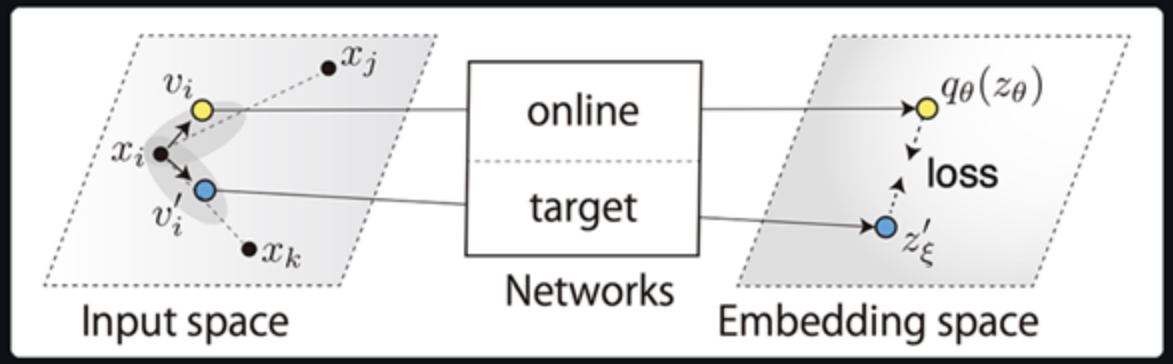
- Explore how to use the Live API to deeply integrate your devices with Ableton Live.
- OSC can be used to communicate with a wide variety of software and hardware.
- Simple and efficient communication with any software (PureData, SuperCollider)

Keep experimenting and learning, and most importantly, have fun!

Using existing code (Python)

A better embedding example (BYOL-A)

nttcslab / byol-a



BYOL for Audio: Exploring Pre-trained General-purpose Audio Representations

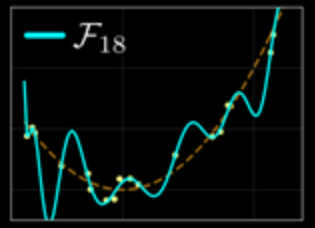
This is a demo implementation of BYOL for Audio (BYOL-A), a self-supervised learning method for general-purpose audio representation, includes:

- Training code that can train models with arbitrary audio files.
- Evaluation code that can evaluate trained models with downstream tasks.
- Pretrained weights.

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

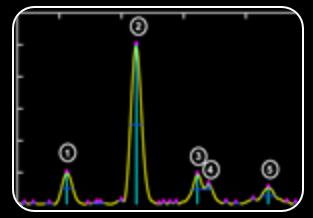
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models … with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above



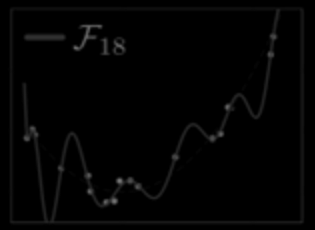
4

The present course is only an **introduction to get you started on these complex topics.**

Detailed program

A deeper overview of the skills that we aim to develop

1



Creative machine learning

Mathematical theory, properties of deep models, using GitHub code

Skills: Mathematics (basic), shell, Python, Github

Using RAVE and training models

Using existing models and how to train them on your own data

Skills: Shell, Python, Github, Max4live, Ableton

2

3

Embedding any model from Github or Huggingface

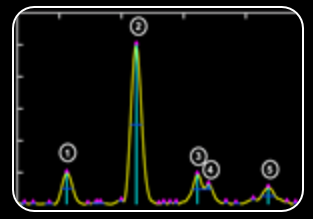
Developing your own approach to AI composition

Skills: Python, MaxMSP, Max4Live, Github

Controling deep models ... with deep models

How to push the boundaries of generation

Skills: Being a bit crazy and all of the above

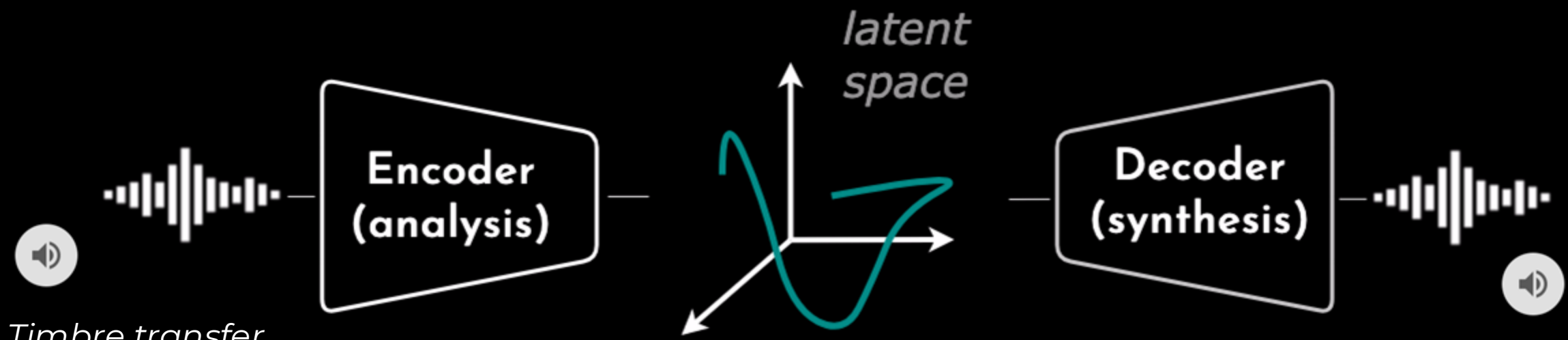


4

The present course is only an introduction to get you started on these complex topics.

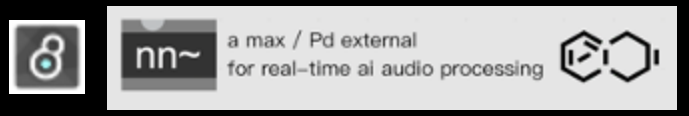
RAVE in its typical use case

Fast and high-quality raw waveform synthesis



Timbre transfer

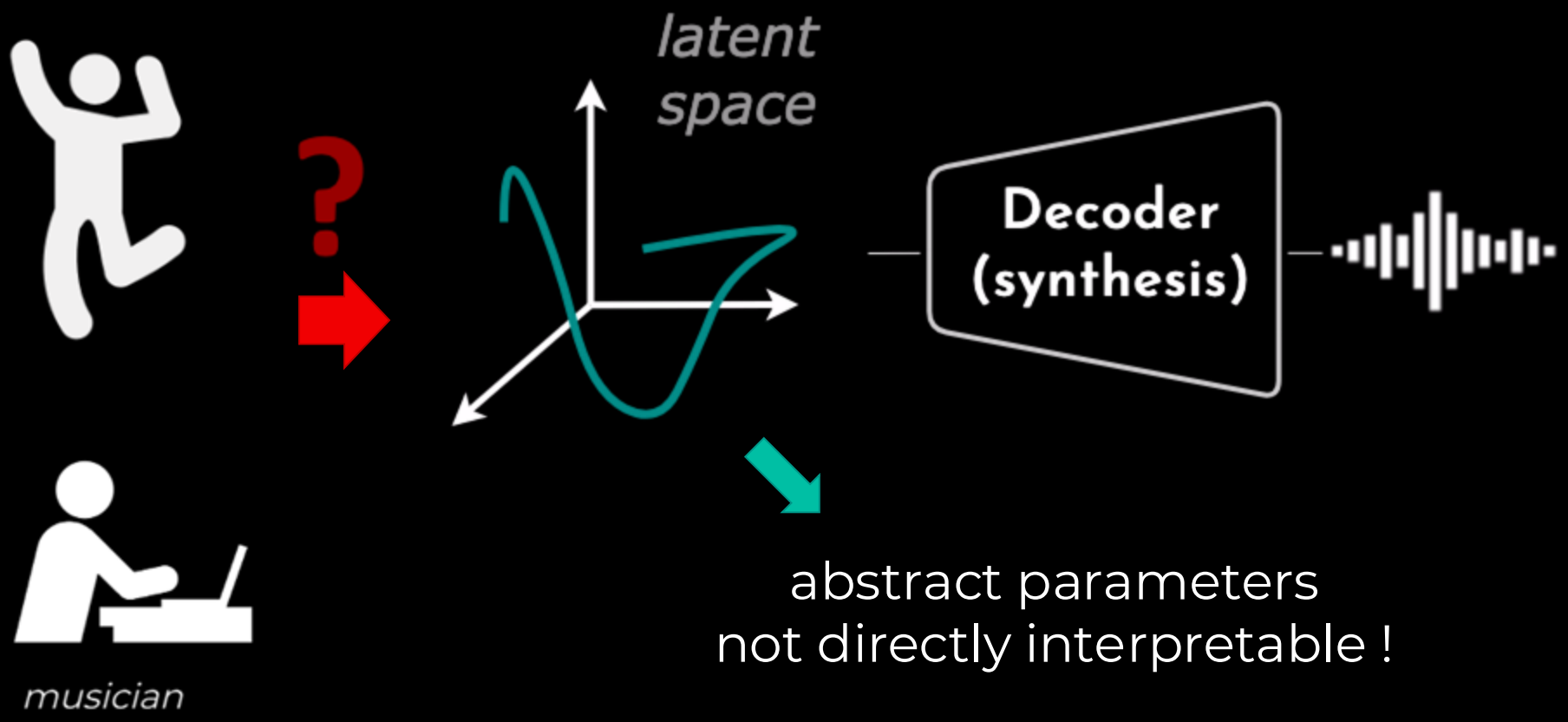
musician



https://github.com/acids-ircam/nn_tilde

Problem statement

How models can be «creatively» explored with ***embodied interaction*** ?



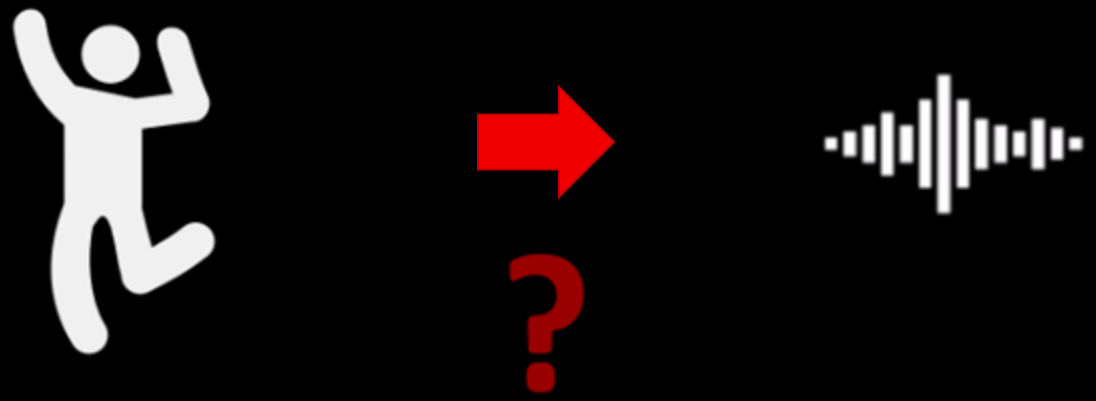
abstract parameters
not directly interpretable !

