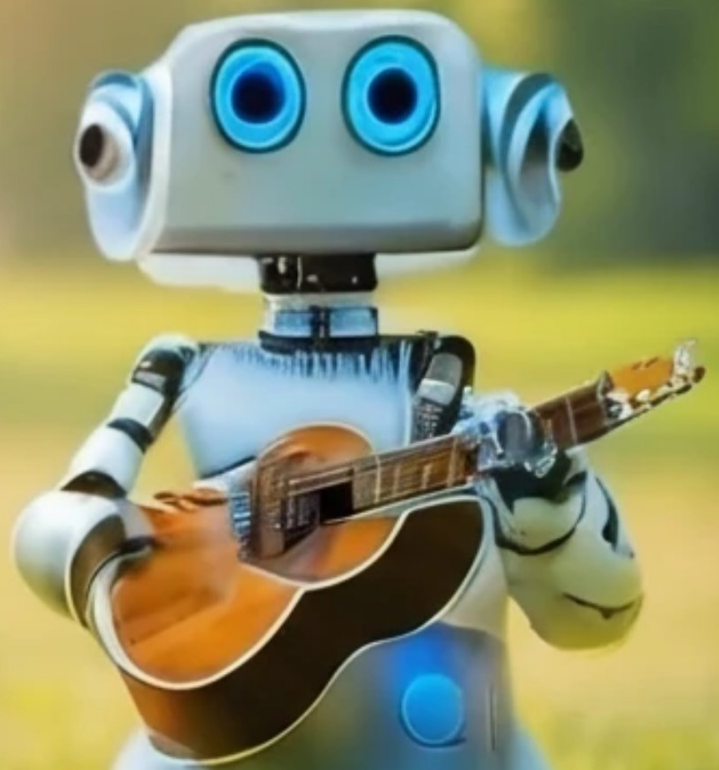
Machine learning tools for music production

Jérémy Cohen
CNRS, CREATIS, Lyon, France
Master Musicologie CCNT

Day 1 · Music + AI fundamentals and pitch detection



Today's schedule

10:00 · 10:30	10:30 · 11:30	11:30 · 12:00	12:00 · 13:00
Overview Generative AI, tools, AI Song Contest, debate	**AI fundamentals** Data, models, training, neural networks	**Sound as data** Sampling, Fourier, spectrograms	**Pitch detection** The problem, YIN, CREPE in detail

Afternoon (3 h): lab. Spectral peak → YIN → pre-trained CREPE → trained mini-CREPE → evaluation → synthesis

15 min break around 11:30 · times are indicative

Round table: you and AI

?

Do you use AI tools? Which ones, and what for?

?!

What worries you? Copyright, jobs, energy…



And for music: composing, mixing, stem separation, vocals…?

!

What excites you? What would you like to be able to do?

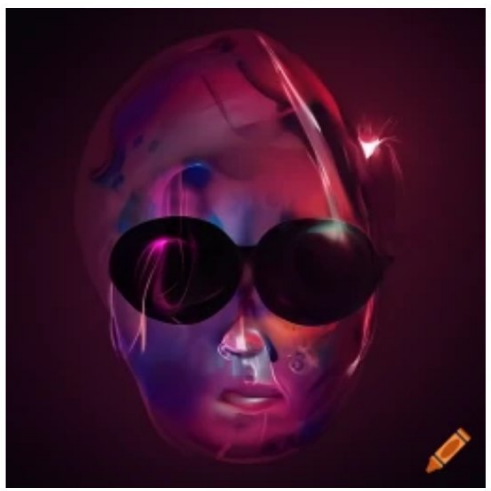
Generative “AI”

Some general opinion: "AI is an advanced technology for generating text and images"

- ChatGPT, Claude, Le Chat…
- Some open source options (Stable Diffusion) and lightweight models, e.g. Crayion

Examples (2024):
music and AI







What about music?

- 2024: “MusicGen” examples from Meta AI (Facebook)

Generative AI is evolving very fast

2025+: new players! Example: Suno



Generates realistic music from:

- a text description
- a recording (voice, instruments)
- song lyrics

Works very well for vocals!

Generative AI: for or against?

For

+ Making music without formal musical training: a new kind of digital artist?

+ Generating new ideas, playing with AI tools

+ ?

Against

− Intellectual property issues (many companies such as Suno are being sued, especially in Europe)

− Musicians replaced by AI?

− Problematic positions against artists' work

Note: I am not an ethics expert.

1. There are many AI-based audio tools

2. (Some) artists have already started to adopt these tools

A contest on AI in music

AI SONG CONTEST

Since 2020, launched in the Netherlands!

A contest on AI in music



2022 winner

Artist: **Yaboi Hanoi**

Title: อสุระเทวะชุมนุม – *Enter Demons & Gods* (“Asura Deva Choom Noom”)

Audio and explanations: link



2024 winner(s)

Artist: **Onda Corta**

Title: *Sudamérica*

Audio and explanations: link

Song Contest: how is AI used?

Generative AI

- Lyrics
- Text → singing [ElevenLabs]
- Drum patterns [demo]
- Ideas for composing (arrangement)

Other “AI” tools

- **Note recognition**
- **Timbre transfer**
- Stem separation [Moises]
- Mixing / mastering
- Text → textures
- Singing voice enhancement
- Synth preset generation

Some tools used (some outdated!):

ChatGPT · Claude · Gemini · AIDrummachine · ElevenLabs · Moises · Mofi · ToneTransfer · BasicPitch · Magenta VST · MuseNet · AIVA · This patch does not exist · …

Roadmap: inside an “AI” tool

1

Machine learning fundamentals

data · models · training · neural networks

2

Sound as data

sampling · harmonics · spectrograms

3

Pitch detection: from YIN to CREPE

the problem · classical methods · one network in detail

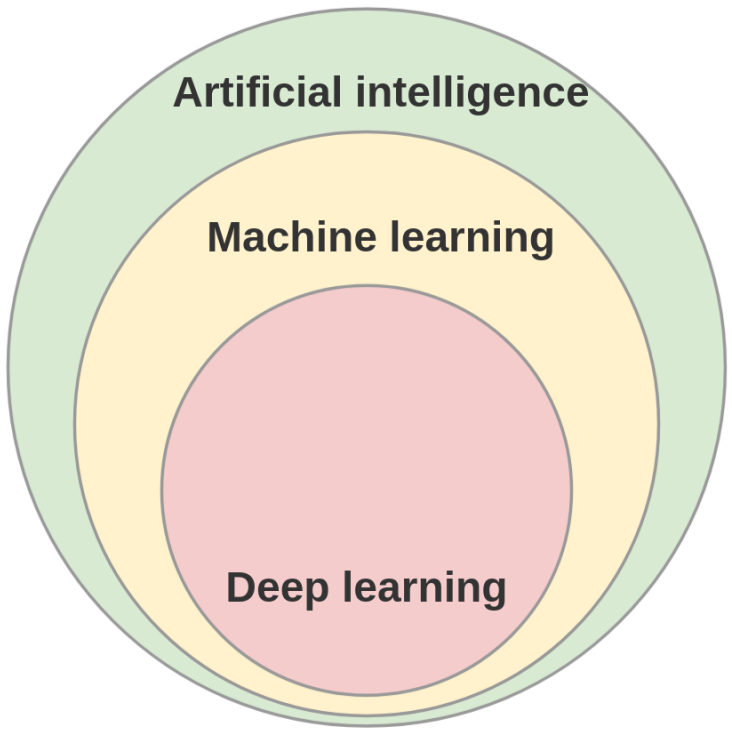
Common misconceptions to discuss

“AI understands music”	A statistical model fitted to data. It imitates regularities, with no intent.
“It costs nothing”	Training, then serving millions of requests, uses energy, water and hardware. Small model ≠ large model.
“It creates from nothing”	Everything comes from the training data → copyright (major labels sued Suno and Udio).
“It’s a magic black box”	The principles can be understood: that is what this course is about!

Machine learning

“AI” is an ambiguous term. Scientists rather speak of machine learning.

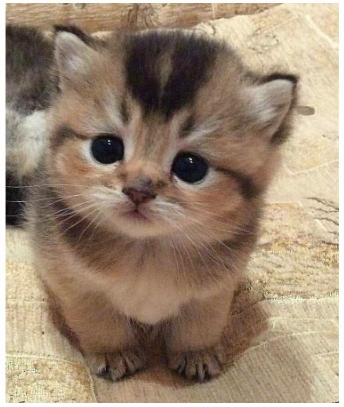
Machine learning: analyzing data with a computer.



- **1956** The term “AI” (Dartmouth)
- **1958** Perceptron (Rosenblatt)
- **1980-90** Expert systems · backpropagation
- **2012** Deep learning: AlexNet, GPUs
- **2017** Transformers (“Attention is all you need”)
- **2018** **CREPE: a CNN for pitch**
- **2022+** Mainstream generative AI (ChatGPT, Suno…)

Kinds of learning

Supervised learning



Cat

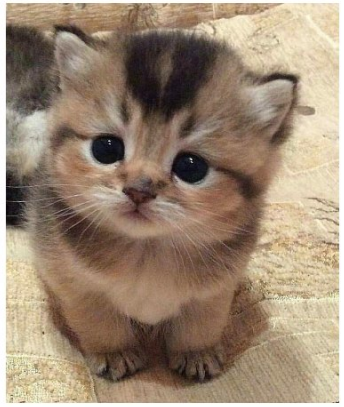


Dog

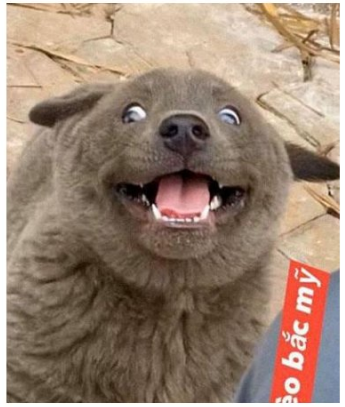


??

Unsupervised learning



??



??



??

More kinds of learning

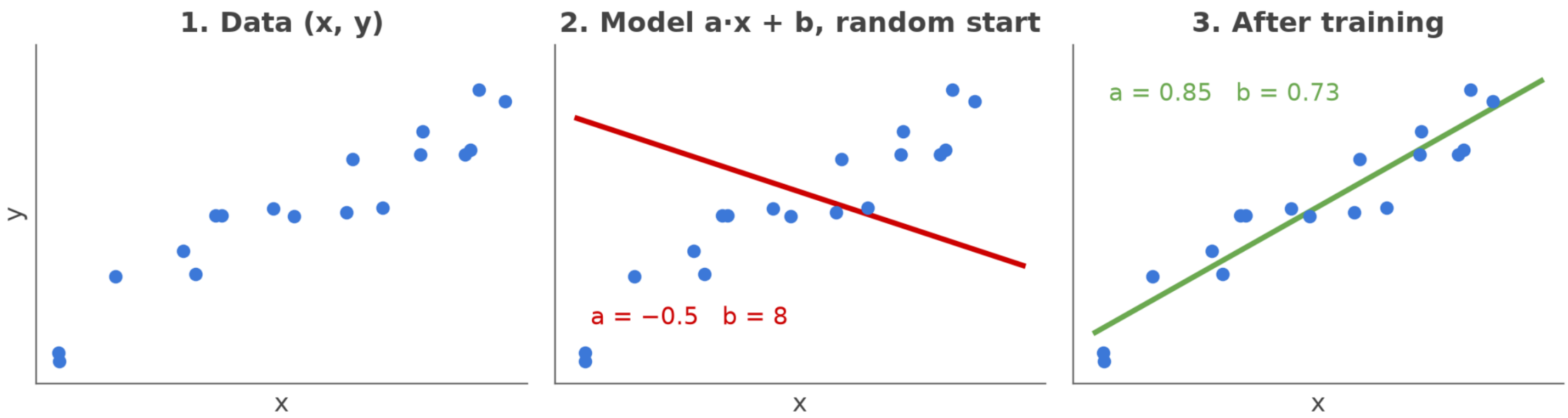
Supervised	Reinforcement	Self-supervised	Unsupervised
Labeled examples (x, y)	Act, receive a reward	The labels come from the data itself	Data x only: find structure
Music example audio → annotated pitch (CREPE, this morning)	**Music example** an agent improvises and gets a score from a listener	**Music example** predicting how a piece continues (MusicGen, LLMs)	**Music example** grouping similar sounds (clustering)

and more...

The 3 pillars of (supervised) learning

1. Training data | **2. Model** | **3. Training**

Example: **linear regression**. Learn a function f(x) = y from points, with model(x; a, b) = a·x + b



Two classical ML problems: regression and classification

Regression: continuous output

- y is a real number
- e.g. frequency f0 = 261.6 Hz
- e.g. tempo = 122.4 BPM
- typical cost: squared error

Classification: categorical output

- y belongs to a finite set
- e.g. instrument $\in$ {piano, flute, …}
- e.g. note $\in$ {C4, C#4, …}
- typical cost: cross-entropy

Question: is estimating the pitch of a note one or the other?

Machine learning: classification

1 · The task | 2 · Training | 3 · Using the model

We want a function **f** that takes an image **x** and returns its label **y**



y = **cat**



y = **dog**

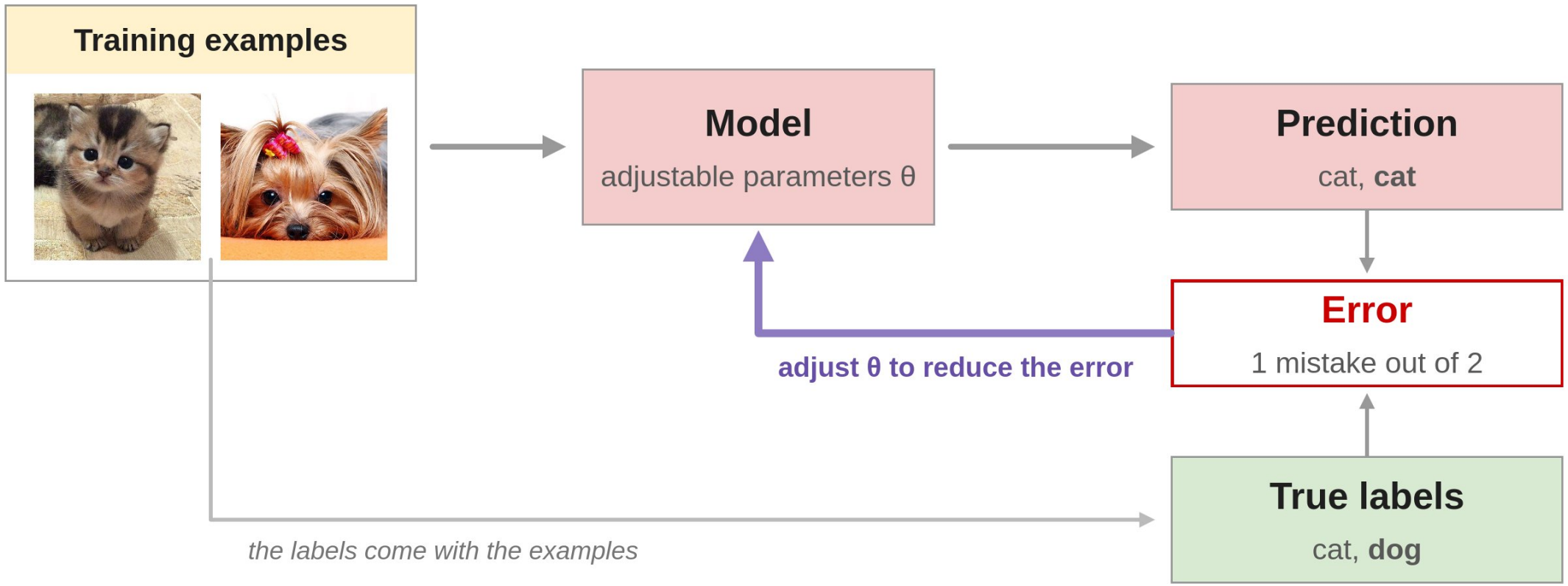


y = **dog**

f is **unknown**: nobody can write it down. We only have **examples (x, y)**, and we will learn f from them.

Machine learning: classification

1 · The task | **2 · Training** | 3 · Using the model



Repeat over thousands of examples: the predictions get closer and closer to the true labels

Machine learning: classification

1 · The task | 2 · Training | **3 · Using the model**

New image, never seen



→

Trained model

parameters θ fixed

→

Prediction

cat

✗ Wrong: it is a dog!