Art-research collaboration

Sarah Nabi and Marie Bruand

"body as instrument"

explore the musicality of the moving body



movement-based musical interaction
through *embodied exploration*

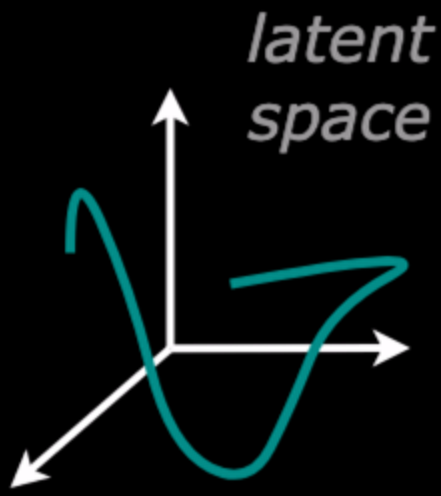
PRÉLUDE
@Art festival
NUIT BLANCHE
PARIS 2023

Objectives

ARTISTIC

1. How to offer new perspectives on movement / sound ?
2. How generative AI could bridge dance, music & tech field ?

“listen”?

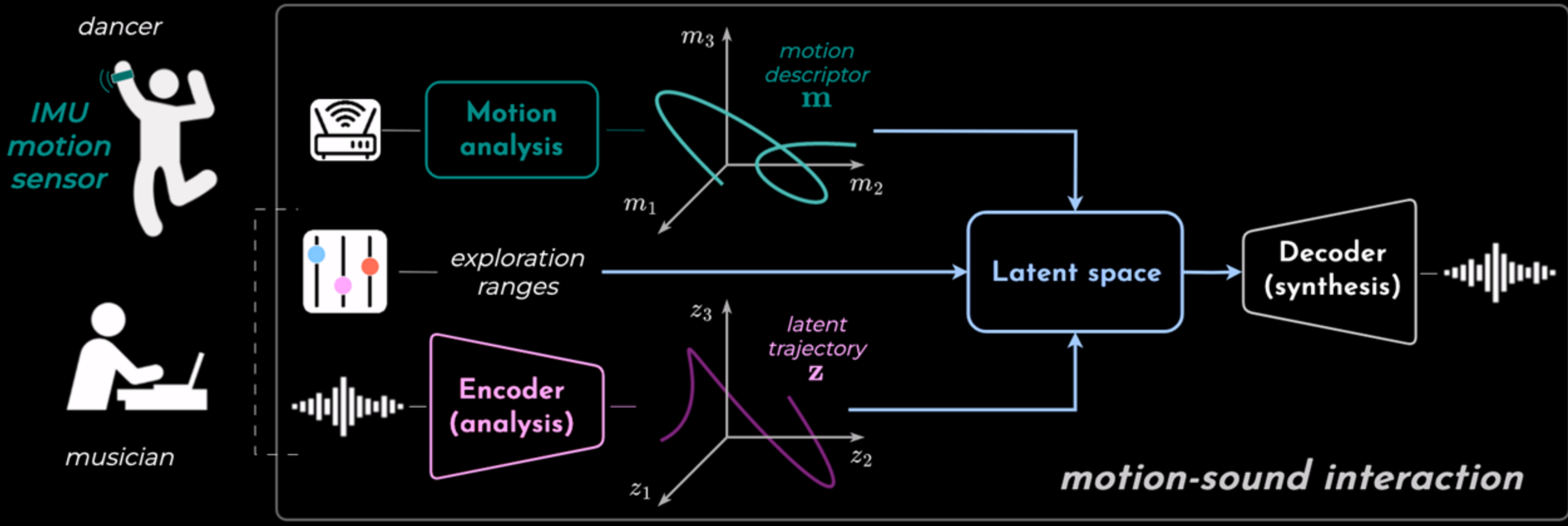


“watch”?

RESEARCH

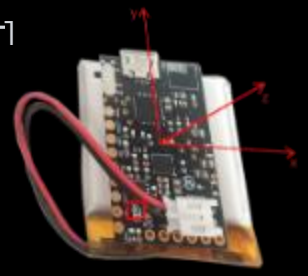
1. How to associate motion sensor and neural synthesis ?
2. How this system be explored, learned and played ?

Our motion sound-interactive system



Technical tools

- motion sensors: R-IoT[1] (accelerometers + gyroscopes)



- patch Max/MSP
- real-time movements analysis library: Mubu[2]
- synthesis model: RAVE[3] (nn~[4])

1. https://ismm.ircam.fr/riot/
2. https://ismm.ircam.fr/mubu/
3. https://github.com/acids-ircam/RAVE
4. https://github.com/acids-ircam/nn_tilde

AI-Based detection

Several objects exists for real-time recognition and musical use

Embed deep AI models from C++ into MaxMSP for use in Ableton Live

MediaPipe

- Hand detector
- Pose detector
- People detector

AI-Based detection

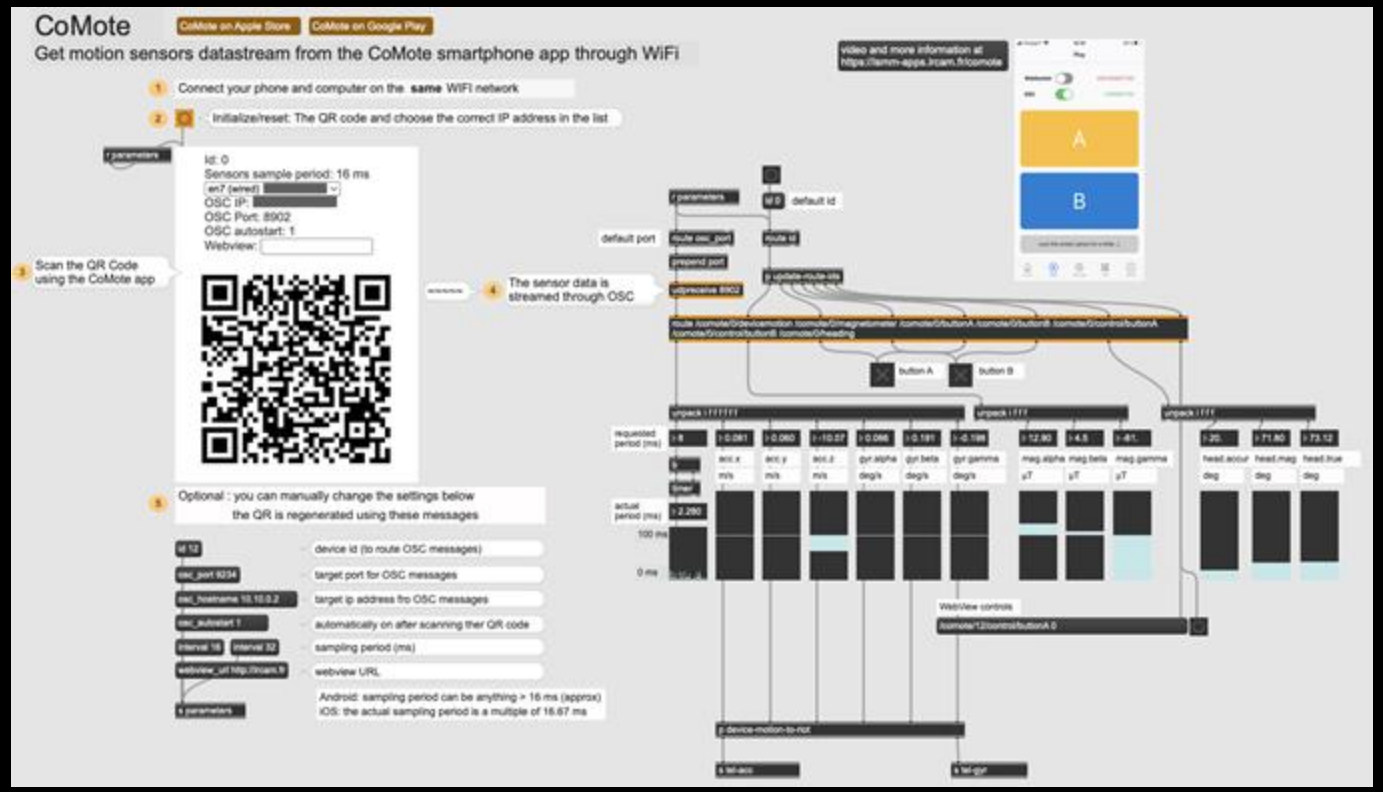
[DEMO TIME]

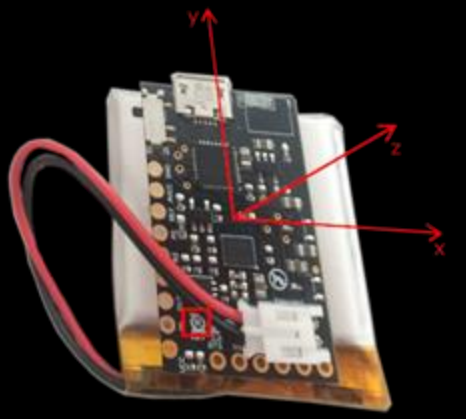
/Users/esling/Sound/experiments/tracking/mediapipe

Using smartphone's IMU sensors instead of R-IoTs

Here, we rely on the **CoMote** package developed by the ISMM team at IRCAM.

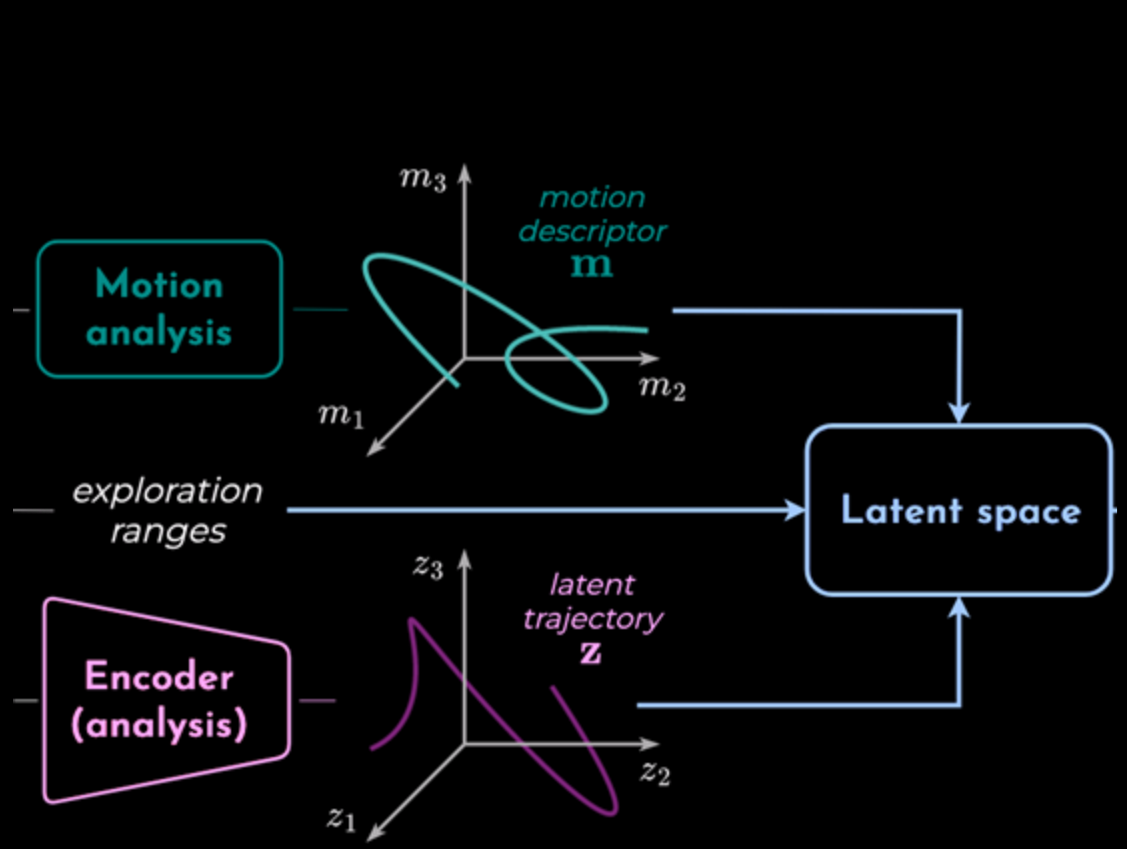
https://github.com/ircam-ismm/embodied-latent-exploration/tree/main/code/smartphones-imu-sensors

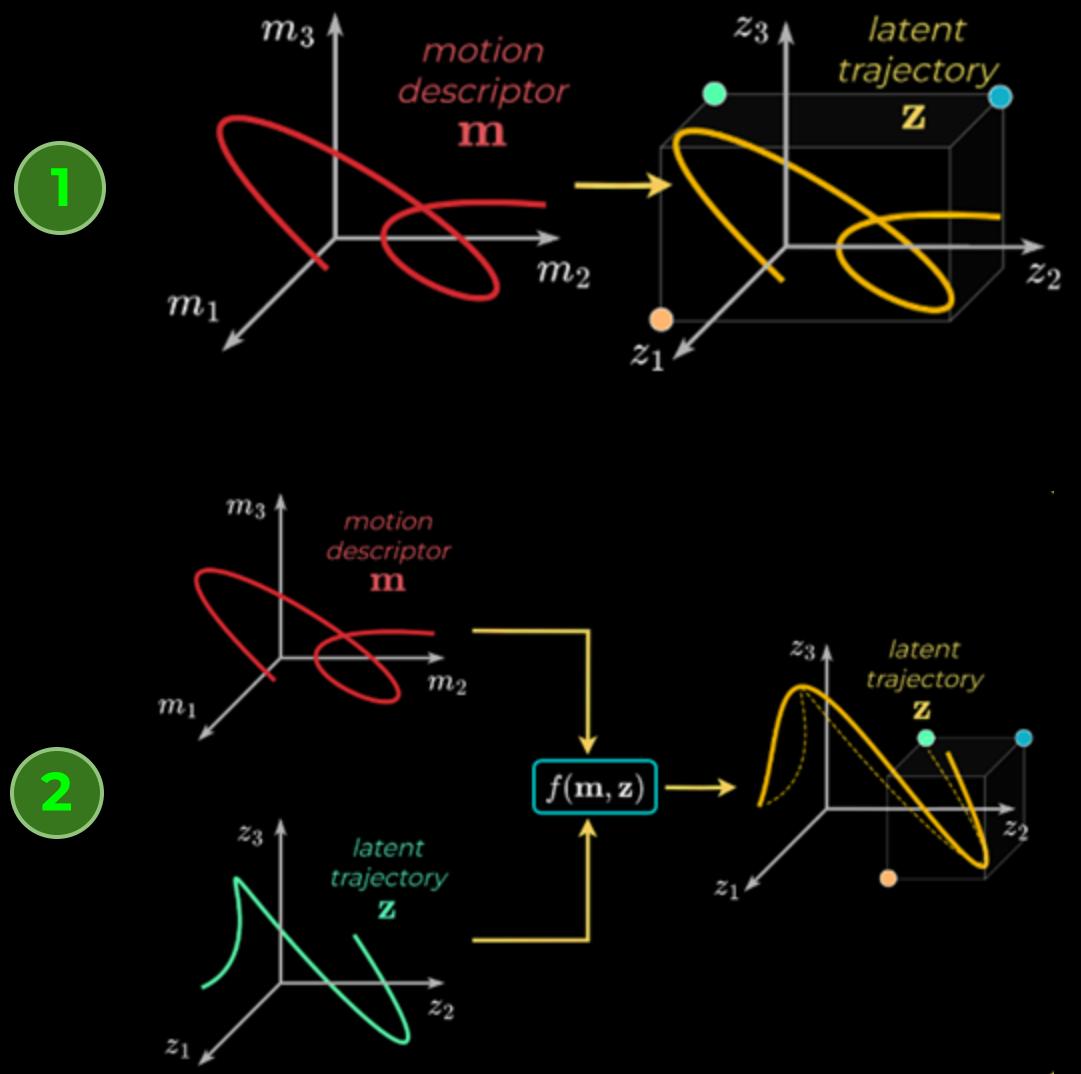




https://ircam-ismm.github.io/misc/riot-devicemotion-compatibility.html

Our motion sound-interactive system





Q&A Session

Thank you for your attention !

Pr. Philippe Esling
esling@ircam.fr

Pre-requisites (mathematical)

Notions required but not covered in this course

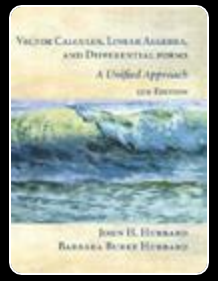
Resources to help you out if you feel behind on these topics ❤

Calculus

Limits and infinitesimals
Differential calculus
Integral calculus

Linear algebra

Vector spaces
Matrix calculus
Geometry and analysis



Hubbard J. & Hubbard B.B
Vector Calculus, Linear Algebra, and Differential Forms: A Unified Approach (5th edition)

- Linear algebra slides
- Probability revision slides
- Statistics course notes
- Sampling pages
- Matrix Cookbook

3 Blue 1 Brown

Essence of linear algebra
Essence of calculus

Programming

Python basics
Functional notions
Object-oriented prog.

Data analysis

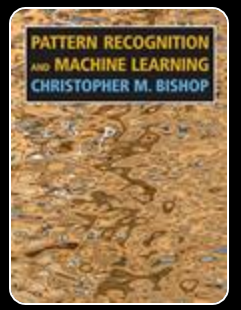
Data visualization
Basic filtering, processing

codewars

codecademy

Machine learning bible

Christopher M. Bishop
Pattern recognition and machine learning



Understanding philosophy

Coding principle

Recommend exploring all concepts by

1. Deriving the mathematical bases ourselves
2. Implementing low-level aspects directly as well
3. Moving up to higher-level (simpler) implementation

Essential principle for a profound understanding of concepts

ML Libraries

Concept mirrored in different types of libraries

1. **Numpy** forces us to do mathematical derivation
2. **JAX** adds automatic differentiation over Numpy
3. **Pytorch** allows to define networks in seconds

Numpy

JAX

PyTorch

Other helpful libraries

Librosa

Scipy

Seaborn

Bokeh

Scikit

Panel

Matplot

Manim

Session Plan

Goals of this course

1. **Introduction to model dissection**
   Unveiling the inner guts of a generative model
2. **Introduction to active divergence**
   A set of methods to go beyond model's "well- behaved" capabilities
3. **Network analysis and bending**
   A practical use case with RAVE
4. **Real-time network bending**
   Destructing your model in real time with nn~

Model dissection

What is dissection?

"Methodically cutting up a ~~body or plant~~ in order to study its internal parts"

neural network

Model dissection

What is a neural network already?

Biological neurons

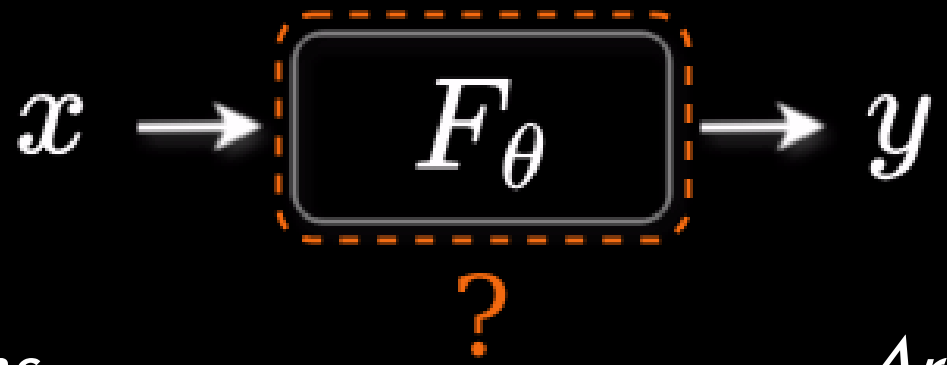
- Specific stimuli → ***strong/weak response***
- ***Trigger sources*** : colour, smell, output of other neuron …
- Learn through ***evolution*** and ***experience***

Artificial neurons

- Numerical inputs and outputs
- Strong/weak ↔ ***parameterized operations***
- Triggering ↔ ***activation functions***
- Learn through ***gradient descent***

Model dissection

What is a neural network already?



Biological neurons

- Specific stimuli → ***strong/weak response***
- ***Trigger sources*** : colour, smell, the output of an other neuron …
- Learn through ***evolution*** and ***experience***

Artificial neurons

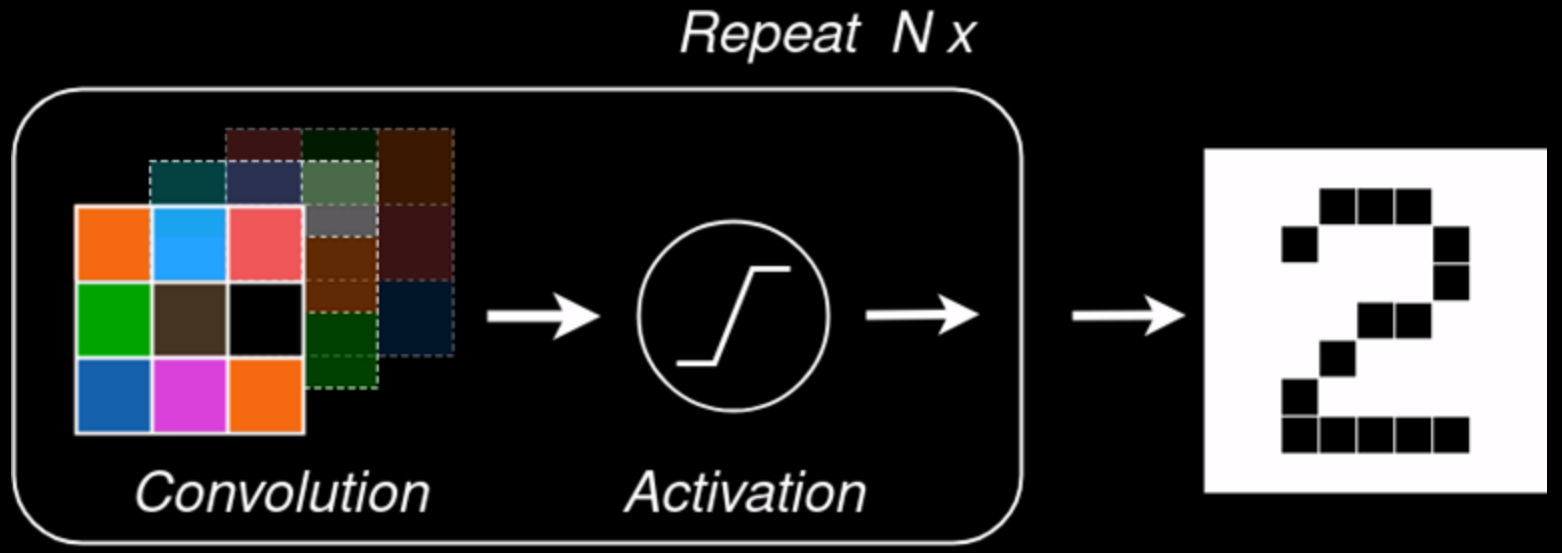
- Numerical inputs and outputs
- Strong/weak ↔ ***parameterized operations***
- Triggering ↔ ***activation functions***
- Learn through ***gradient descent***

What are these "parameterized operations" and "activation functions" ?