The real goal: work well on **new data** (generalization). It is never guaranteed → we need data the model has never seen to check it.

Pillar 1: data

After collecting data (which is a difficult problem in itself), labeled examples are split into **three separate sets**

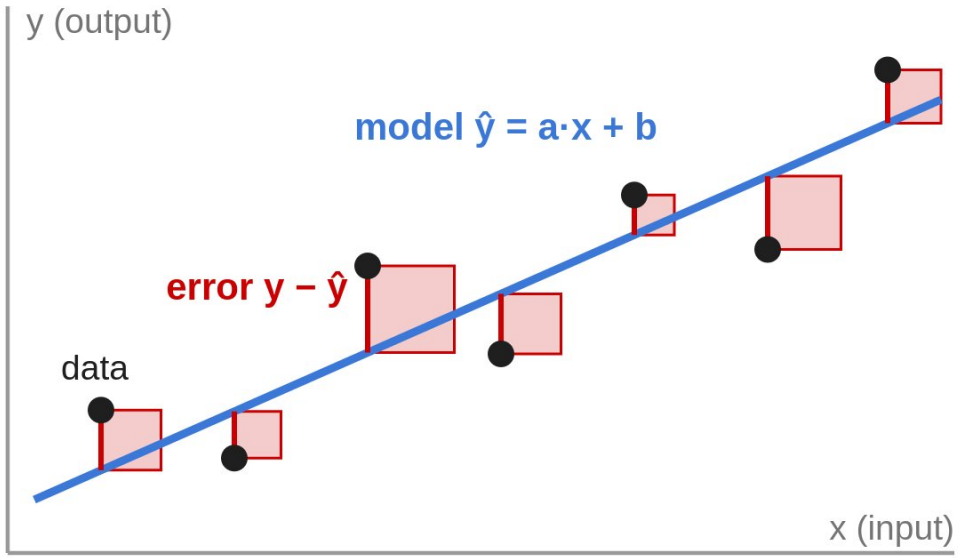
Training set · ~60%	Validation set · ~20%	Test set · ~20%
The model **learns** from these (we tune the parameters here)	We **check progress** on these while building the model (helps choosing the model if several choices are available)	Kept **hidden until the end**, used once for the final, honest score
Like reheasing for a concert	*Like playing for your teacher, he tells you what to work on*	*Like the real concert: did you learn the program well enough?*

Why? A model can simply **memorize** its examples. Only data it has **never seen** tells us if it really learned.

- Label quality limits model quality (“garbage in, garbage out”)
- Data encodes **biases**: gender, race, nationality, styles, instruments, tunings, languages…
- Labeling is expensive → synthetic data, self-supervision, augmentation

Pillar 3 - Training: the loss function

Regression: squared error



Loss = average of $(y - \hat{y})^2$, the **area of the red squares**. Big errors weigh much more.

Classification: cross-entropy



Input image · true label: **cat**

The model outputs a **probability for each class**. Two possible models:

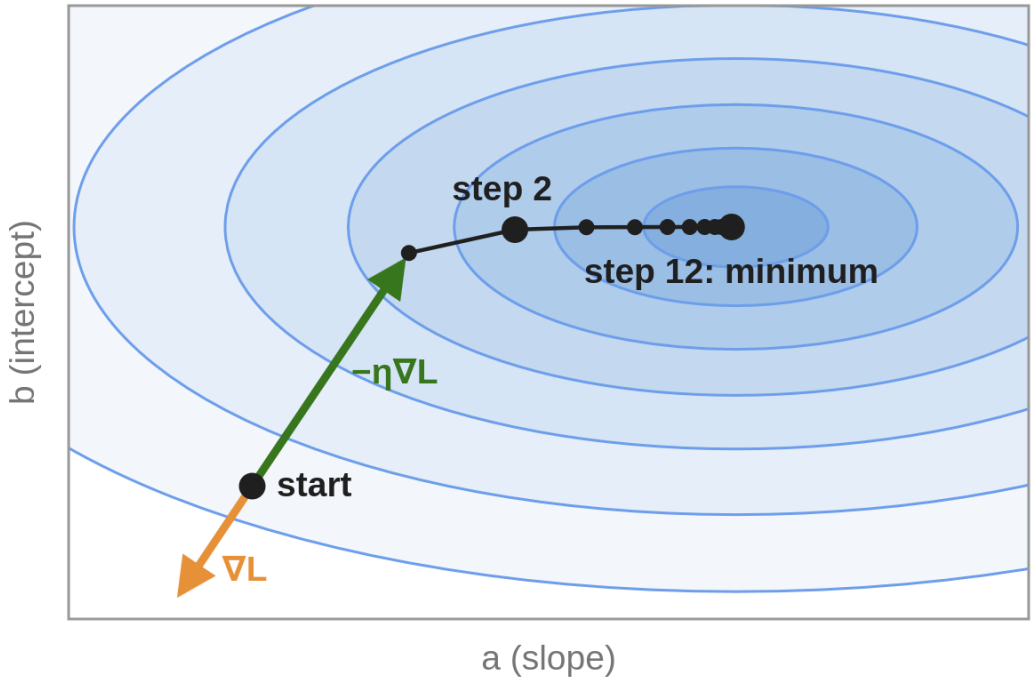
class		Model A	Model B
	**cat**	0.90	0.05
	**dog**	0.10	0.95
loss −log p(cat)		**0.1 ✓ small**	**3.0 ✗ large**

The less probability on the right answer, the higher the loss: **confident mistakes cost a lot**.

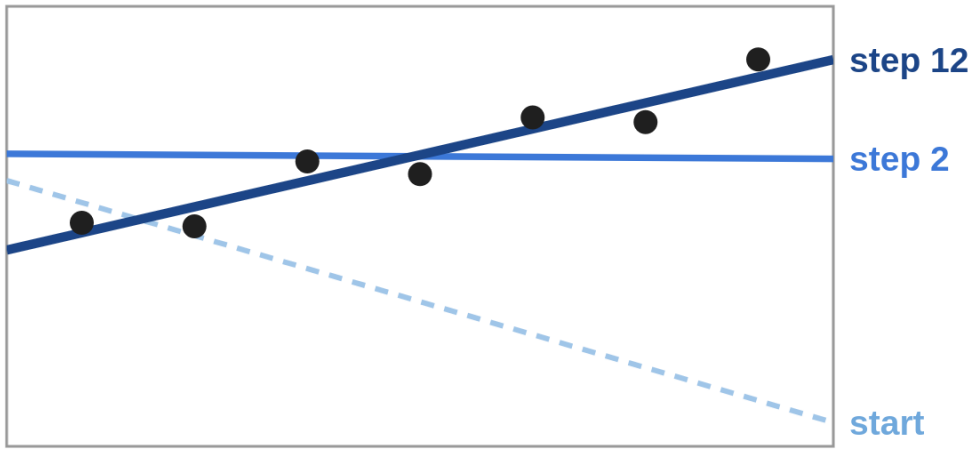
Training = finding the parameters θ that **minimize the loss**, averaged over the training data

Training: gradient descent

The loss landscape: L(a, b)



The line ŷ = a·x + b on the data



Each point (a, b) of the map is **one line**; each ring = lines with the **same loss**.

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$

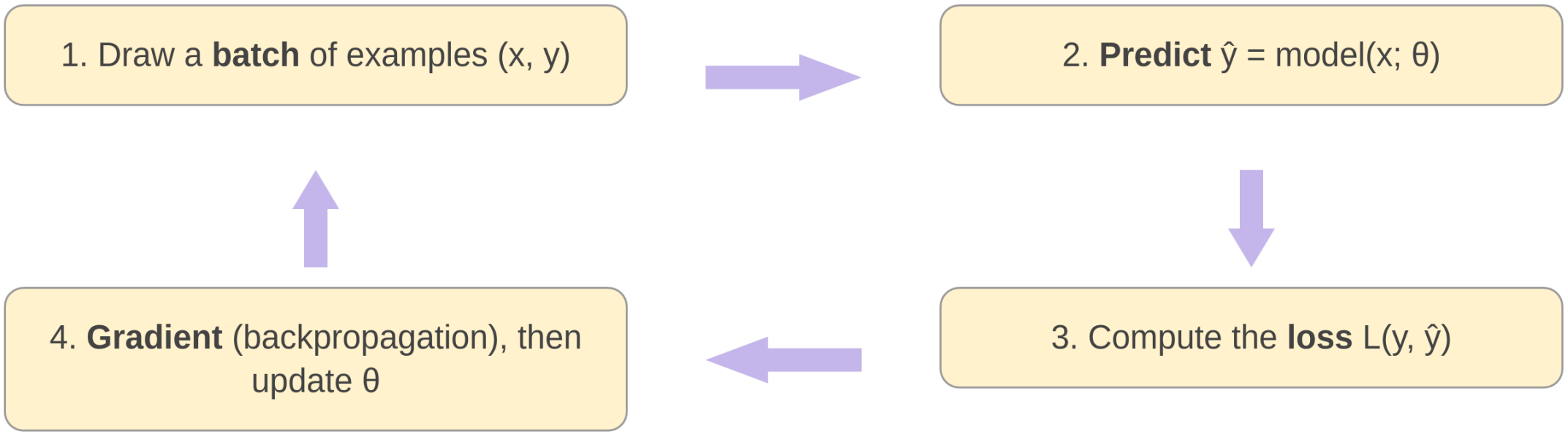
θ = (a, b): where we stand

∇L: arrow pointing uphill

−η∇L: small step downhill

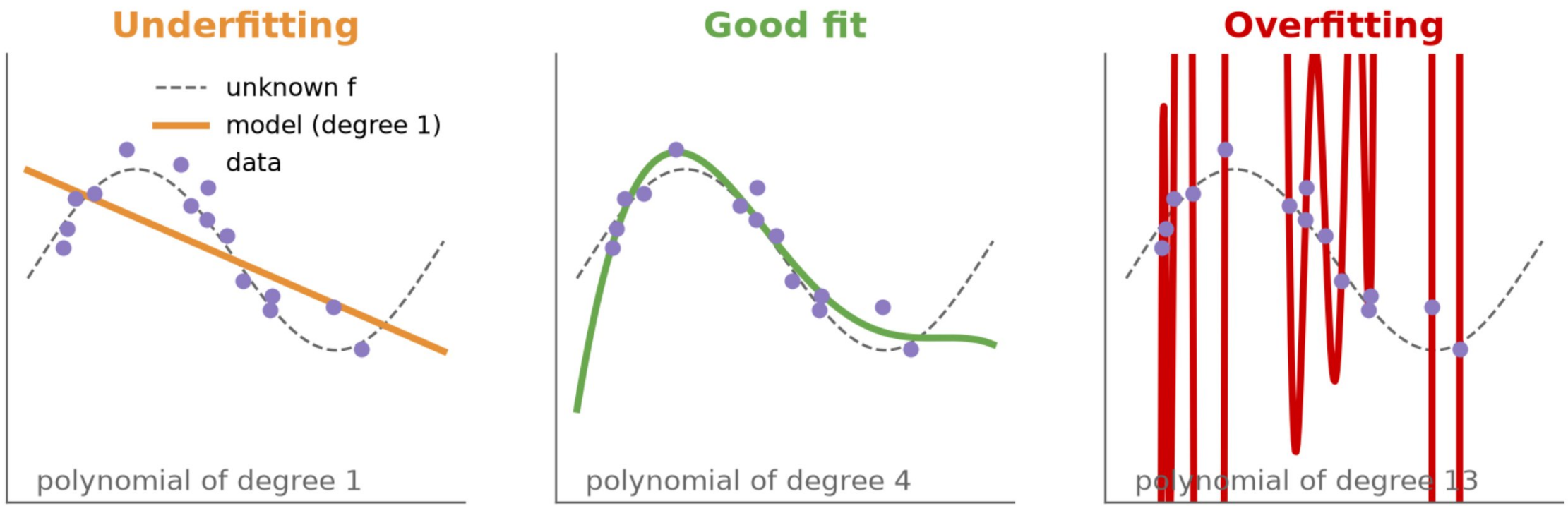
Like walking down a valley "in the fog": feel the slope, step **downhill**, repeat. η = size of the step.

Training in practice



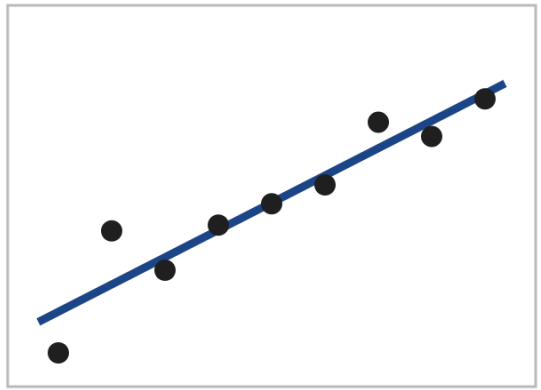
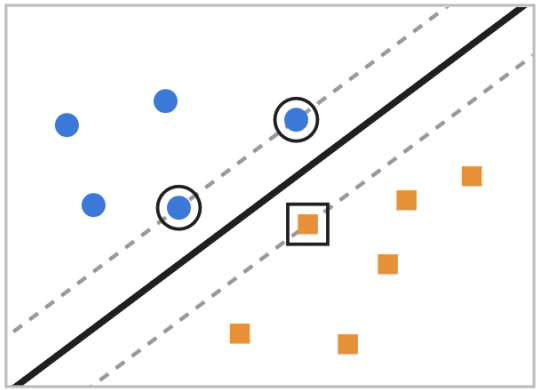
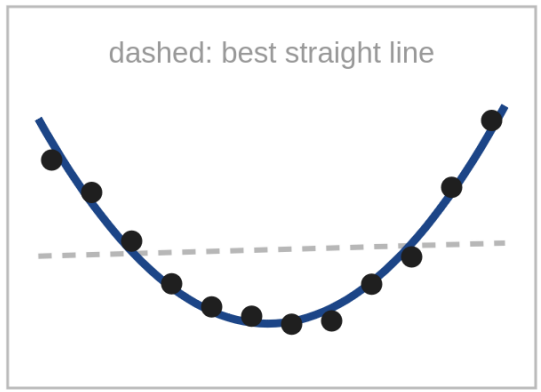
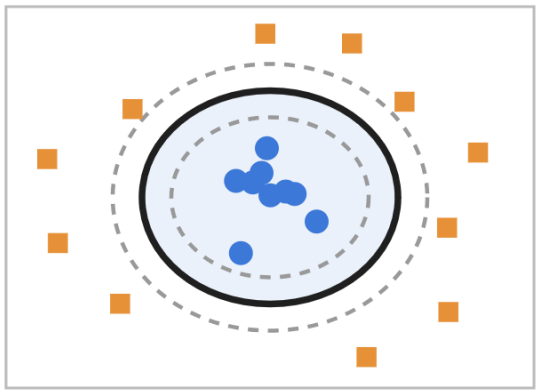
- 1 **epoch** = one pass over all the data (or a fixed number of batches)
- Optimizers: SGD, **Adam** · computed on **GPUs** · tools: PyTorch, TensorFlow / Keras

Pillar 2 - Model: Generalize, don't memorize



- A model that is too rich **memorizes** the noise in the training data
- What matters: the error on **unseen** data (validation, test)