Model dissection

What is a neural network already?

- Starting from elementary operations (multiply, add, exponentiate ...)
- Assembling these operations creates a complex function
- Some of these operations are denoted as ***layers***
- Each layer takes the result of the previous one as input, and produces an output for the next one

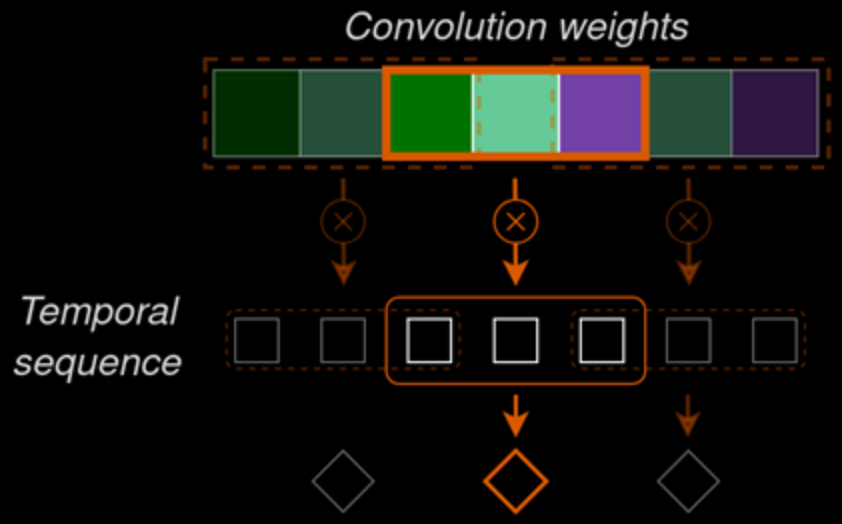


Example of the structure of a neural network composed of a succession of convolutions and activation functions

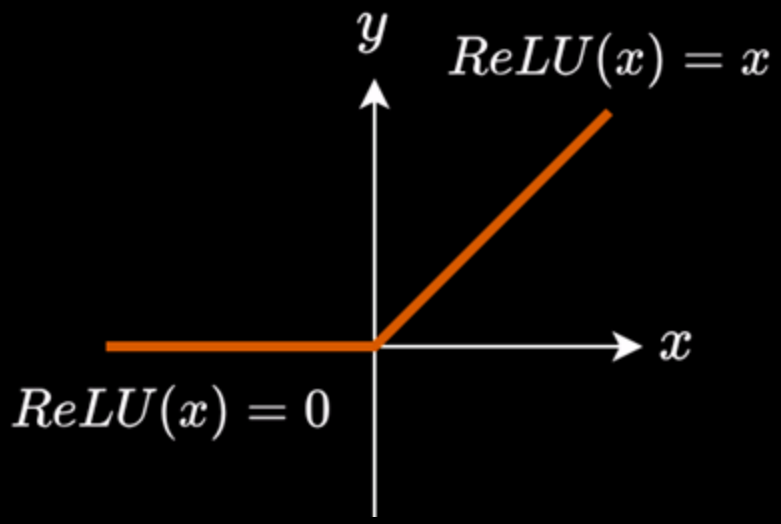
Model dissection

What is a neural network already?

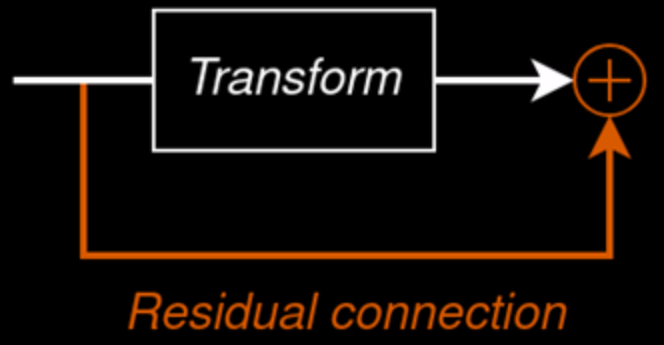
Convolution
Overlapp-add with learned ***weights*** and ***biases***



ReLU
Cut all signal below 0



Residual connection
Add untransformed signal to the output

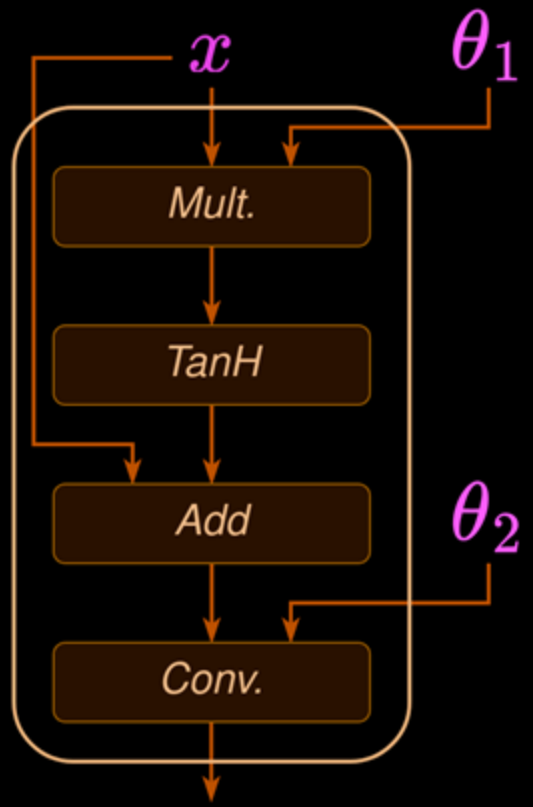


Model dissection

What is a neural network already?

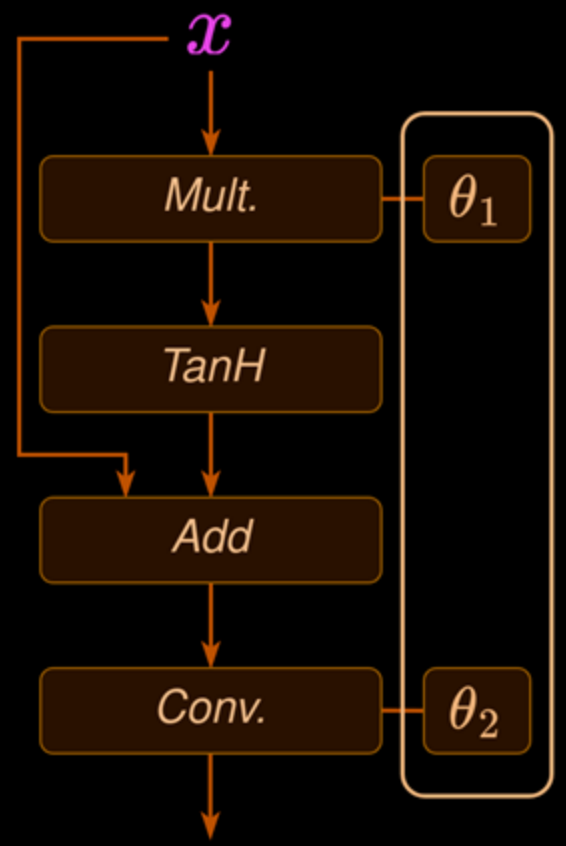
Computational graph

- Chain of operations within the network
- = list of effects within a rack (e.g. EQ → reverb → compressor ...)
- Without the parameters values !



State dict

- Set of parameters of the network
- = Preset of a synth patch (oscillator type, filter cutoff...)
- Without the actual synth engine !

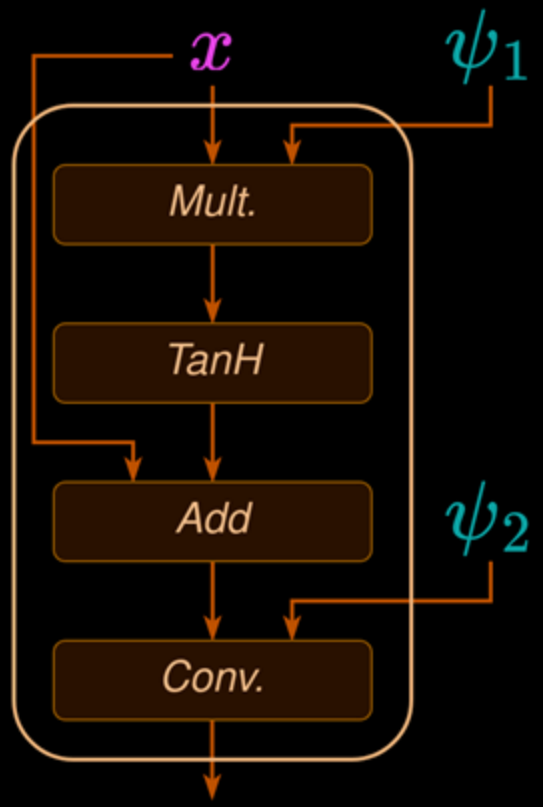


Model dissection

What is a neural network already?

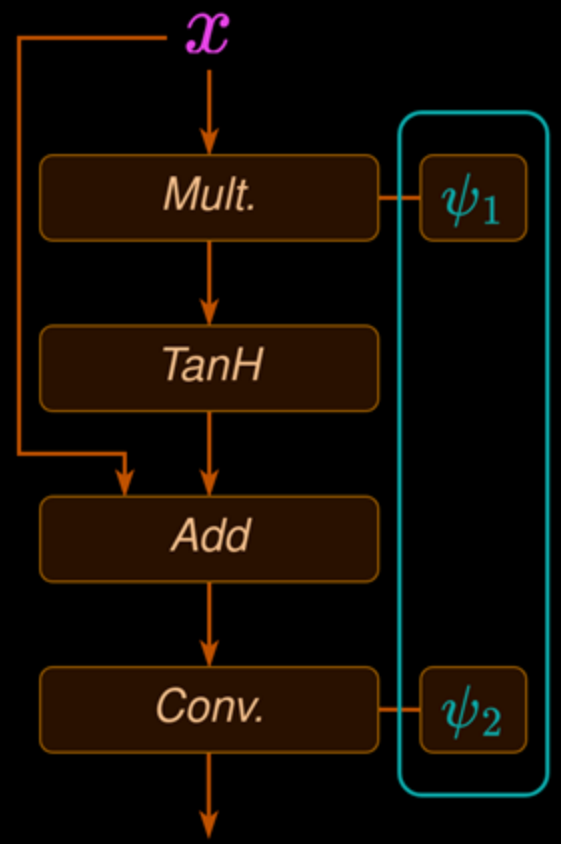
Computational graph

- Chain of operations within the network
- = list of effects within a rack (e.g. EQ → reverb → compressor ...)
- Without the parameters values !
- **Same comp. graph**
- (= same effect chain)



State dict

- Set of parameters of the network
- = Preset of a synth patch (oscillator type, filter cutoff...)
- Without the actual synth engine !
- **Different state dict**
- (= different preset for each effect)

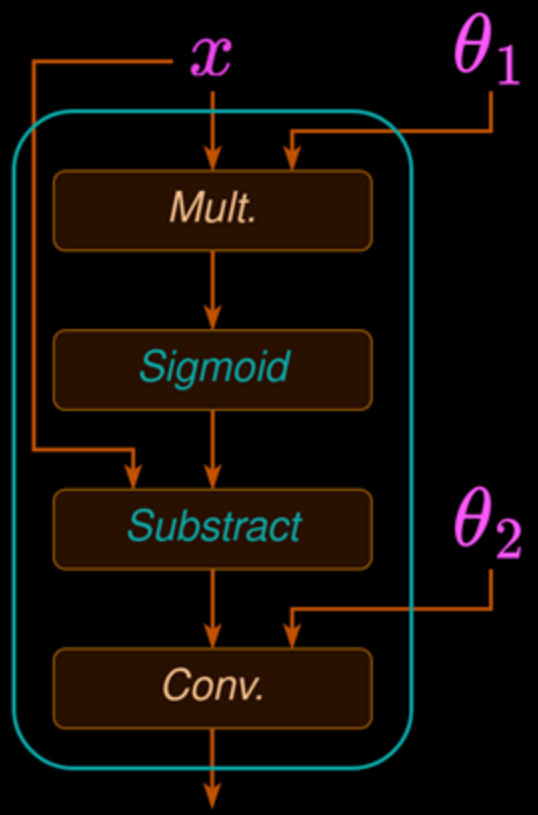


Model dissection

What is a neural network already?

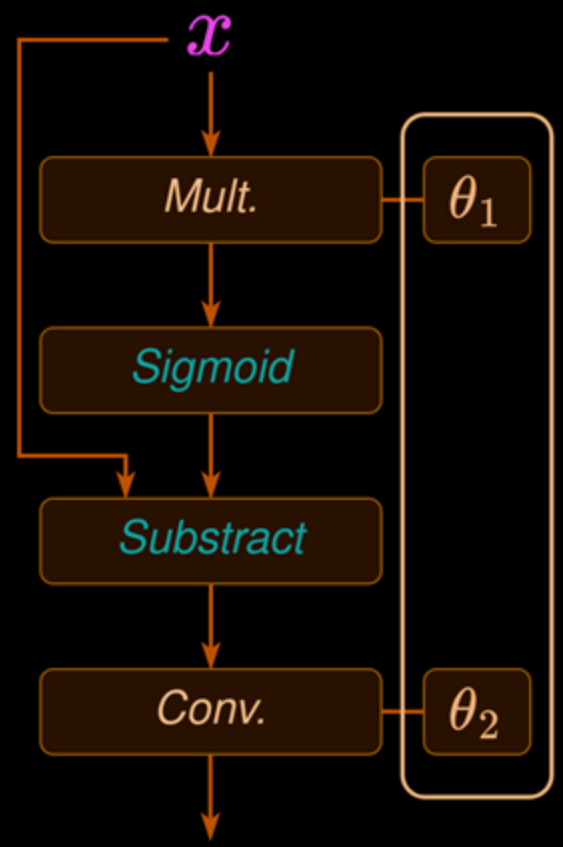
Computational graph

- Chain of operations within the network
- = list of effects within a rack (e.g. EQ → reverb → compressor ...)
- Without the parameters values !
- **Different comp. graph**
- (= changing the compressor inside the effect chain)



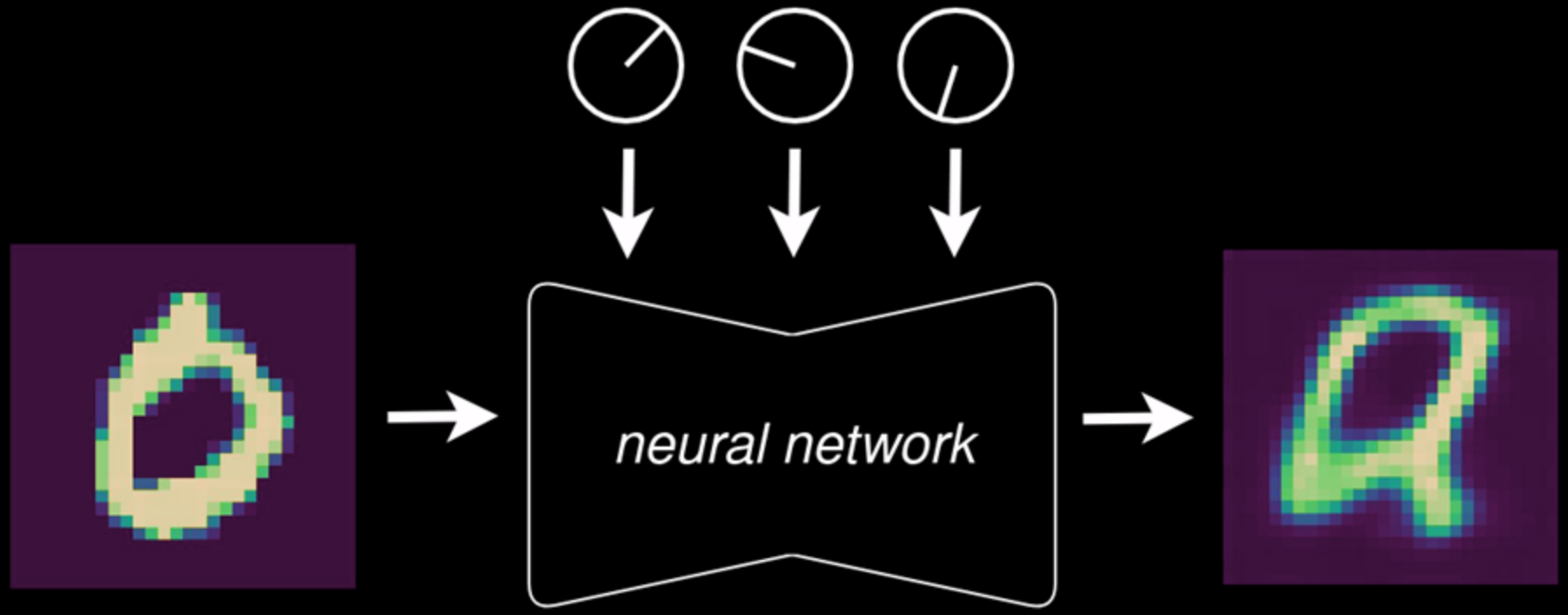
State dict

- Set of parameters of the network
- = Preset of a synth patch (oscillator type, filter cutoff...)
- Without the actual synth engine !
- **Same state dict**
- (= keeping the same settings for the new compressor)



Model dissection

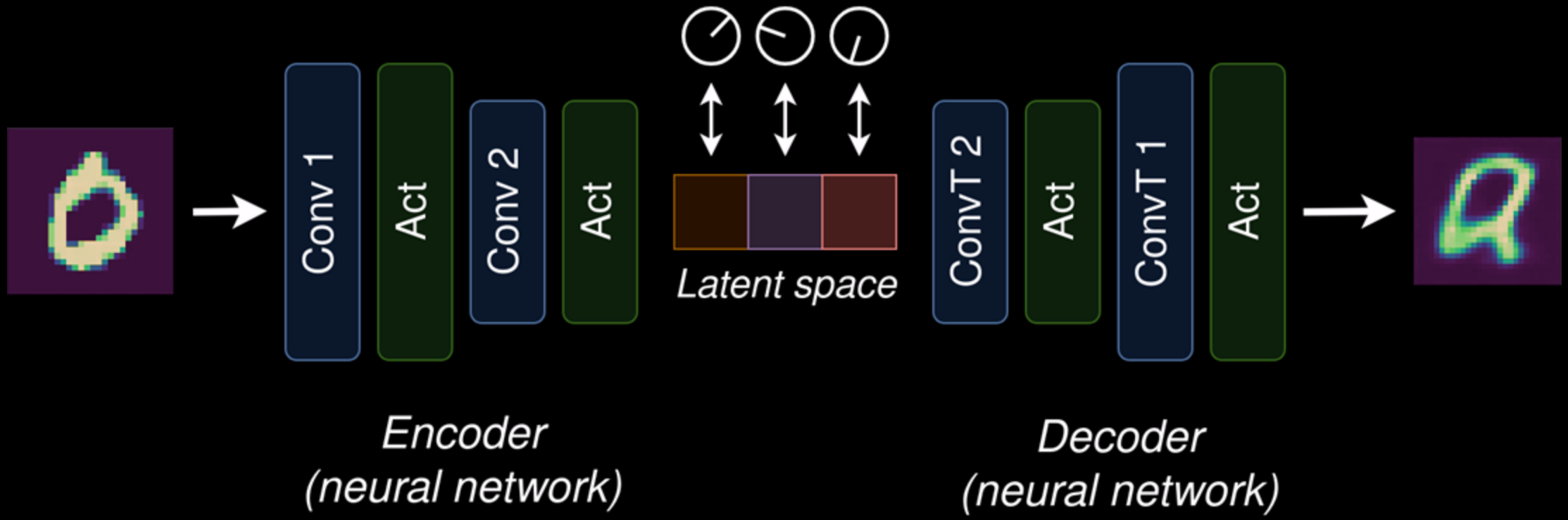
An example with a simple generator



*Example of a an **Auto-Encoder** (neural generation+attribute manipulation) trained to synthesize digits*