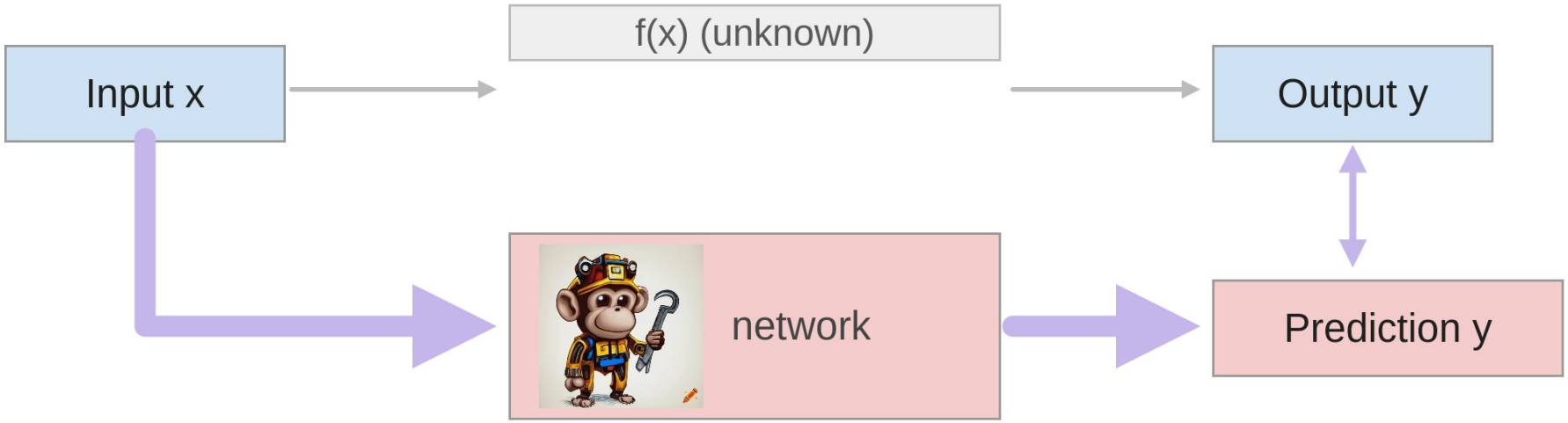
Pillar 2 – Model: simple choices

	Regression	Classification
Linear model straight line, flat boundary		
	Linear regression $\hat{y} = a \cdot x + b$ 2 parameters: a, b	**Linear SVM** a straight boundary with the **widest margin** (dashed) circled: support vectors
+ Feature design add x^2, $x_1 \cdot x_2$… to get curves		
	Polynomial regression $\hat{y} = a \cdot x^2 + b \cdot x + c$ features: x, x^2 still linear in a, b, c	**SVM, polynomial kernel** features: x_1^2, x_2^2, $x_1 \cdot x_2$ → **curved** boundary no straight line works here

Feature design: **transform the inputs** by hand, then keep the same simple linear model

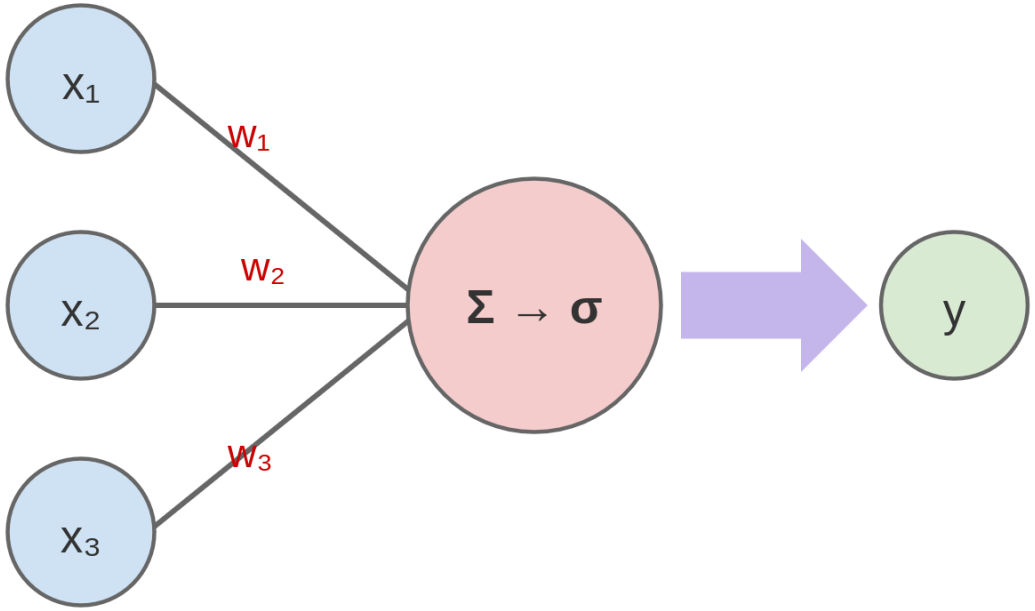
Neural networks, a handy tool

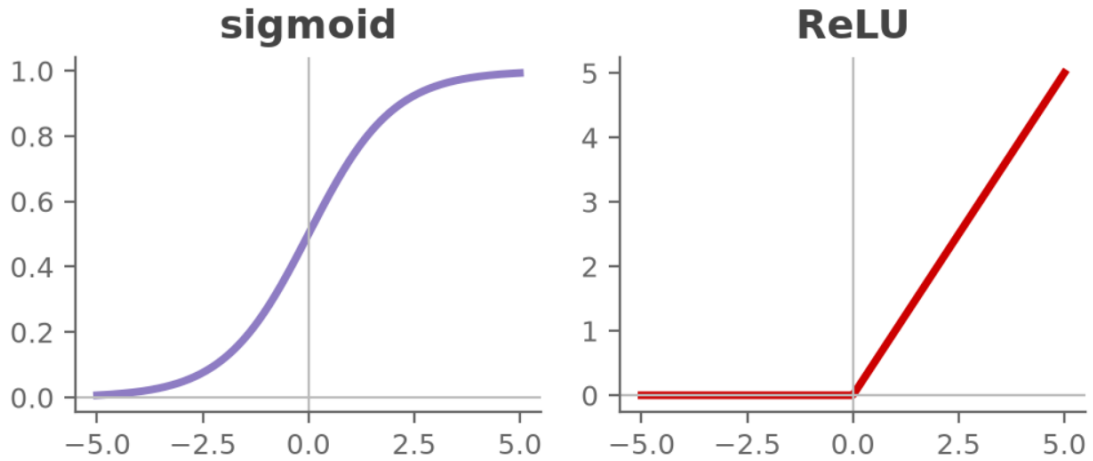
Neural network = a rich model that somehow does not overfit



Once trained,  can approximately reproduce f.

The artificial neuron

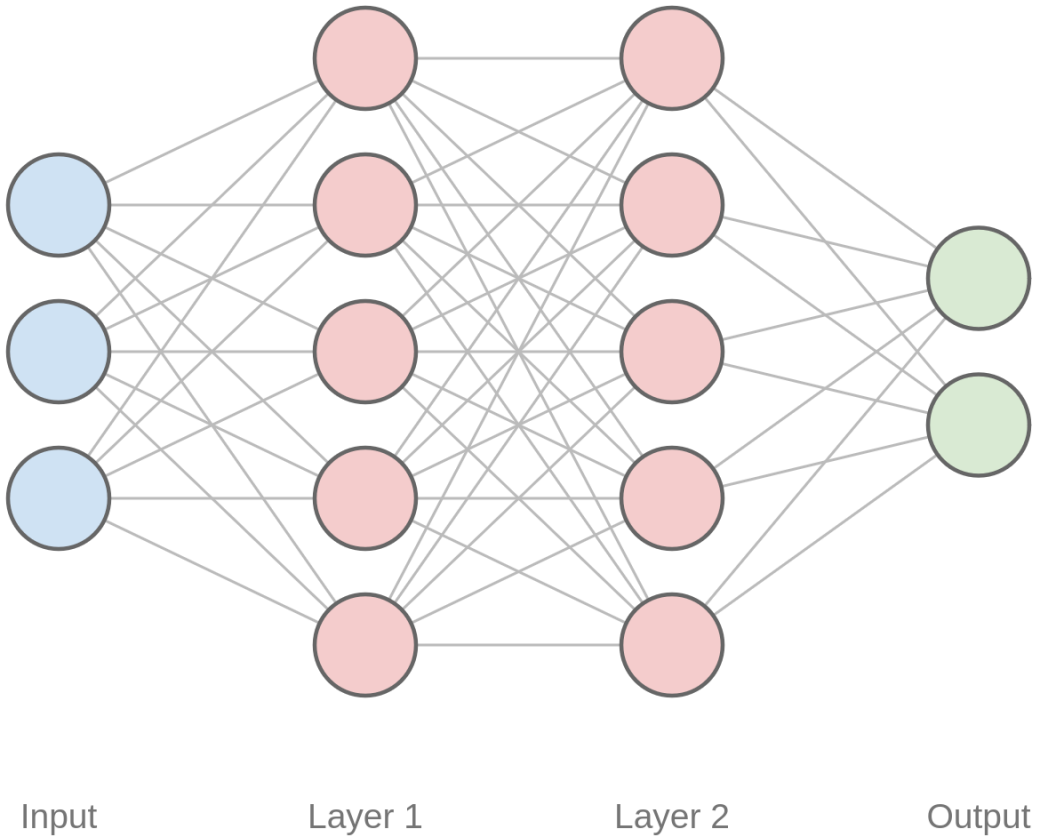




- Weights w and bias b: **learned**
- σ: non-linear **activation** function
- Without σ: just a linear regression!

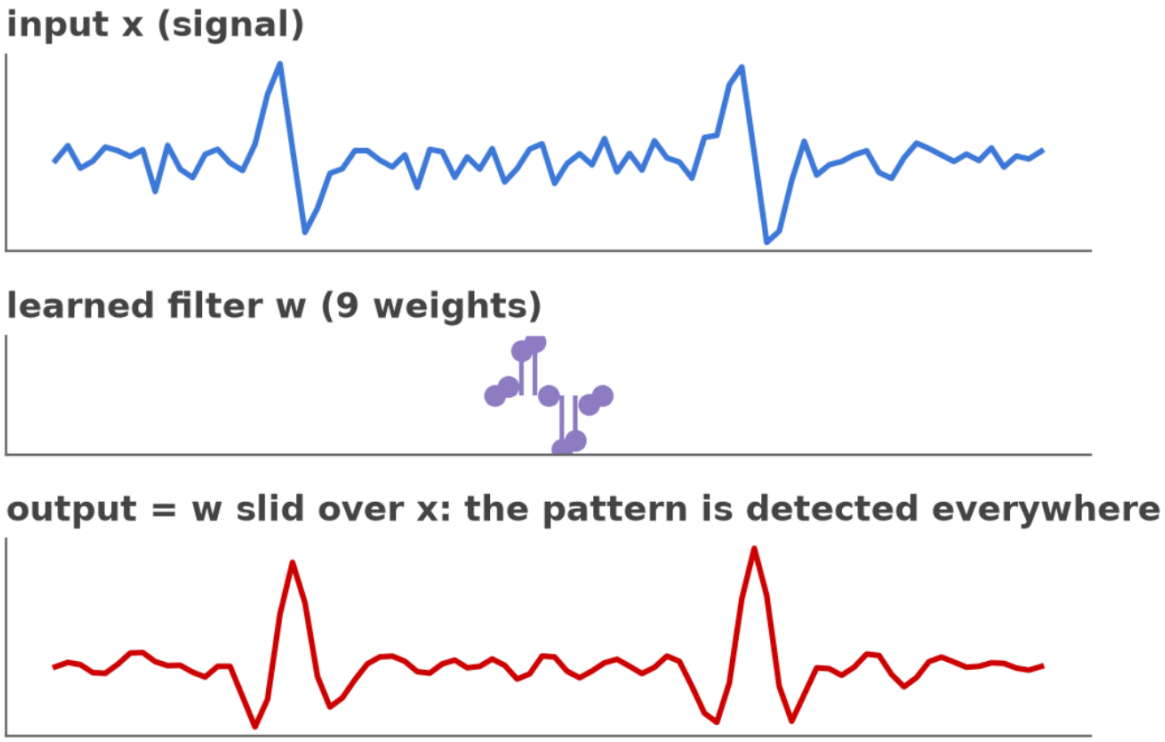
$$y = \sigma(w_1 x_1 + \ \ldots \ + w_n x_n + b)$$

Fully connected network (MLP)



- Stacked layers of neurons: “**deep**” = many layers
- Can approximate any function… in theory
- For audio: **a huge number of parameters**, and no notion of “a pattern that moves in time”

Convolutional networks (CNN)



- A **filter** of a few weights slides over the whole input
- Same weights everywhere: few parameters, pattern recognized **wherever it is**
- **Pooling**: summarize (max) and shrink the size
- Images (2D), spectrograms (2D), waveforms (1D)
- **CREPE is a 1D CNN**

Sequences, attention and Transformers

1 · Cut the input into tokens

Text

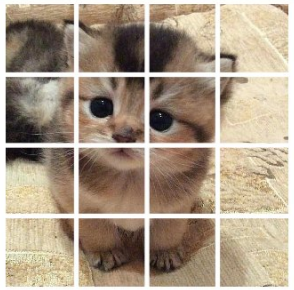
“The cat sat on the mat.”

↓



tokens ≈ words or pieces of words

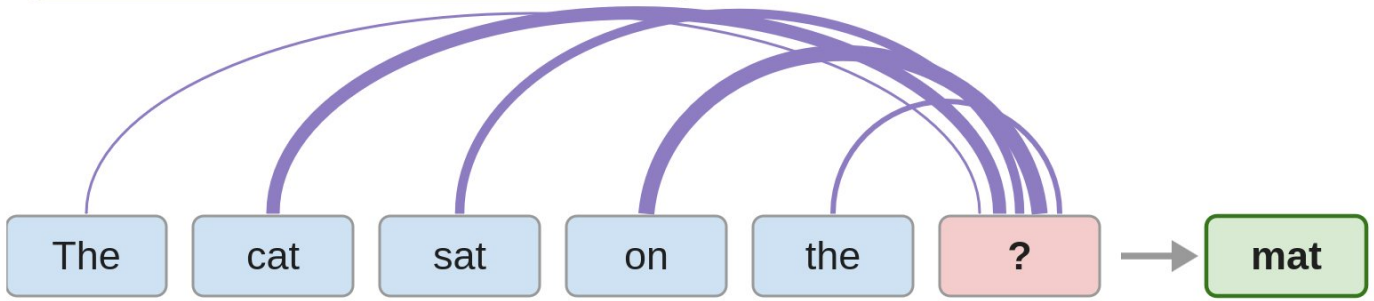
Image



Cut into a grid of **patches**

4 × 4 = **16 tokens**, read like the words of a sentence

2 · Attention: which tokens matter?

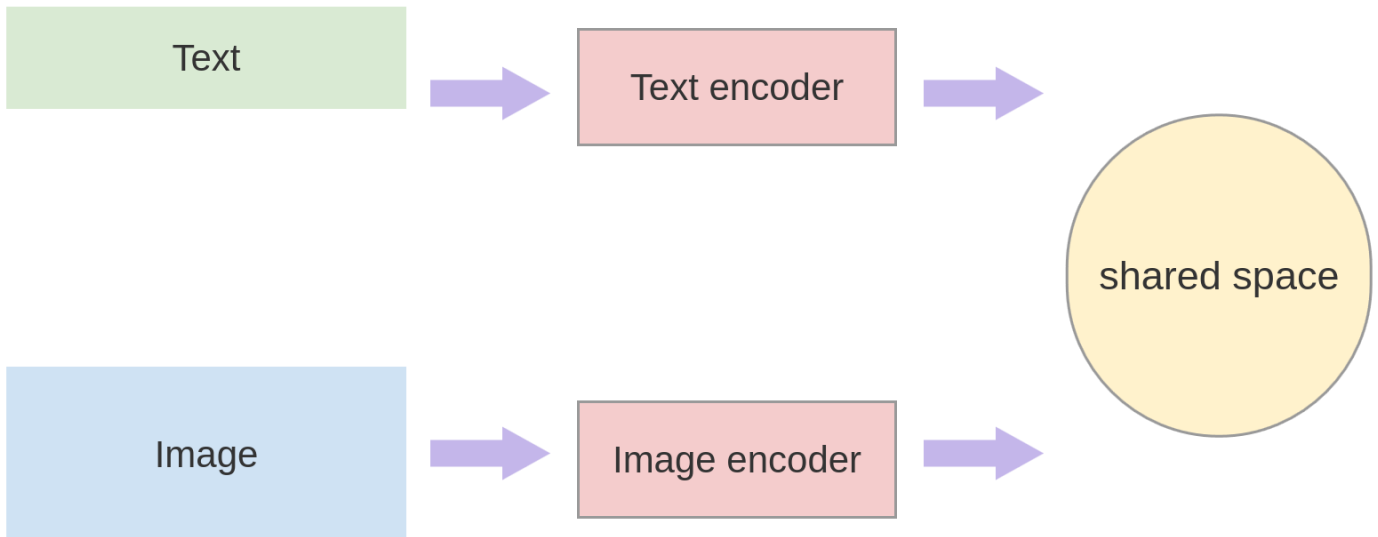


To guess the **next token**, the model looks at **all** the previous ones at once

Arc thickness = attention weight, **learned** from data

Transformer (2017) = tokens + self-supervision + attention → basis of LLMs, image and music generators

Multimodality: linking text and sound



What changes

- scale: data, compute, cost
- the data: rights, biases

What does not change

- data + model + training
- a loss function to minimize

Orders of magnitude

Model	Task	Parameters	Training data
Linear regression	a line	2	a few points
CREPE tiny (2018)	pitch	≈ 0.5 M	≈ 22 h of audio (paper)
CREPE full (2018)	pitch	≈ 22 M	≈ 22 h of audio (paper)
MusicGen (Meta, 2023)	text → music	0.3 to 3.3 B	20,000 h of licensed music
GPT-3 (OpenAI, 2020)	text	175 B	hundreds of billions of words
Suno, Udio…	text → song	not published	not published (lawsuits)

M = million · B = billion. More parameters → more data, more compute… and more energy.