Model dissection

An example with a simple generator

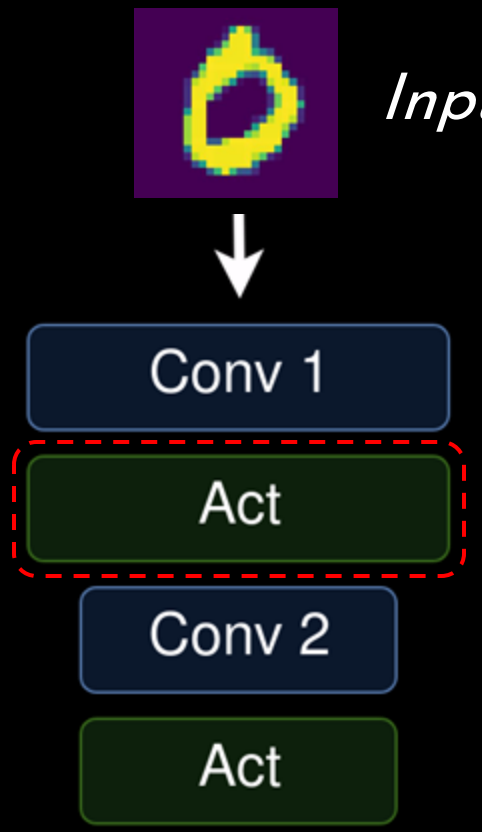


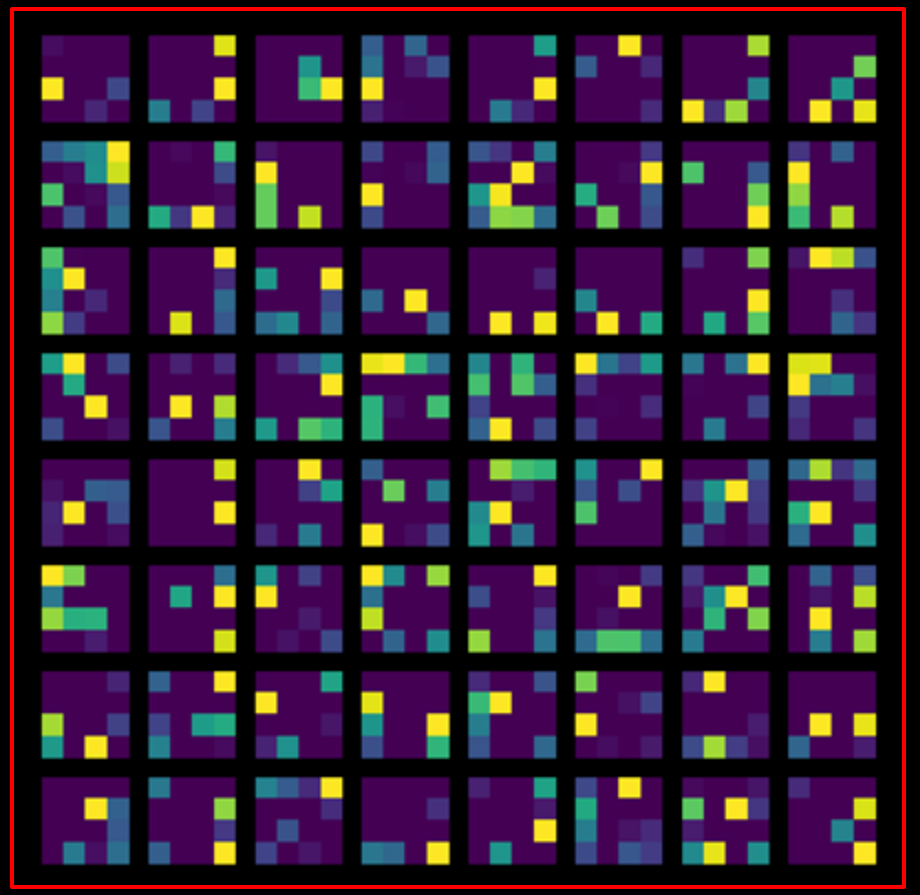
*Detailed overview of a an Auto-Encoder : **encoder** (feature extraction) and **decoder** (generation)*

Model dissection

An example with a simple generator

- Computational block : input → output
- Succession of blocks : gradually transforms **input** into **completely different output**
- Can be seen as a long and complex **audio effect chain**
- Dissection : analyzing these **intermediate outputs** / the **parameters** of these blocks

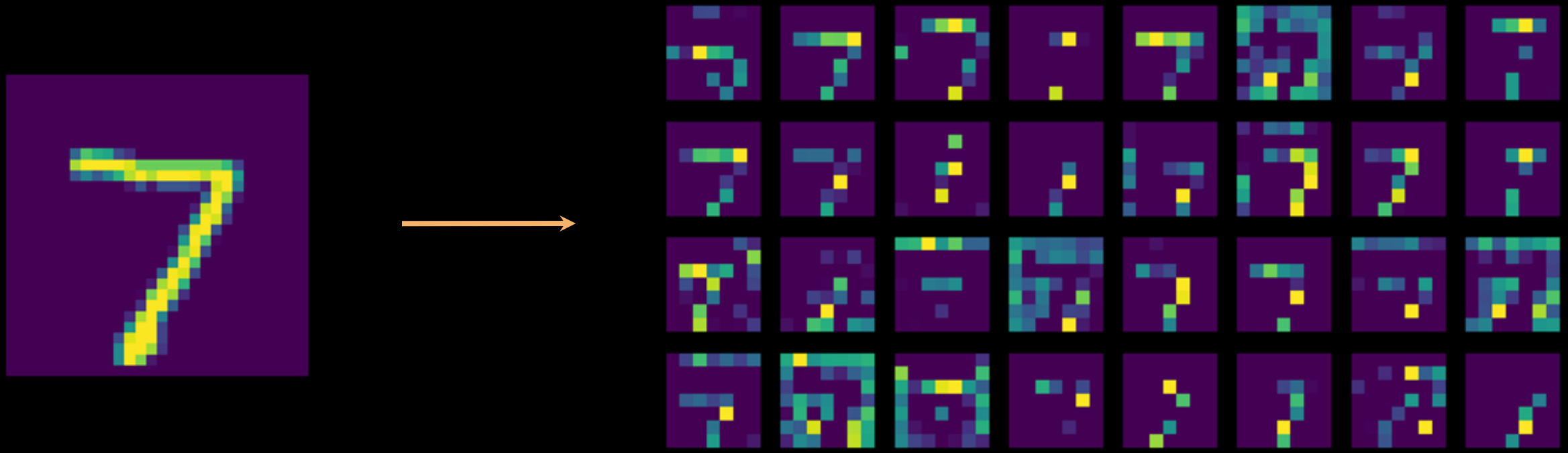




Intermediate outputs of the digits auto-encoder

Model dissection

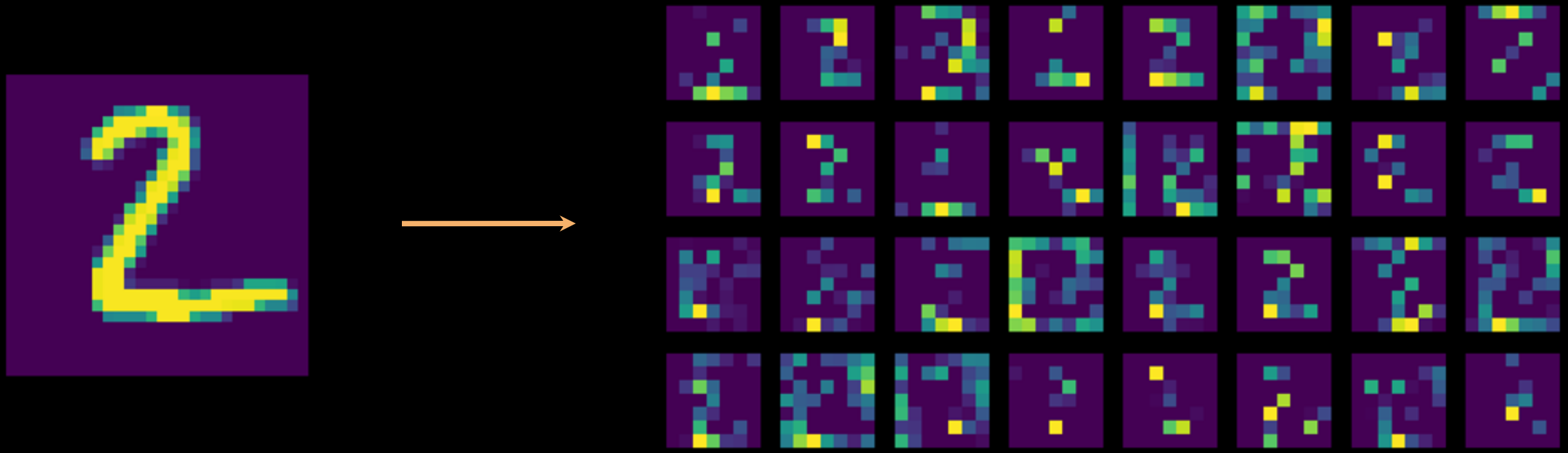
An example with a simple generator



Examples of intermediate activations for our neural digits generator

Model dissection

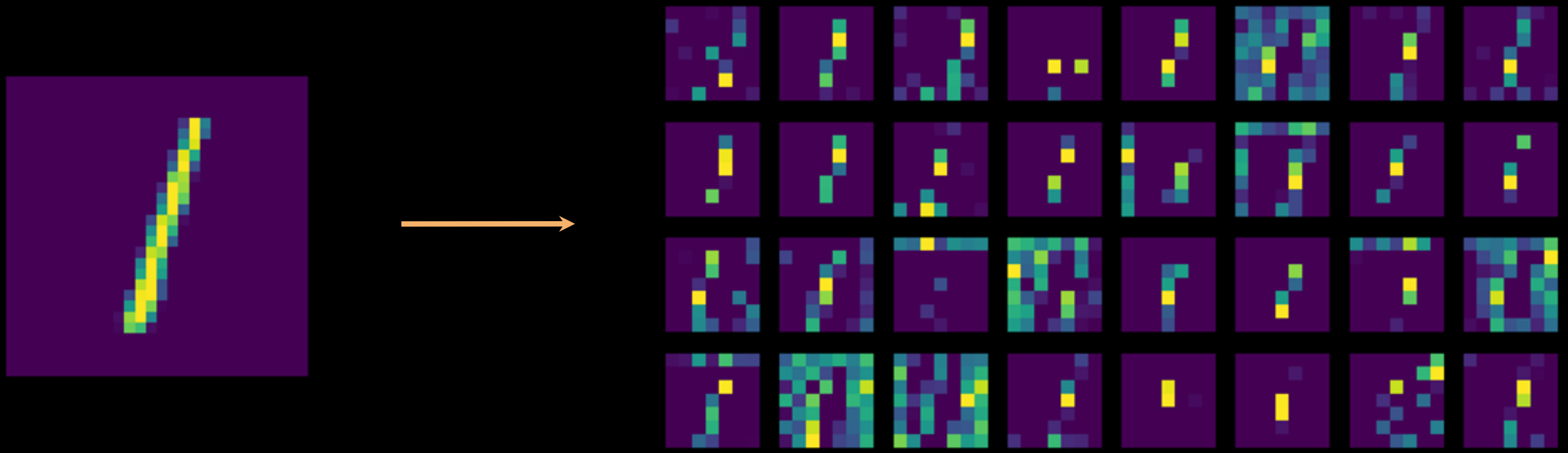
An example with a simple generator



Examples of intermediate activations for our neural digits generator

Model dissection

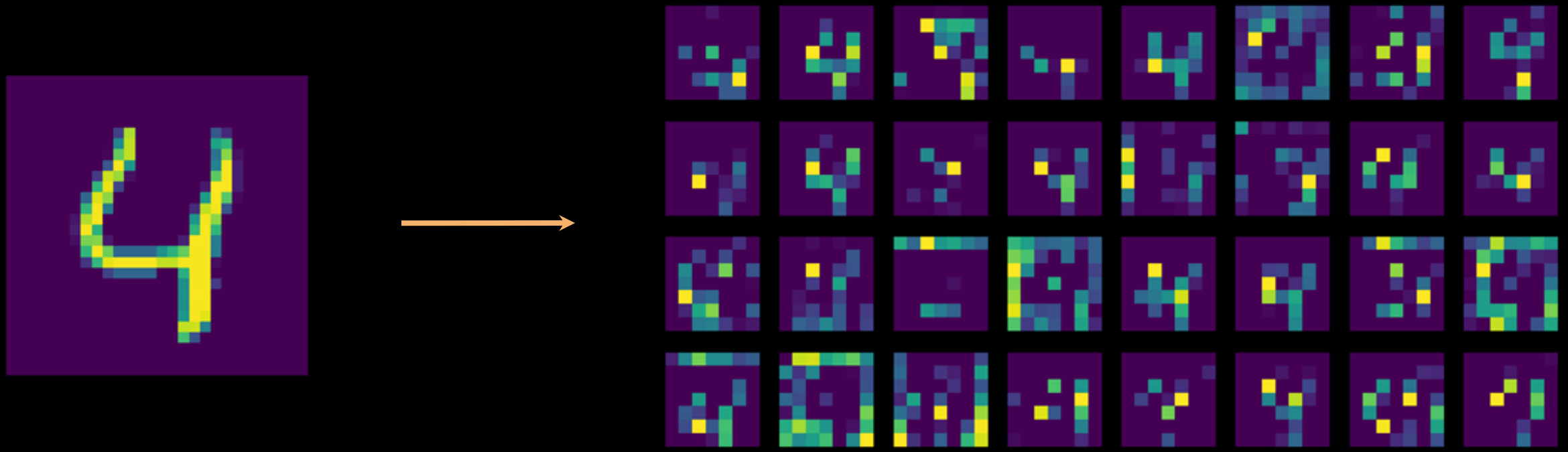
An example with a simple generator



Examples of intermediate activations for our neural digits generator

Model dissection

An example with a simple generator



Examples of intermediate activations for our neural digits generator

Model dissection

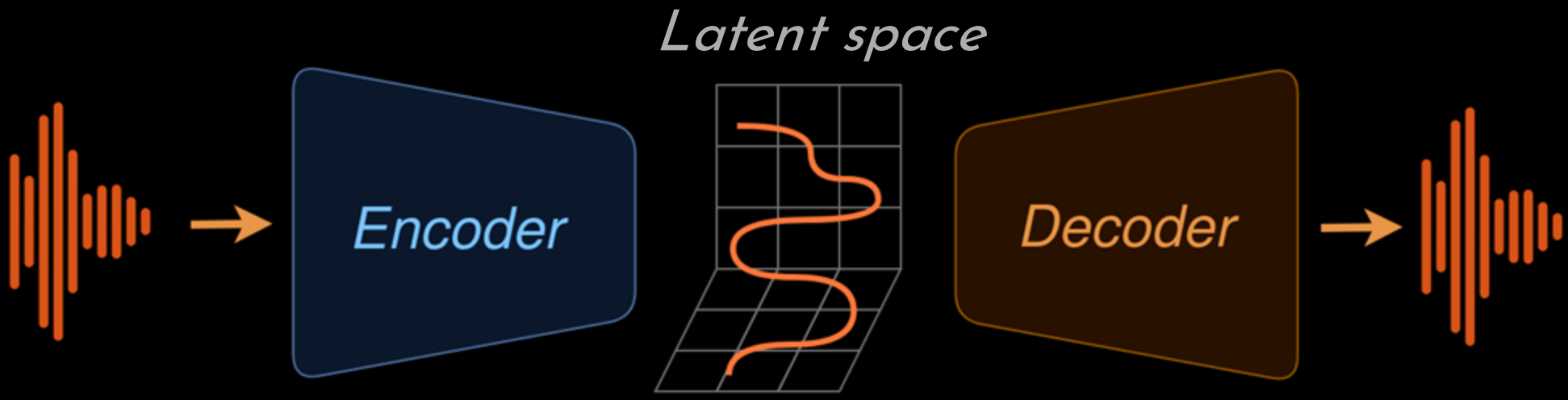
Why?

- Helps to understand what is going on inside these networks
- Disentangling the black box : understanding how it **associates concepts**, how **complex patterns** emerge from the association of **simple features**
- Can be used to **remove harmful content**
- Producing **smaller models** by removing useless parts
- Enhances the general knowledge on how neural networks work

For the curious ones, more details neural network interpretability can be found in this blog , along with sweet visualisations

Model dissection

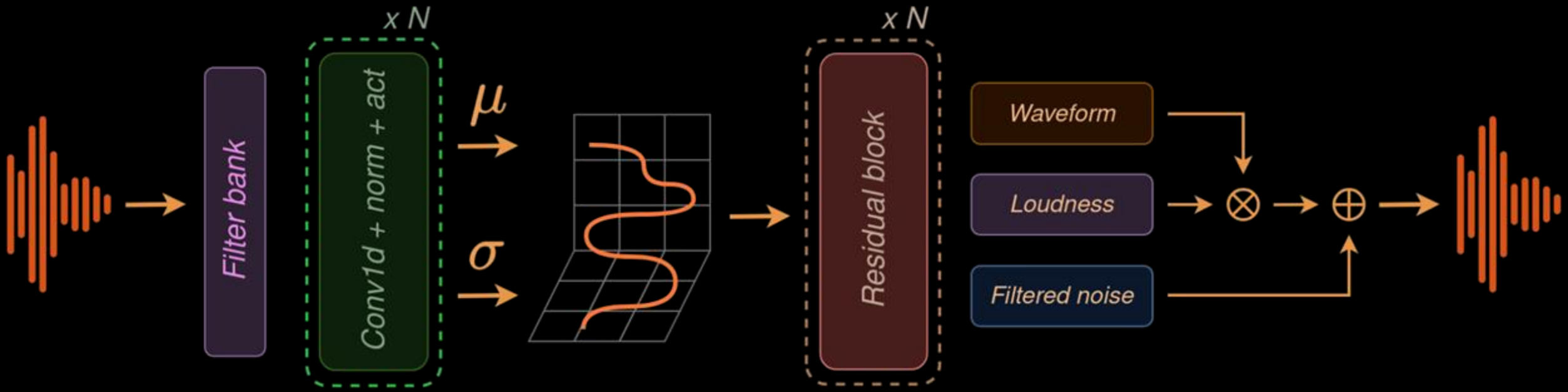
Application to RAVE (zoom 1x)



Overview of the RAVE model : the encoder produces latent trajectories/curves, which serve as inputs to the decoder to generate sound

Model dissection

Application to RAVE (zoom x1.3)

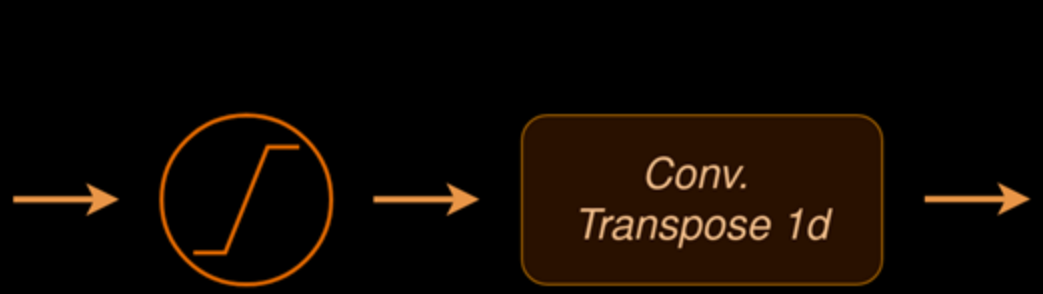


Detailed architecture of RAVE : succession of 1-d convolutions+activations.
Decoder : outputs filtered noise, waveform, and a loudness enveloppe, combined to produce audio

Model dissection

Application to RAVE (zoom x2)

Building block of RAVE decoder

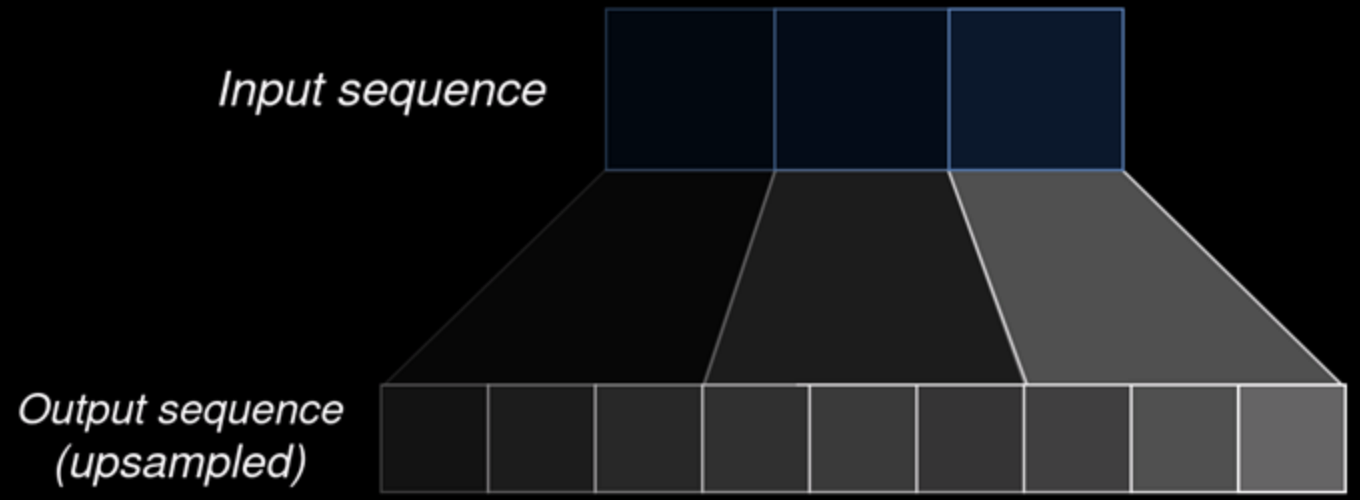


Detailed upsampling block

Detailed residual block

Model dissection

Application to RAVE (zoom x4)



Representation of an upsampling block, which increases the temporality of an input sequence

Model dissection

Application to RAVE (zoom x4)

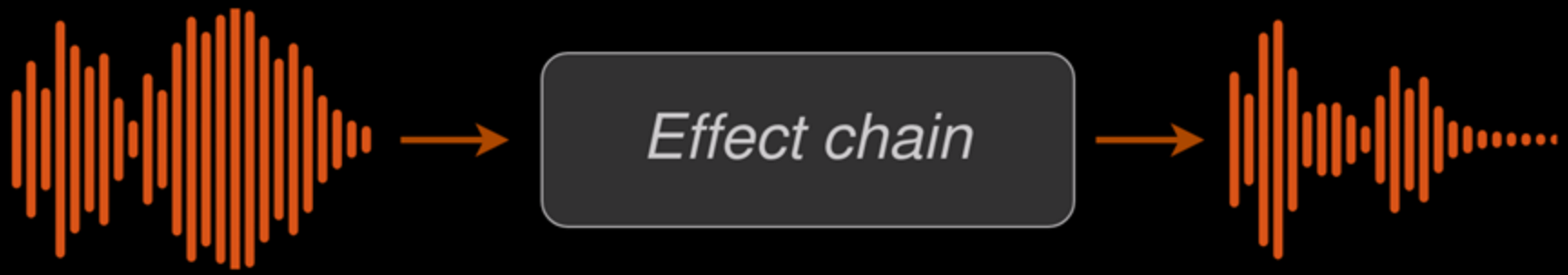
- Might seem complicated at first sight…
- …But fundamentally close to a “traditional” synthesizer !
- Generator : equivalent to a powerful **additive+subtractive synthesizer**
- Latent space : **Controllable*** **parameters** of this synth
- Encoder : **automates** these knobs to match a particular input
- Acting on these automations to **alter sonic properties** of the input (pitch, loudness, darkness…)

** we will see later how to access hidden controls…*

Model dissection

Why?