Key points: fundamentals

1

Data

train / validation / test, quality, biases

2

Model

neurons, layers, CNNs, Transformers

3

Training

loss + gradient descent

4

Generalization

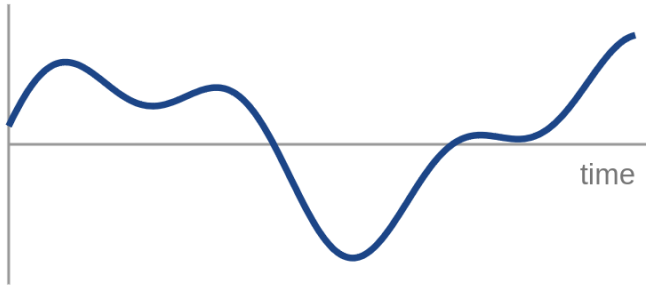
monitor validation, regularize

A neural network = a function with many parameters, fitted on examples

What about audio and music ?

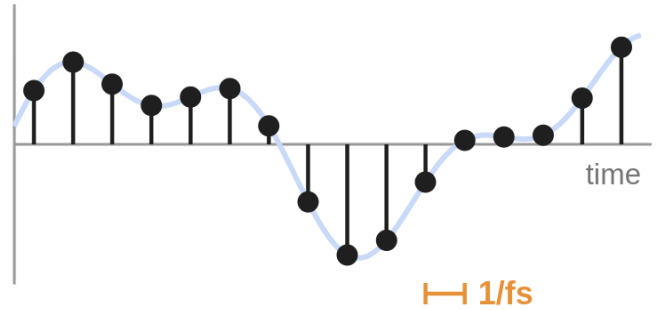
Analog to digital audio signal

1 · Analog signal



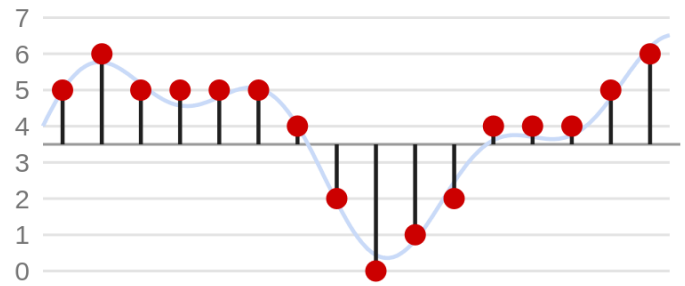
The microphone turns air pressure into a **continuous** voltage

2 · Sampling



Measure it **fs times per second** (sampling rate)

3 · Quantization



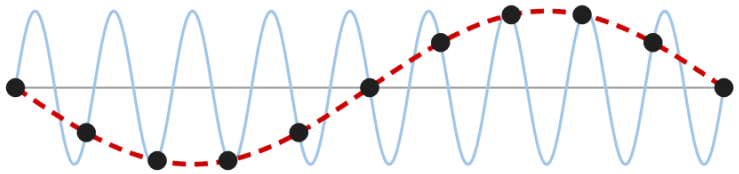
Round each value to the nearest **level** (here 3 bits = 2^3 = 8 levels)

A sound = a sequence of numbers

```
x = [5, 6, 5, 5, 5, 5, 4, 2, 0, 1, …]
```

This vector is what a computer, or a neural network, actually sees

Sample fast enough! (Nyquist)

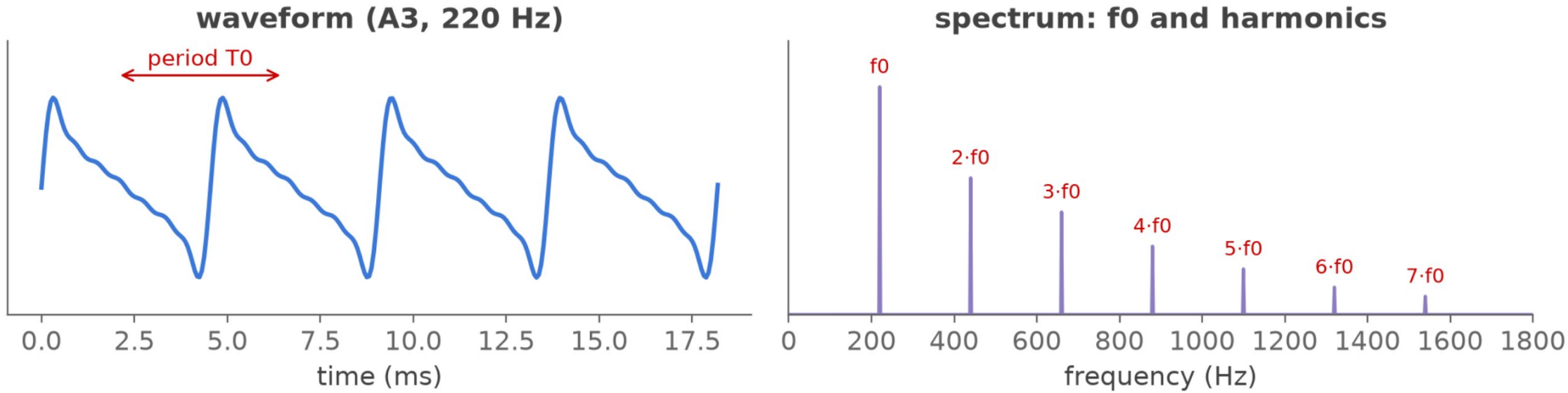


Too few samples: a **high** note looks like a **low** one

fs > 2 × highest frequency

CD quality: **44,100** samples/s × **16 bits** → 1 minute of stereo ≈ **5.3 million numbers**

Pitch, fundamental and harmonics

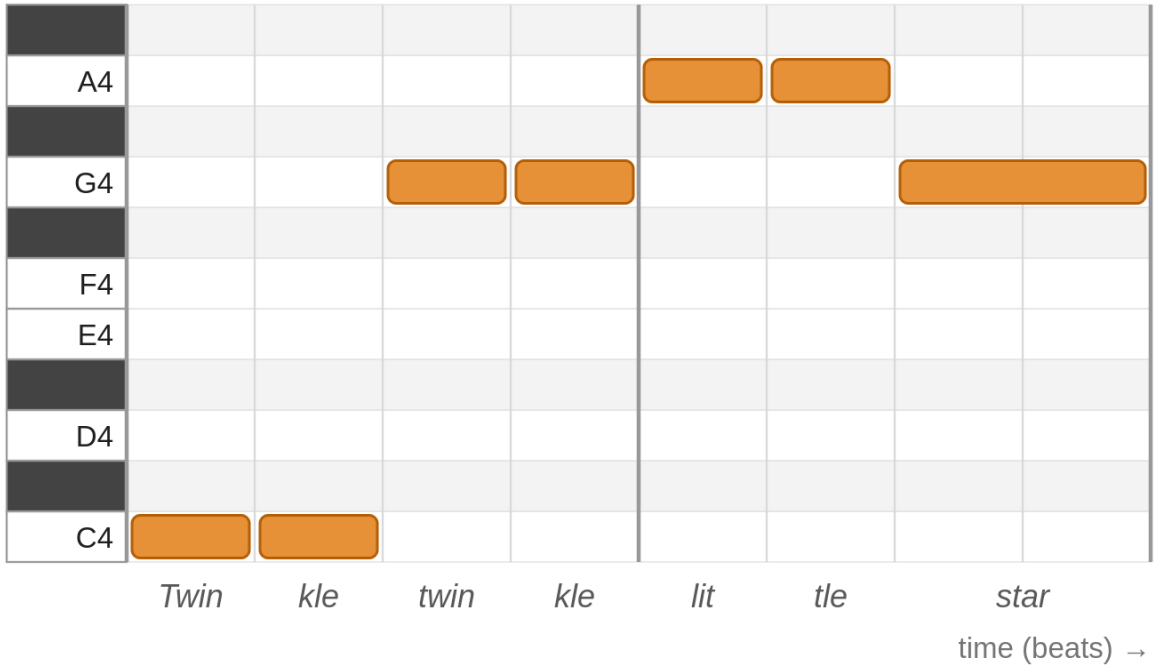


- Periodic sound with period T0 → **fundamental f0 = 1 / T0**, plus harmonics at 2·f0, 3·f0…
- **Pitch** = perception · **f0** = physical measurement · **timbre** depends on the harmonics

Symbolic music: MIDI

MIDI does not store the **sound**, it stores the **notes**: which key, when, how long, how hard

Piano roll (what you see in a music software)



The same notes, as a MIDI file stores them

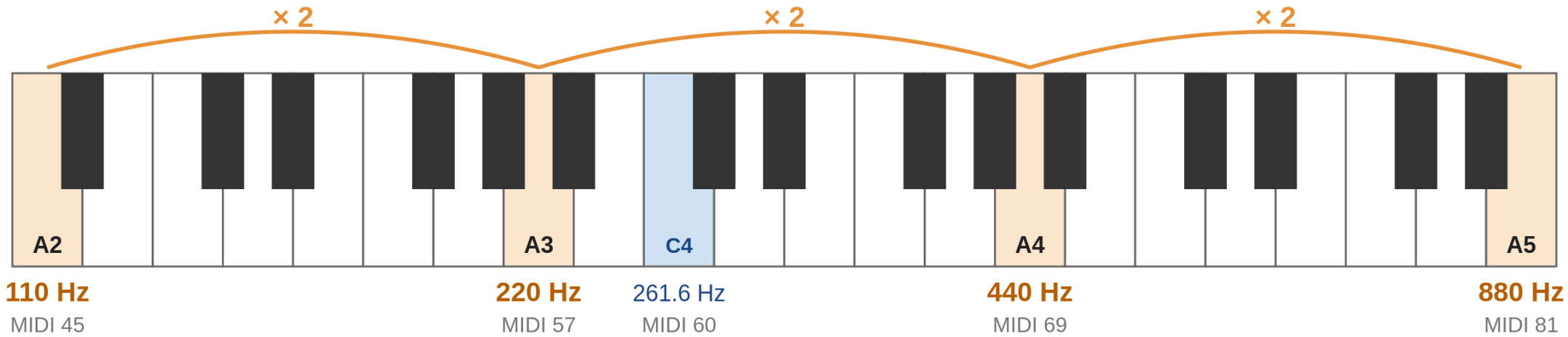
Note	MIDI #	Start	Length	Velocity
C4	60	0	1	90
C4	60	1	1	80
G4	67	2	1	90
G4	67	3	1	80
A4	69	4	1	95
…	…	…	…	…

MIDI #: one number per key, +1 per semitone (C4 = 60, A4 = 69)

Velocity: how hard the key is hit, from 0 to 127

A few text info per note, playable by **any instrument**

From frequency f0 to notes



Octave = frequency × 2

12 semitones per octave, each one × 1.059 (= 2 to the power 1/12)

$$f = 440 \cdot 2^{(m-69)/12} \text{ Hz}$$

m = MIDI number, A4 = 69 = 440 Hz

Between two keys: cents



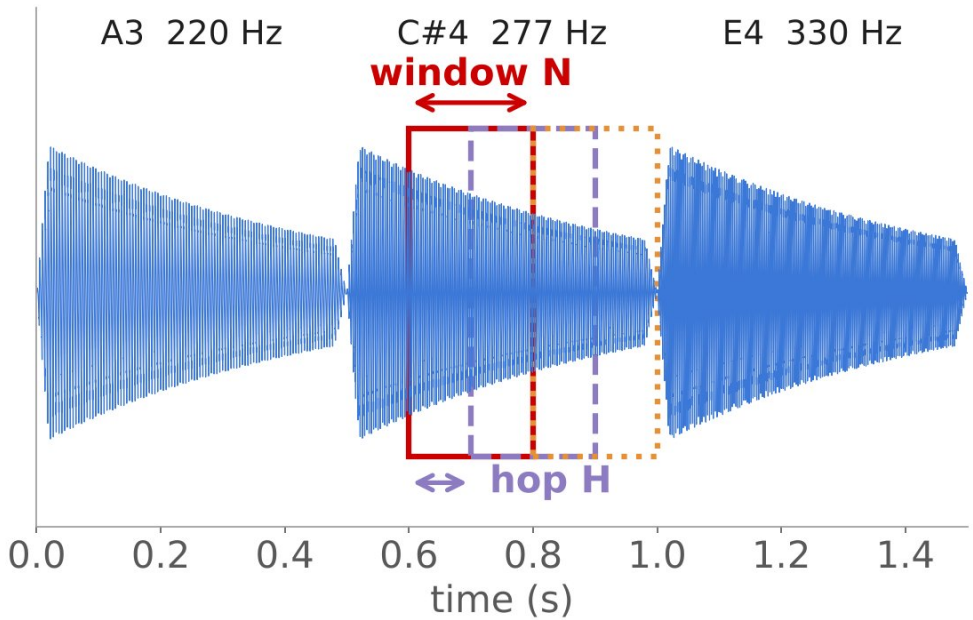
1 semitone = 100 cents

1 octave = 1200 cents · for vibrato, microtones, other tunings

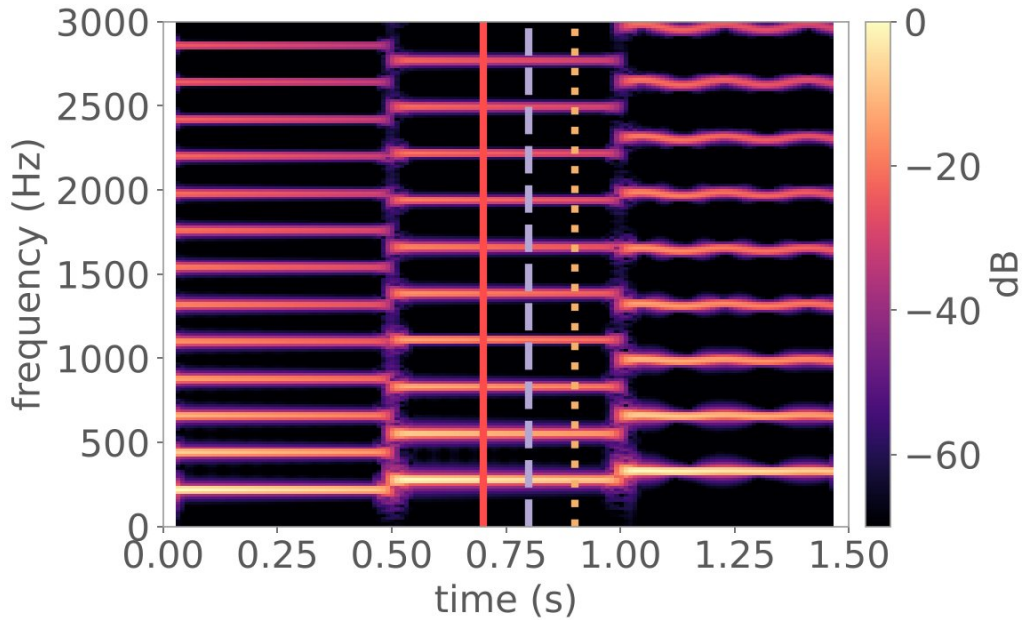
Humans hear **ratios**: pitch lives on a **logarithmic** scale

From signal to spectrogram (STFT)

Signal → Frames (window N, hop H) → FFT of each frame → Spectrogram |X(t, f)| in dB



Time signal: cut into overlapping **frames**



Spectrogram: **each frame becomes one column** · [live demo]

Data in music?

x
input

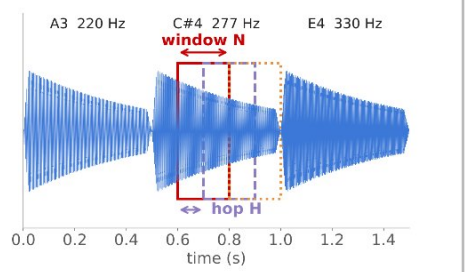
→

f
model, parameters θ

→

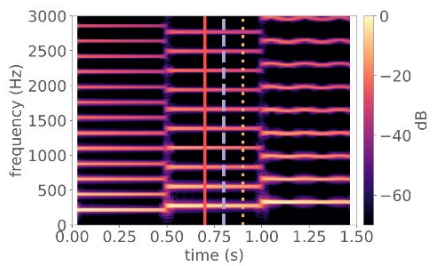
y
target

What is x?



Time signal

a vector of samples



Spectrogram

an image: frequency × time

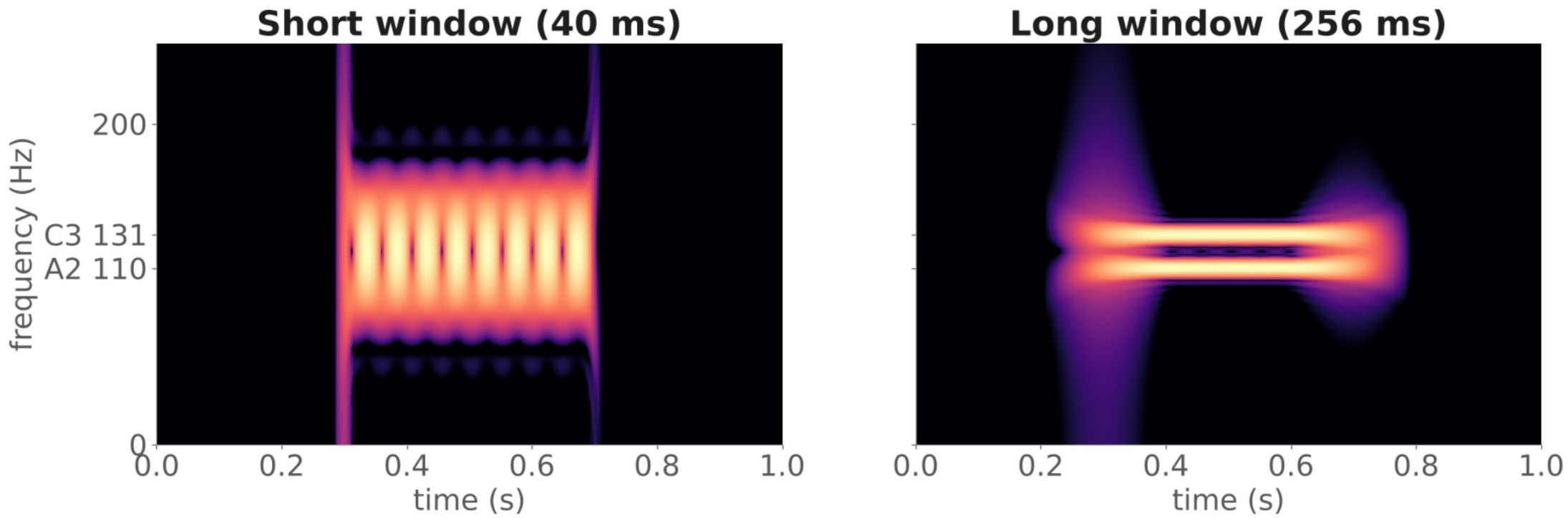
What is y? It depends on the task

Task	Target y	Problem
Transcription	the notes played (MIDI)	classification
Pitch tracking	f0 in Hz, every few ms	"regression"?
Instrument recognition	piano, flute, voice…	classification
Separation, generation	another spectrogram or sound	regression

Same recipe as for cats and dogs: examples **(x, y)**, a model **f**, a loss, gradient descent

The time-frequency trade-off

A two-note chord, A2 (110 Hz) + C3 (131 Hz), played from 0.3 s to 0.7 s

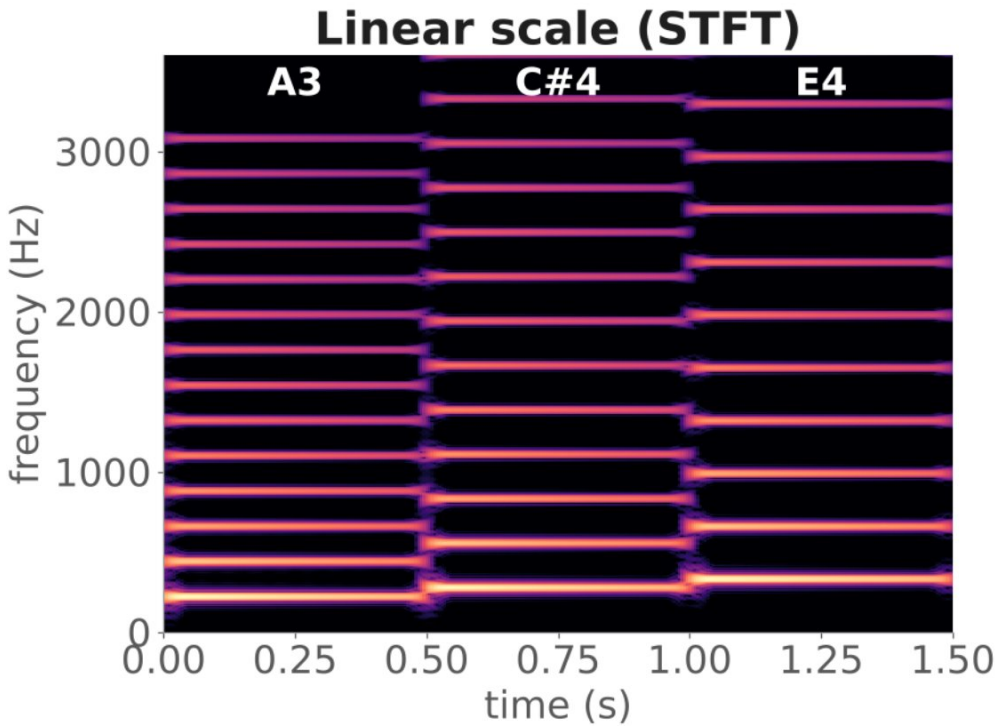


Precise in time

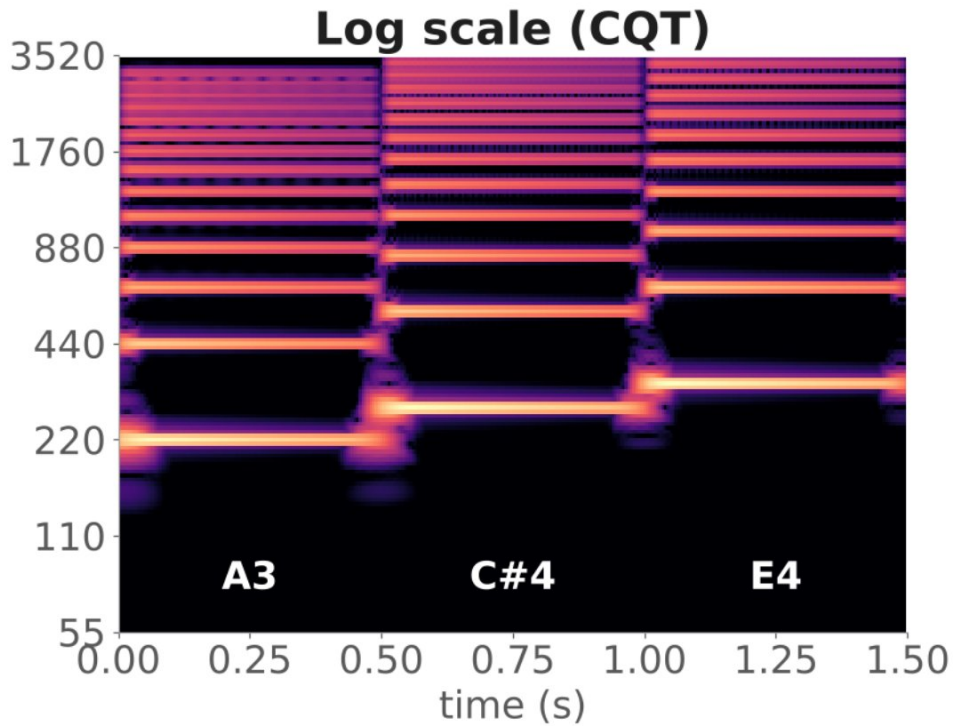
2 notes separated

Longer window = finer in frequency, coarser in time: **you cannot have both**

Logarithmic scales: Mel and CQT



Harmonics f0, 2·f0, 3·f0… **equally spaced in Hz**: the spacing **changes** with the note

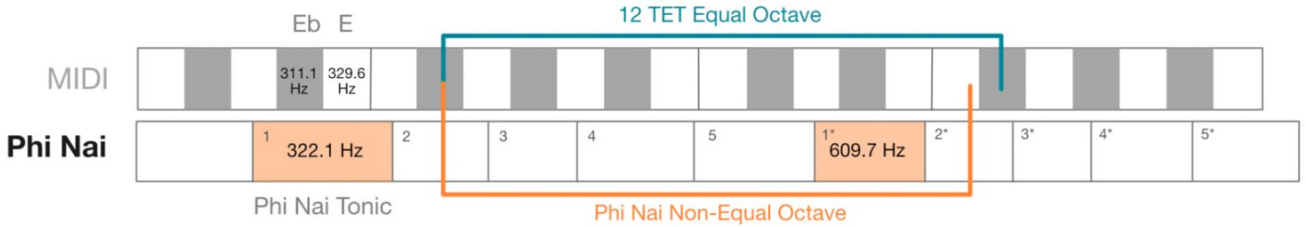


The **same pattern** for every note, just shifted

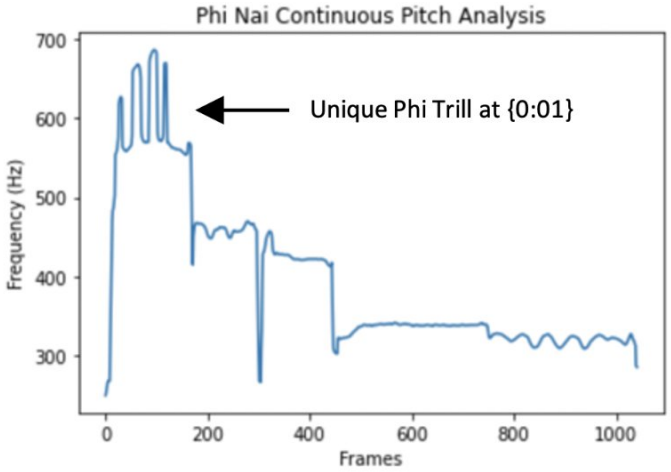
CQT: a fixed number of bins per octave (here 24) · **Mel**: a similar perceptual scale, the most common network input

Yaboi Hanoi and machine learning

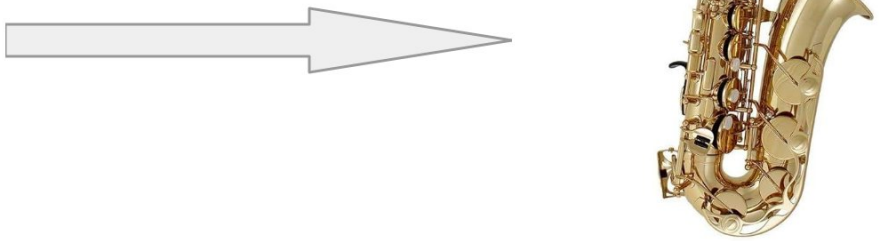




Fundamental detection



Timbre transfer
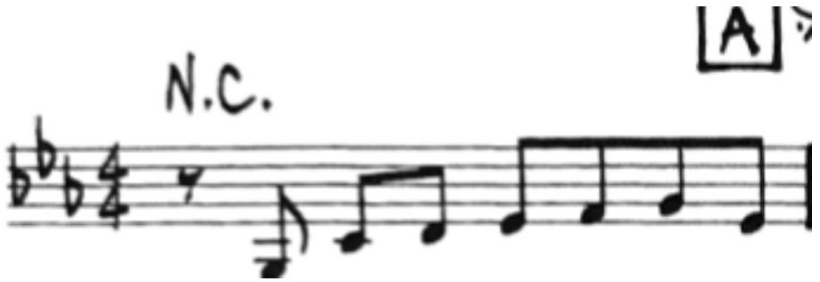




Yaboi Hanoi and machine learning



Watch the video on YouTube

Note recognition vs f0 estimation





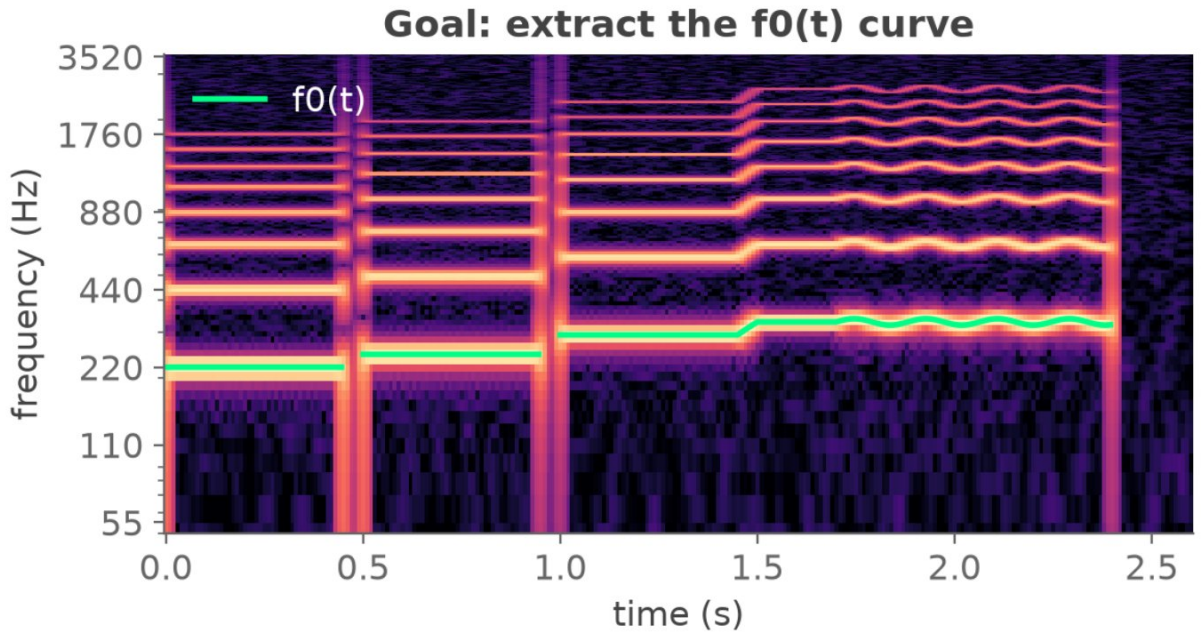
f0 estimation (today)

- **monophonic** sound
- one pitch value every 10 ms
- e.g. YIN, pYIN, **CREPE**

Note transcription

- often **polyphonic**
- notes: pitch, onset, duration
- e.g. Basic Pitch

The problem: estimating f0(t)



Input: a **monophonic** signal (one note at a time)
Output: f0 every 10 ms + **voiced / unvoiced** (confidence)

What is it for?

- Pitch correction (autotune)
- Transcription, scores
- Analysis: ornaments, ragas, vibrato
- Driving a synth (timbre transfer)
- Annotating datasets
- Voice and ear training

Transcription and f0 estimation: challenges

- Polyphonic instruments (guitar, piano)
- Recording conditions (reverberation…)
- Instrument timbre (two pianos sound different)
- Bends (guitar, voice, winds…)
- Not only pitch, but also note duration!

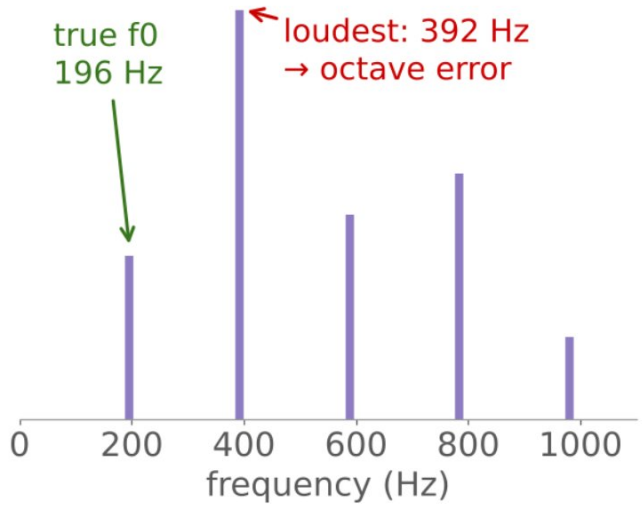
There are dozens of scientific papers on the topic, and as many algorithms!