- Helps to understand **what is going on** inside the network
- Imagine having a long and complex effect chain
- Understanding each effect **individually** : allows to understand the full rack

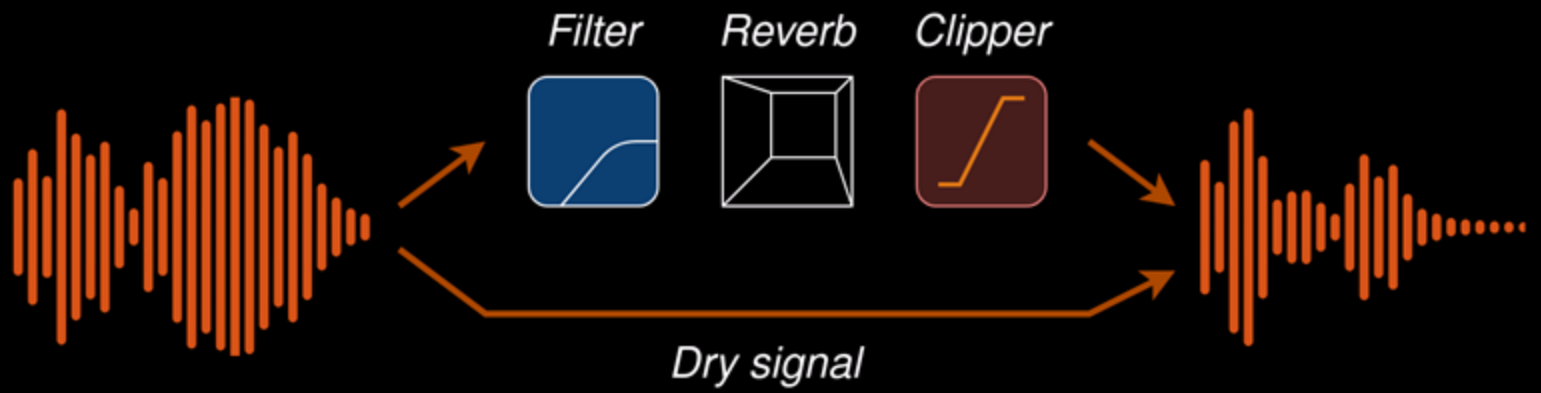


Hidden effect chain : unknown transformations between input and output

Model dissection

Why?

- Helps to understand **what is going on** inside the network
- Imagine having a long and complex effect chain
- Understanding each effect **individually** : allows to understand the full rack



Revealing effect chain makes everything clearer !

Model dissection

Why?

- Making **smaller models** by removing useless parts
  - Realtime generation
  - Embedded synthesis

- **Altering the output** of the model and enhancing its generative potential
  - Like having thousands of exciting knobs !
  - Cf. **bending** (a bit later...)

Model dissection

Now, how to dissect a model (concretely)?

- It is complicated...
- We would like to be able to access the outputs of each operation...
- ...But neural networks tend to be rather hermetic and tightly sealed
- So how can we still dissect them ?

Model dissection

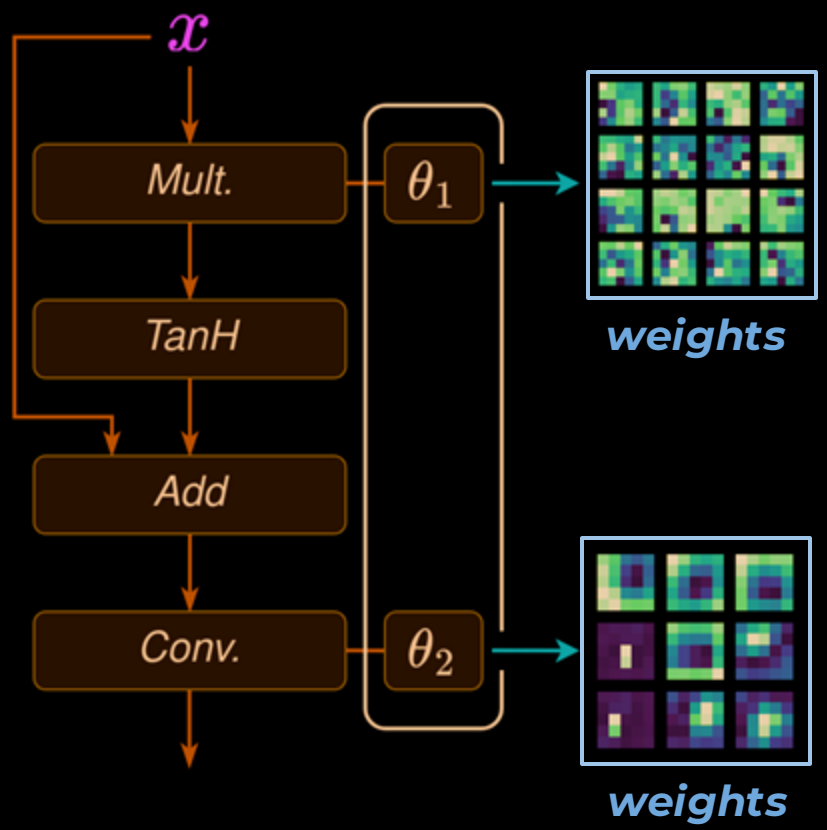
torchbend

torchbend is a *very experimental* framework made for :

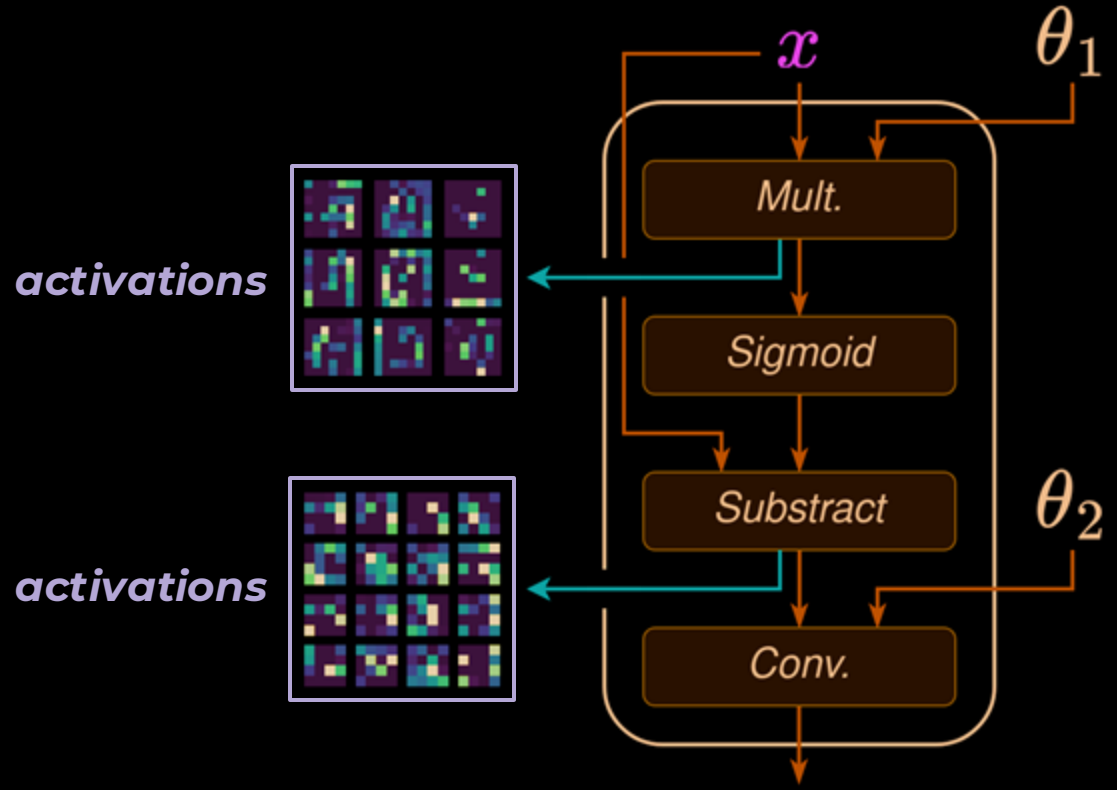
- model dissection (this morning)
- messing everything up (this afternoon)

Model dissection

How to dissect a model?



*Dissecting the **state dict** (parameters)*



*Dissecting the **graph** (layers outputs)*

Model dissection

How to dissect a model?

Building the network (basic blocks)

Forward function (signal flow between these blocks)

Creating the model

Getting the state dict

```python
class MyModel(nn.Module):
    def __init__(self):
        super().__init__()

        # Building a toy network composed of a convolution and an activation
        self.convolution = nn.Conv1d(in_channels=1, out_channels=1, kernel_size=4)
        self.activation_function = nn.Tanh()

    def forward(self, input):

        # Applying a convolution to the input
        convolved_input = self.convolution(input)

        # Applying an activation function to the output of the convolution
        activation = self.activation_function(convolved_input)

        return activation

neural_network = MyModel()

print(neural_network.state_dict())
```

A toy example : dissecting the state dict of a simple neural network

Model dissection

How to dissect a model?

Output of this code

```
{'convolution.weight': tensor([[[ 0.4883,  0.1896, -0.0974,  0.4747]]]), 'convolution.bias': tensor([0.0404])}
```

Name of the parameter

Value of the parameter

A toy example : dissecting the state dict of a simple neural network

Model dissection

How to dissect a model?

Make network dissectable with torchbend

Record intermediate outputs for a given input

```
# Dissecting the activations of a network using torchbend
import torchbend as tb

# Wrap the neural network inside a BendedModule
bendable_network = tb.BendedModule(neural_network)
input = torch.zeros(4, 1, 16)

# Trace the network using a fake input
bendable_network.trace(x=input)
bendable_network.print_activations()
```

Extracting the intermediate outputs of our simple network

Model dissection

How to dissect a model?

Output of this code

```
------------------  -----------  ------------------  ----------------------
x                   placeholder  x                   torch.Size([4, 1, 16])
convolution         call_module  convolution         torch.Size([4, 1, 13])
activation_function call_module  activation_function torch.Size([4, 1, 13])
------------------  -----------  ------------------  ----------------------
```

Operation name

Output dimensionality

Extracting the intermediate outputs of our simple network

Model dissection

Let's go