(Probably) the best-known algorithm: YIN (no learning)

2002, Alain de Cheveigné (IRCAM) and Hideki Kawahara (Wakayama University)

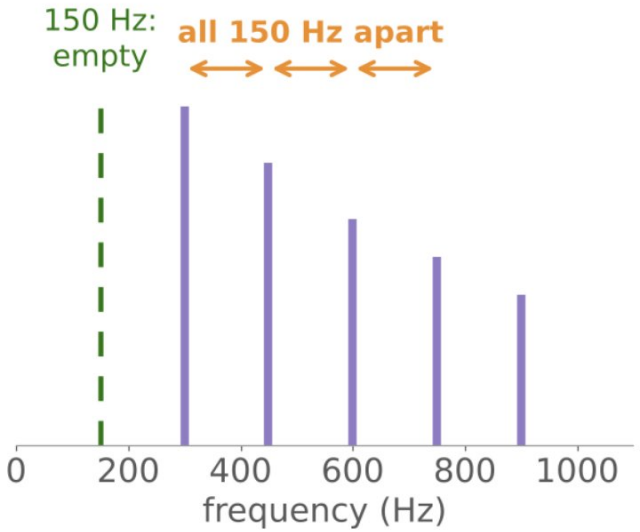
Traps in the spectrogram

Dominant harmonic



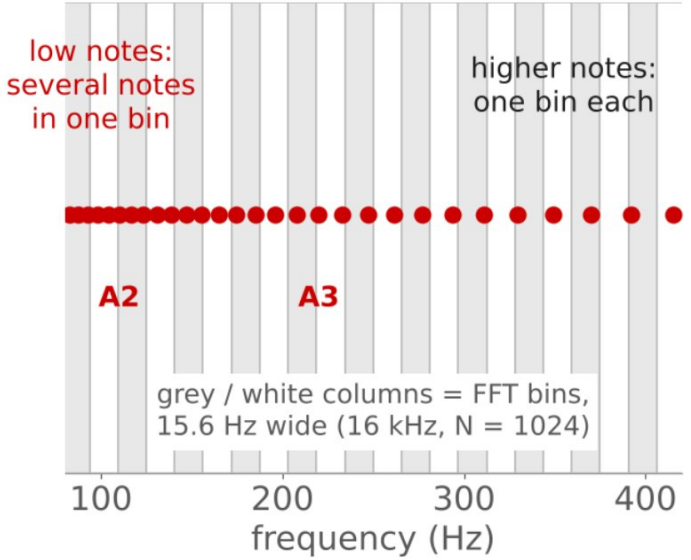
The loudest peak is not always f0: **octave errors**

Missing fundamental



No energy at 150 Hz (e.g. small speaker), yet **we hear 150 Hz**
pitch = the **spacing** of the harmonics

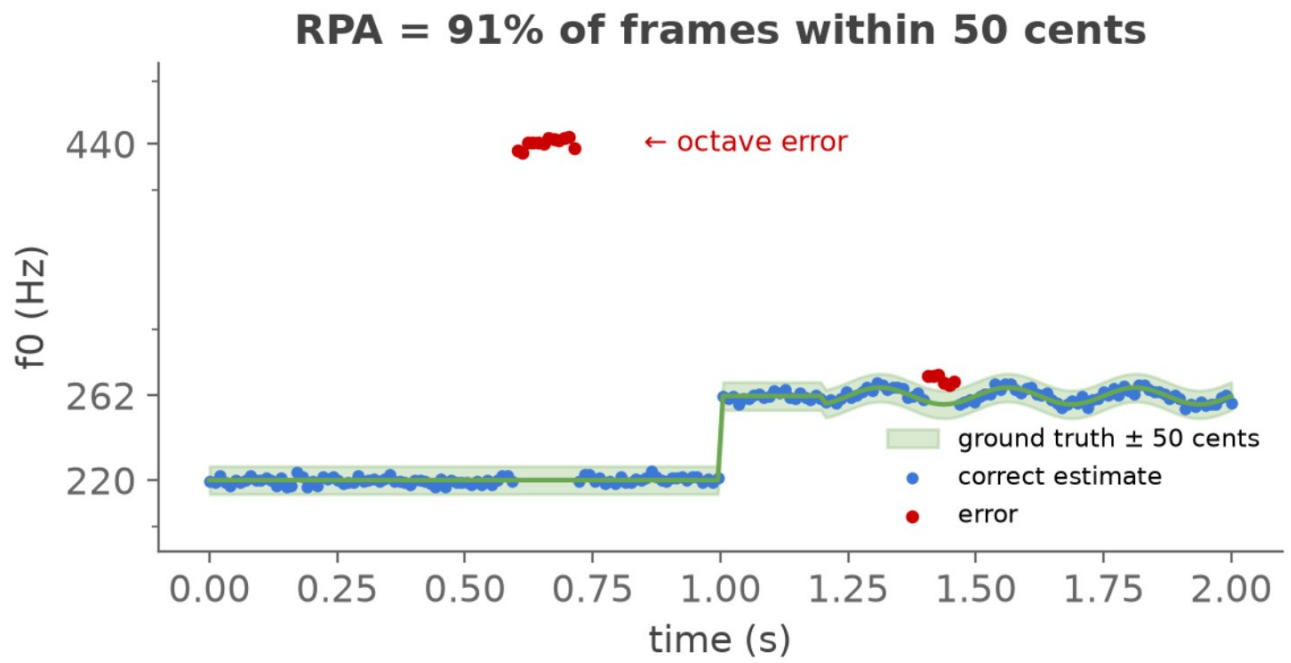
FFT resolution



Low notes are a few Hz apart: **too close** for the FFT grid
Similar problem with fast repeated notes

Also: vibrato and glides, note transitions, breath and consonants, noise, reverb, **inharmonic** instruments ((old) pianos, drums, bells), very low or very high notes

Evaluating a pitch detector



Challenge: getting a **perfect ground truth** (manual annotations are imprecise)

- **RPA**: Raw Pitch Accuracy

% of frames within **50 cents** of the truth

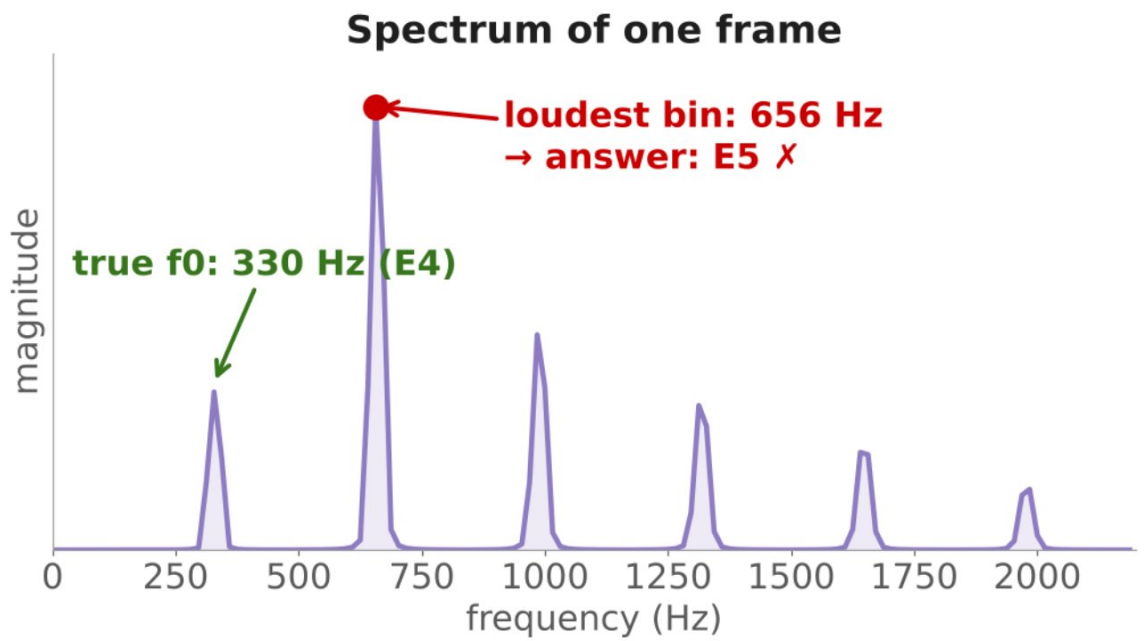
- **RCA**: Raw Chroma Accuracy

same, but forgiving octave errors

- Tool: **mir_eval** (Python)

Baseline 1: the spectral peak

Spectrogram → Loudest bin in each frame → f0(t) = frequency of that bin



One frame of an E4 whose 2nd harmonic is the loudest

Problems

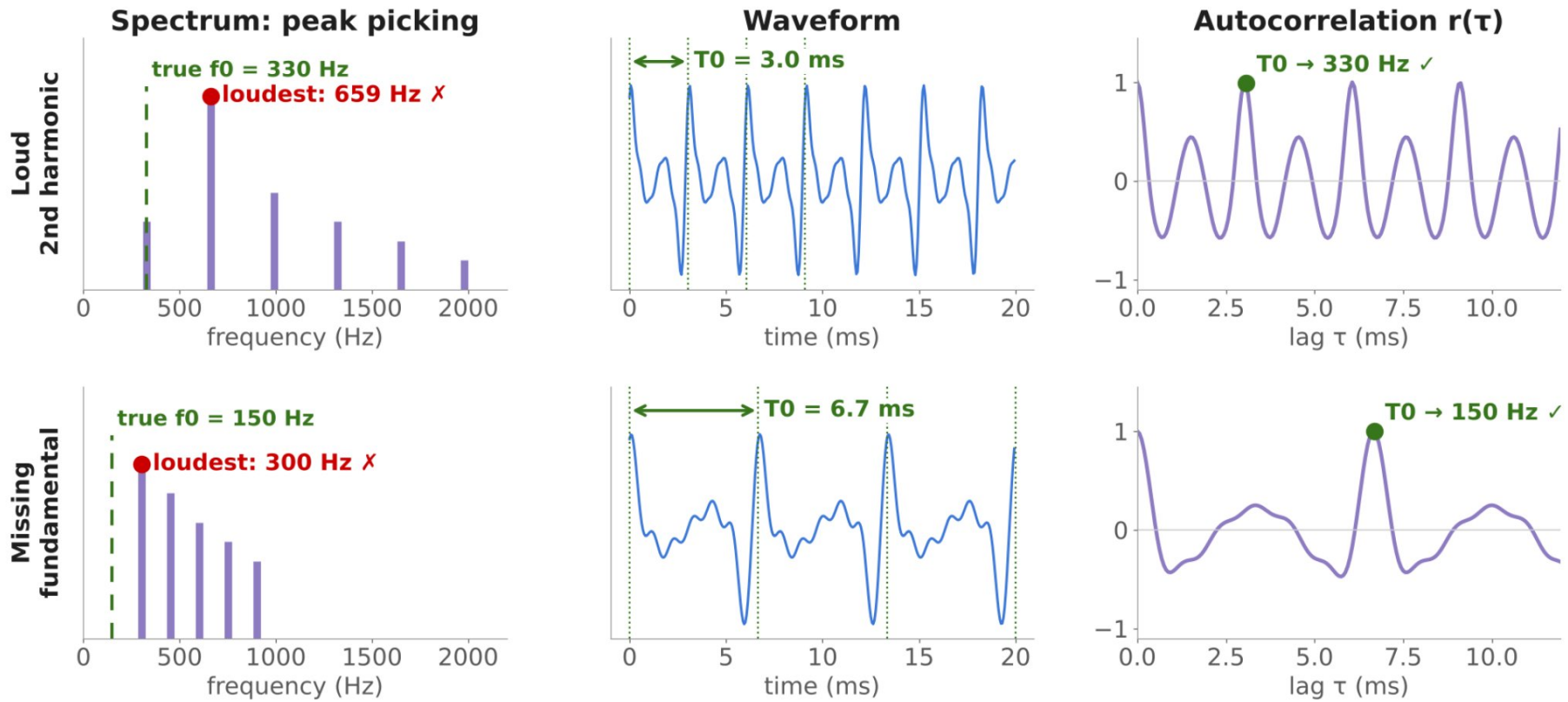
- **Octave errors**: harmonic 2 can dominate (as here)
- Missing fundamental
- Noisy environement

Improvements

- Change representation: CQT, mel...
- Harmonic sum / product: favor f0 when 2·f0, 3·f0… are present

Baseline 2: autocorrelation

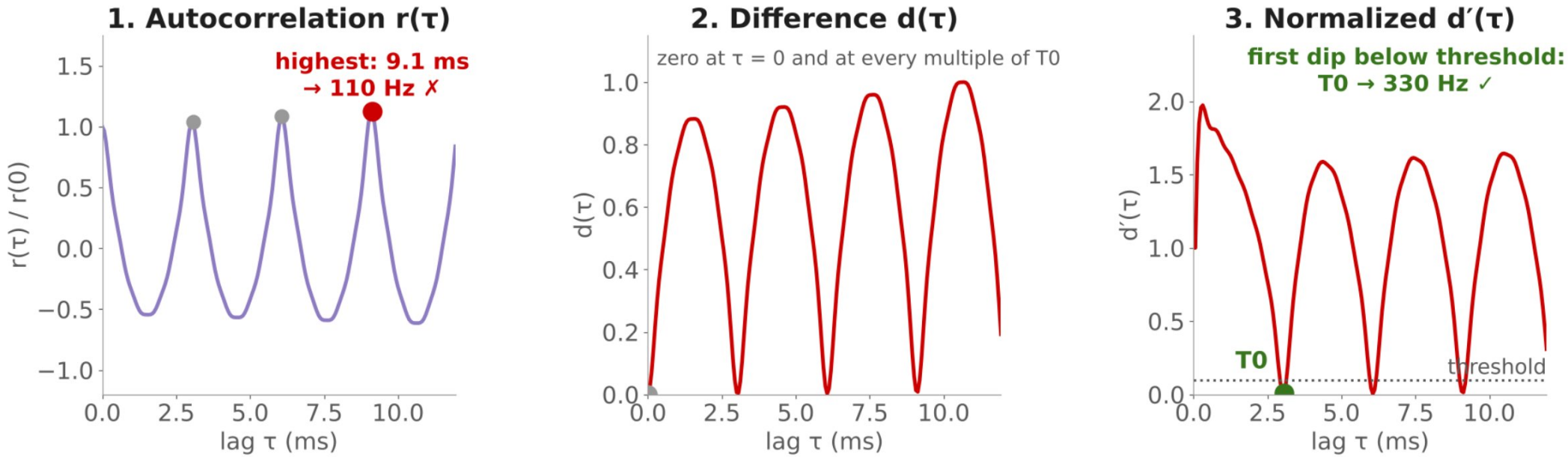
Shift the signal by τ, multiply with itself, sum: **r(τ) = Σ x_j · x_(j+τ)** is largest when the shifted copy **lines up**, at τ = T0



The spectral peak looks at **one** harmonic; autocorrelation looks at the **period of the whole waveform**

Baseline 3: YIN, what it adds

Problem: r(τ) has peaks at T0, 2·T0, 3·T0… **which one?** Example: an E4 (330 Hz) getting louder



1 · Difference

d(τ) = Σ (x_j − x_(j+τ))²: zero when the copy lines up

2 · Normalize

d′ = d / its running mean: starts at 1, no false dip at τ = 0

3 · Threshold

take the **first** dip below 0.1, not the deepest: avoids 2·T0, 3·T0

4 · Interpolate

fit a parabola around the dip: finer than one sample

No learning, a few well-chosen **heuristics** (2002). Limits: noise, reverb, several notes at once

The state of the art before 2018

1967	**Cepstrum**	Noll · periodicity of the log-spectrum
1976-1995	**Autocorrelation, RAPT, Praat**	speech · normalized correlation
2002	**YIN**	de Cheveigné and Kawahara · normalized difference
2008	**SWIPE**	Camacho and Harris · sawtooth template
2014	**pYIN**	Mauch and Dixon · probabilistic YIN + HMM

Common point: hand-designed **heuristics**

CREPE: learn from data

CREPE (2018)

"CREPE: A **C**onvolutional **RE**presentation for **P**itch **E**stimation"

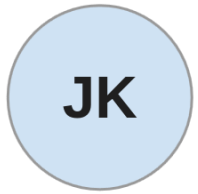
Kim, Salamon, Li, Bello · ICASSP 2018 · NYU

- A **CNN** directly on the **waveform** (no spectrogram)
- **Learned** from annotated data, no heuristics
- Better than pYIN and SWIPE, more robust to noise
- **Open source** + pre-trained model: pip install crepe

The recipe in 4 points

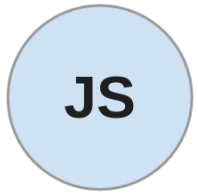
1. Input: 1024 samples
2. 6 convolutional layers
3. 360 outputs = 360 pitches
4. Gaussian target + cross-entropy

A word on the research behind CREPE



Jong Wook Kim

PhD student, NYU MARL (later OpenAI)



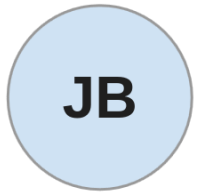
Justin Salamon

NYU MARL and CUSP (later Adobe Research)



Peter Li

NYU MARL

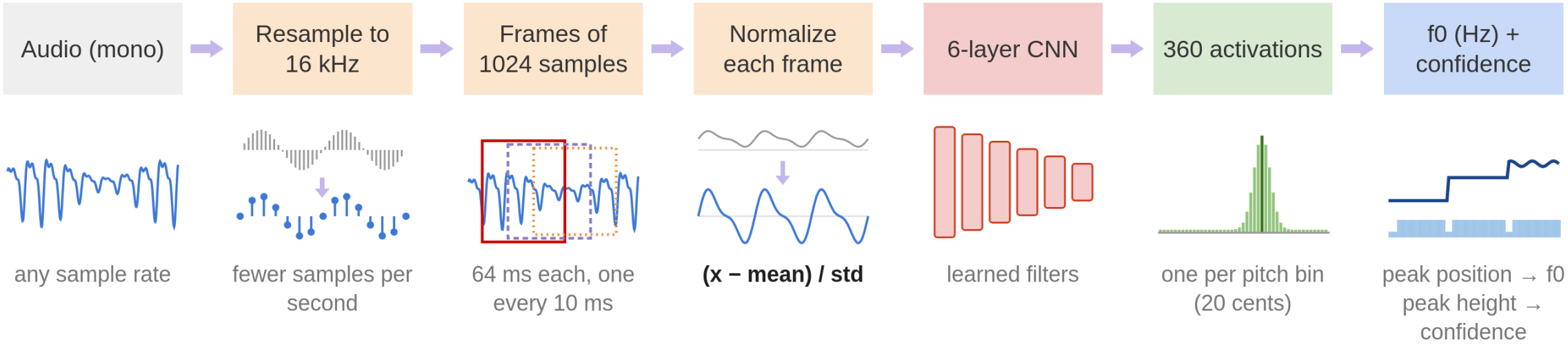


Juan Pablo Bello

professor, director of MARL, NYU

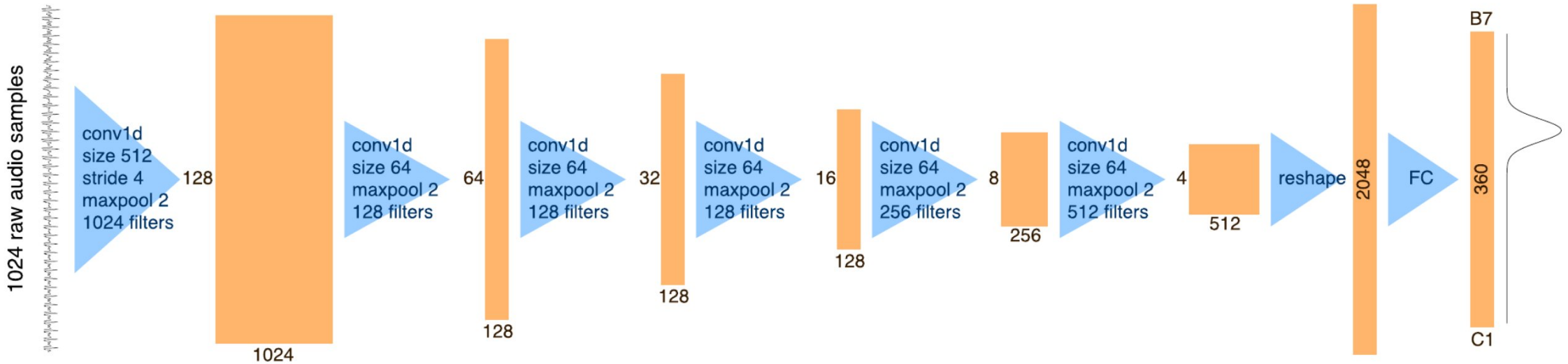
- An ecosystem: **MedleyDB** (data), **mir_eval** (metrics), deep salience (Bittner et al., 2017)
- Justin Salamon: PhD on melody extraction (MELODIA, MTG Barcelona)
- **Open** research: paper, code and model released → reused everywhere (DDSP, annotation…)

CREPE: overview



- 1024 samples at 16 kHz = **64 ms**: at least 2 periods of the lowest note (C1, T0 = 30.6 ms)
- Normalization → insensitive to **loudness**: a quiet and a loud note look the same to the network
- Each frame is processed **independently**

Network architecture



Kim et al., ICASSP 2018, Fig. 1

- 6 **conv1d** layers + ReLU + **max-pooling 2**
- Batch normalization, **dropout 0.25**
- 1st layer: 1024 filters of **512** samples, stride 4
- Then filters of width 64 · dense layer → **360 sigmoids**

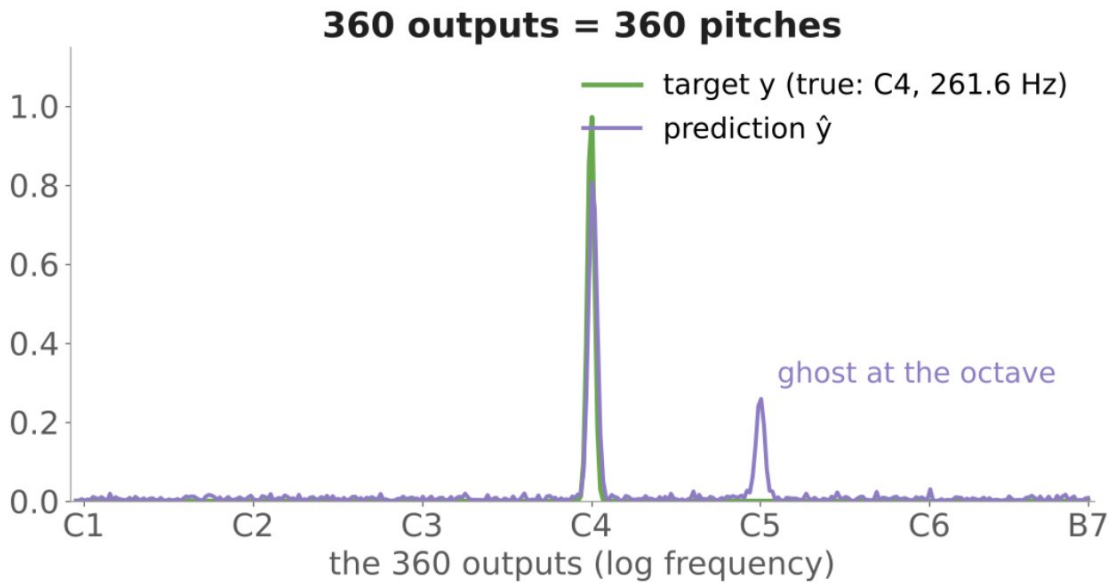
Architecture: sizes and parameters

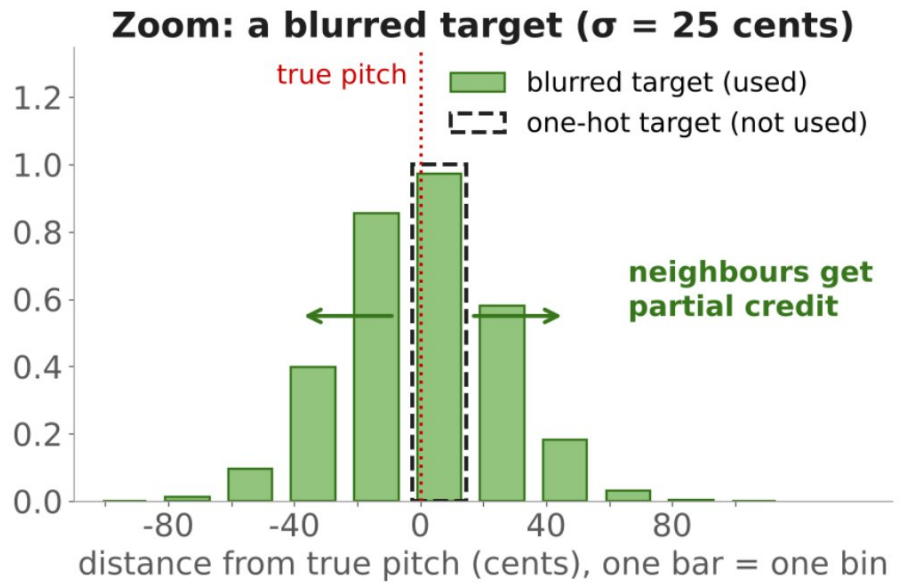
Layer	Filters × width	Output	Parameters
Input	–	1024 × 1	–
Conv 1 (stride 4) + pool	1024 × 512	128 × 1024	0.53 M
Conv 2 + pool	128 × 64	64 × 128	8.39 M
Conv 3 + pool	128 × 64	32 × 128	1.05 M
Conv 4 + pool	128 × 64	16 × 128	1.05 M
Conv 5 + pool	256 × 64	8 × 256	2.10 M
Conv 6 + pool	512 × 64	4 × 512	8.39 M
Flatten + dense	–	2048 → 360	0.74 M
Total			≈ 22.2 M

Capacity	× filters	Params
tiny	4	0.5 M
small	8	1.6 M
medium	16	5.9 M
large	24	12.8 M
full	32	22.2 M

Choosing the capacity: speed vs. accuracy

Output: 360 pitch classes





- 360 bins of **20 cents**, 6 octaves: **C1** → **B7** (32.7 → 1975.5 Hz)
- Sigmoid output: a score between 0 and 1 per pitch

Why classify rather than regress?

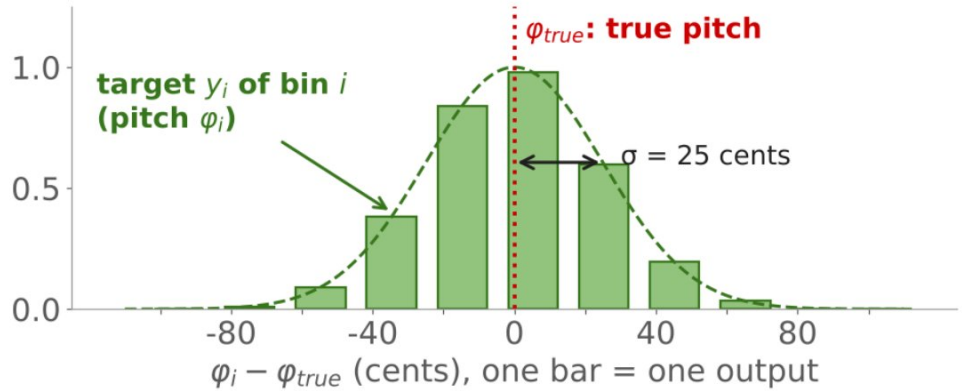
- Express **hesitation** (which octave?)
- Get a **confidence** → voicing

Gaussian target and loss function

Target: the true pitch blurred by a Gaussian (σ = 25 cents)

$$y_i = \exp\left(-\frac{(\phi_i - \phi_{true})^2}{2 \cdot 25^2}\right)$$

An **almost right** prediction is barely penalized

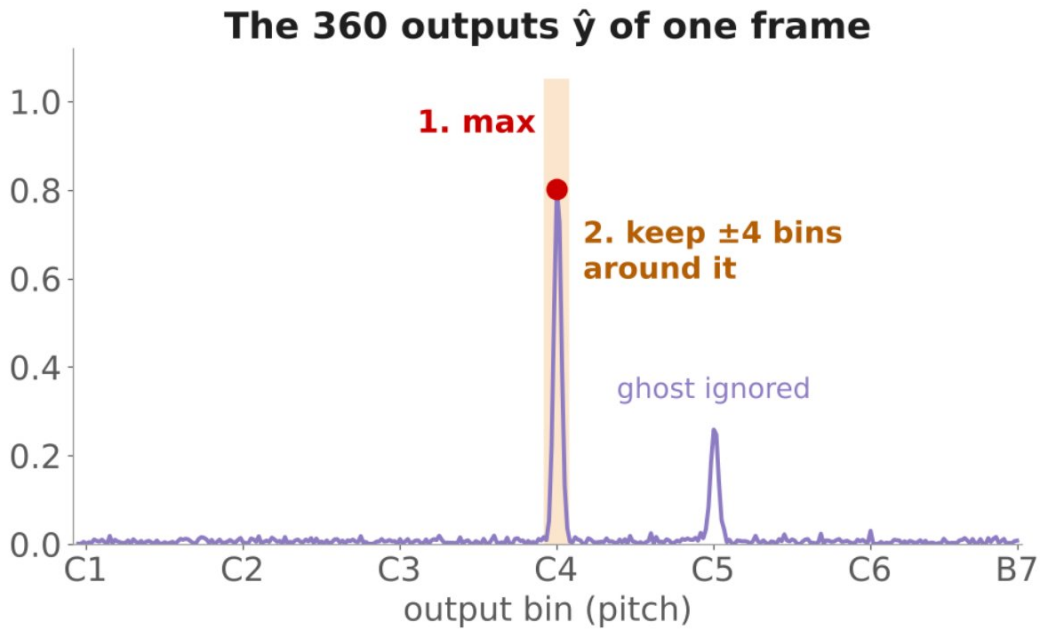


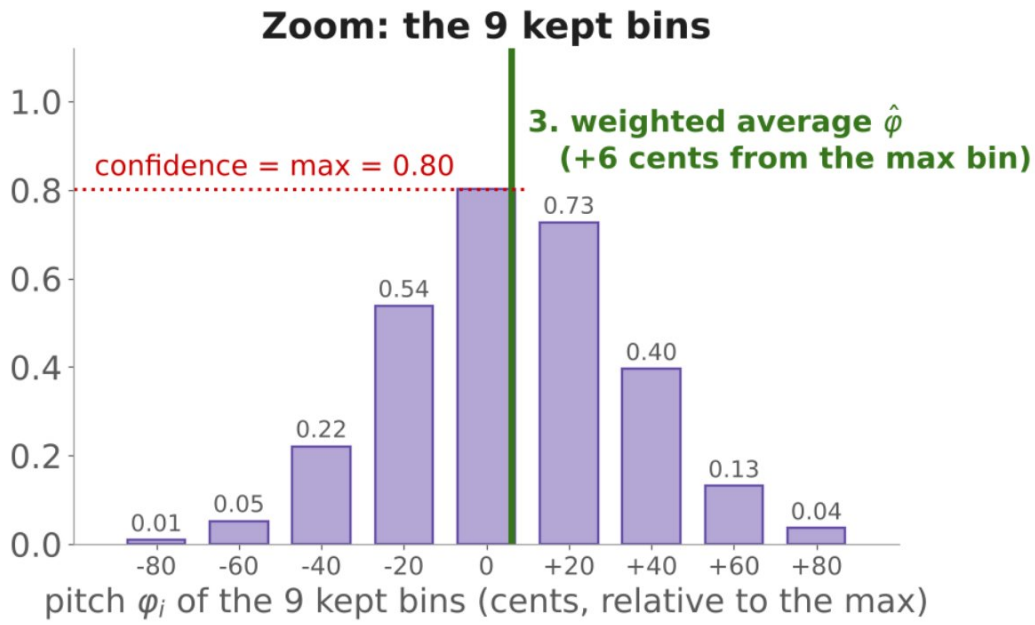
Loss: binary cross-entropy over the 360 outputs

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^{360} -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)$$

- Each output i is a small classification: “is it this pitch?”
- φ_i: pitch of bin i in cents · φ_true: the true pitch · y_i: the blurred target (green bars)

Decoding: from 360 scores to one frequency





$\hat{\varphi} = \Sigma\, \hat{y}_i\, \varphi_i \,/\, \Sigma\, \hat{y}_i$
over the 9 kept bins

$\hat{f} = 10 \cdot 2^{(\hat{\varphi} / 1200)}$ Hz
cents back to Hz

confidence = max $\hat{y}_i$
low → unvoiced (silence, breath)

The average falls **between bins**: precision finer than 20 cents

Training CREPE

Data (paper)	Content
RWC-synth	6.16 h · fixed timbre, synthetic
MDB-stem-synth	230 tracks · 25 instruments · 15.56 h

- **Resynthesis** → f0 known **exactly**
- **5-fold** cross-validation, 60/20/20
- **Artist-conditional** folds for MDB-stem-synth

Optimization recipe

- Adam, $\eta = 0.0002$
- batches of 32 examples
- 1 epoch = 500 batches
- stop if validation stalls for 32 epochs
- Keras

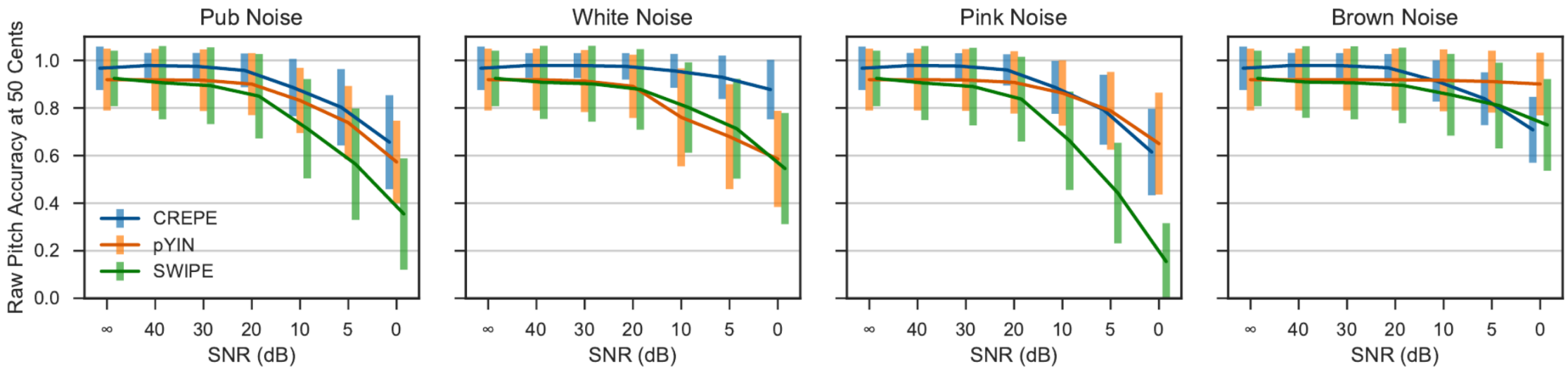
Released model: trained on more data (MIR-1K, Bach10, NSynth…)

Results: accuracy

Data	Threshold	CREPE	pYIN	SWIPE
RWC-synth	50 cents	0,999 ± 0,002	0,990 ± 0,006	0,963 ± 0,023
	25 cents	0,999 ± 0,003	0,972 ± 0,012	0,949 ± 0,026
	10 cents	0,995 ± 0,004	0,908 ± 0,032	0,833 ± 0,055
MDB-stem-synth	50 cents	0,967 ± 0,091	0,919 ± 0,129	0,925 ± 0,116
	25 cents	0,953 ± 0,103	0,890 ± 0,134	0,897 ± 0,127
	10 cents	0,909 ± 0,126	0,826 ± 0,150	0,816 ± 0,165

- Mean RPA ± standard deviation, Kim et al. 2018, Table 2 (best score in bold)
- CREPE's advantage grows with stricter **precision** and more **varied timbres**

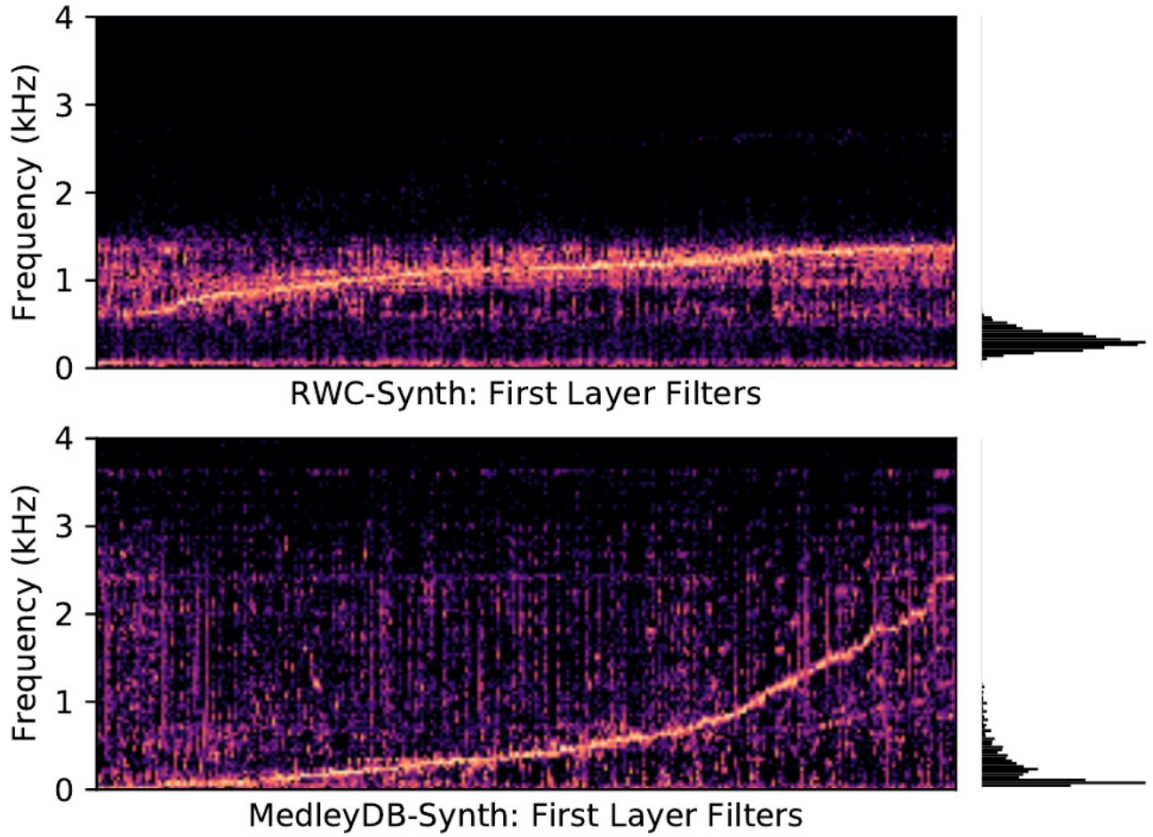
Results: robustness to noise



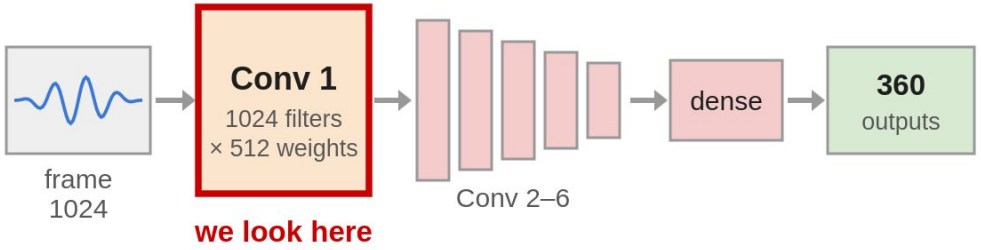
Kim et al., ICASSP 2018, Fig. 2

- CREPE is best below **10 dB** signal-to-noise ratio, and more **stable** (lower variance)
- Exception: **brown** noise (low frequencies), to which pYIN is almost insensitive

What did the network learn?



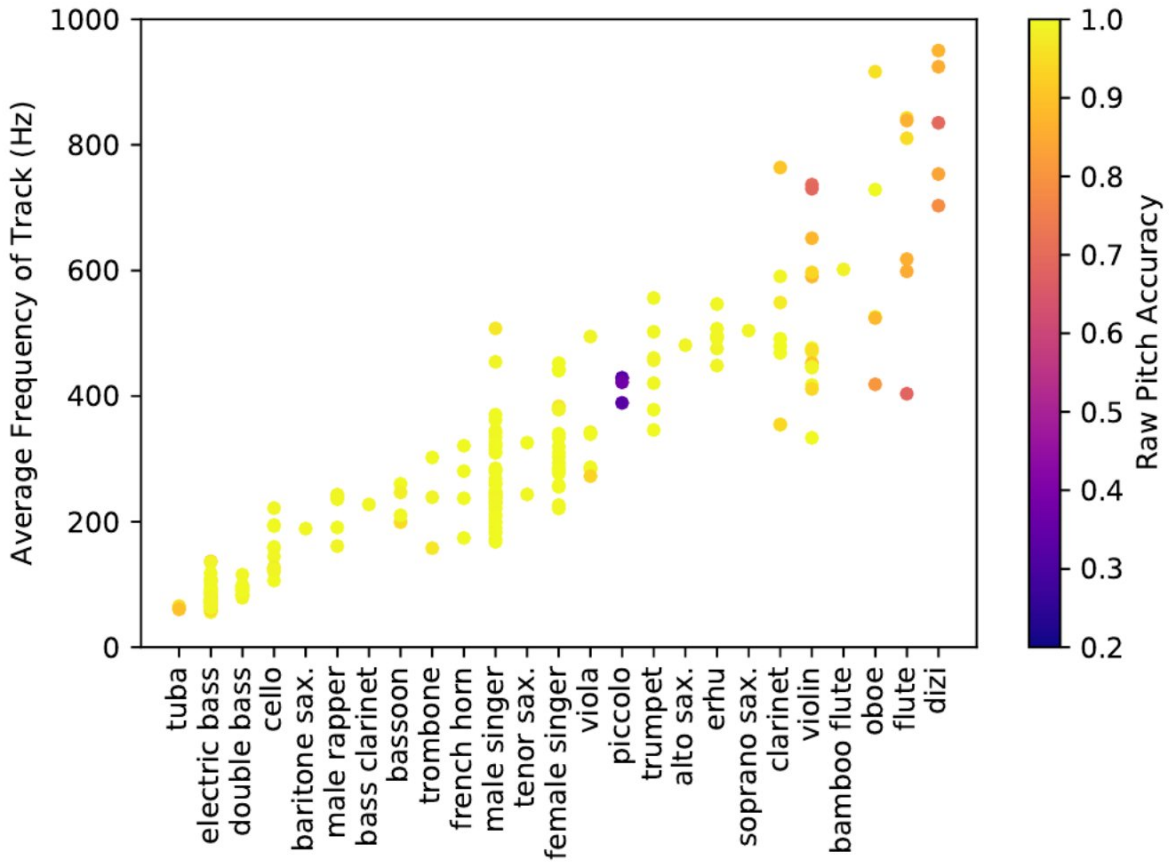
Spectra of the first-layer filters (Fig. 3)



A filter is made of **weights of the network**, not of processed signal. Gradient descent sets them from the training data: **different data, different filters**. The figure: the FFT of each of the 1024 filters, side by side, for **two separate trainings** (RWC-synth, MDB-stem-synth).

- Fixed timbre (RWC): filters tuned to the **harmonics** (600–1500 Hz) rather than to f0
- Varied timbres (MDB): filters also cover the **f0 range**
- The network **adapts to the data** it is shown

Performance per instrument



RPA per track and instrument (Fig. 4)

- Worse on **high-pitched** instruments
- **Dizi** (Chinese flute): timbre absent from training → failure
- **Piccolo**: annotation error in the dataset
- **A model only generalizes to what resembles its training data**

Limitations and caveats

Monophonic	one note at a time; polyphony needs other models (e.g. Basic Pitch)
No context	each frame is independent → add smoothing (they did it later on)
Data	synthetic, few non-Western timbres → generalization
Fixed range	C1 → B7; nothing outside it
Cost	22 M parameters: "slow" on CPU (use tiny, or a GPU) for real time.

What does it cost? 30 s of audio, roughly

3 000 frames
30 s, one frame every 10 ms

→

× 2.8 GFLOP
per frame (1.4 G multiply-adds, mostly Conv 2)

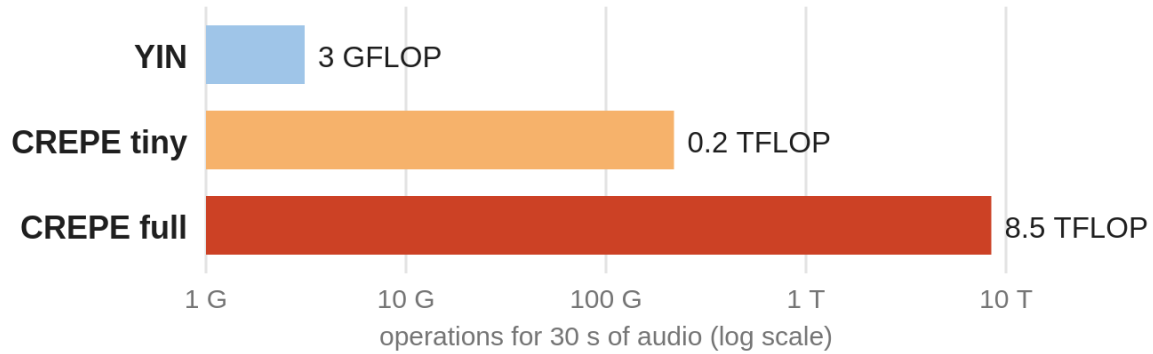
→

≈ 8.5 TFLOP
for the whole clip

→

≈ 0.1 to 1 Wh
laptop CPU: 1–2 min · GPU: a few seconds

Compared with the other methods



Cheaper is not free: tiny is ~40× cheaper than full and YIN ~3 000× cheaper, but both are **less accurate**

≈ a 10 W LED bulb for 1 to 6 minutes
0.1 to 1 Wh of electricity

≈ 1 % to 7 % of a phone charge
a smartphone battery holds about 15 Wh

≈ 0.005 to 0.05 g of CO_2 in France
grid at about 50 g CO_2 per kWh; several times more elsewhere

Running the model is cheap.
Training (many passes over hours of audio, hyperparameter searches) and the **hardware** itself cost more.

After CREPE

torchcrepe	PyTorch port of CREPE (handy for the lab)
SPICE (Google, 2019)	self-supervised: learns pitch without labels, from transpositions
Inverse synthesis (Engel et al., 2020)	self-supervised, with a differentiable synthesizer (DDSP, day 2)
PESTO (Riou et al., 2023)	self-supervised, very light, real time
Basic Pitch (Spotify, 2022)	polyphonic note transcription

Key points: pitch detection

The problem

Pitch is a period, not a peak

octave errors, missing fundamental

Every representation is a trade-off

time vs frequency, linear vs log

Easy for the ear, hard to compute

noise, reverb, vibrato, several notes at once

Lessons for AI on audio

Knowledge goes into the design

cents, a fine pitch grid, a blurred target

The data decides what is learned

clean labels, varied timbres, honest splits

Always compare with a simple baseline

YIN: no training, 3 000× cheaper

Deep learning does not replace signal processing: **it builds on it**

A few closing remarks

– (Generative) AI for music is now developed by industry

+ Tools are becoming more and more powerful and mature

− We lose knowledge of how these tools work (private research)

– Many problems are still open, such as multipitch transcription!

– Generative AI focuses on European music (simple harmonies, compressed sound, equal temperament, piano-bass-guitar-drums)

→ follow Yaboi Hanoi for music and talks on **Thai AI**

This afternoon: lab and homework

Lab session (3 h)

1. Baseline: spectral peak (STFT / CQT)
2. YIN with librosa
3. Pre-trained CREPE on your own clips
4. Mini-CREPE: implement and train
5. Evaluate with mir_eval
6. f0 → MIDI → synthesizer

Homework: a tool sheet (2 pages max)

An AI / ML tool of your choice:

- Context: who made it, when?
- Distribution: open source, paid?
- What is it for?
- How does it work: data, architecture, training
- Existing “classical” tools
- Your critical opinion

References

- Kim, Salamon, Li, Bello. **CREPE: A Convolutional Representation for Pitch Estimation**. ICASSP 2018. arxiv.org/abs/1802.06182
- Code and models: github.com/marl/crepe · torchcrepe (PyTorch port)
- de Cheveigné, Kawahara. **YIN, a fundamental frequency estimator for speech and music**. JASA 2002
- Mauch, Dixon. **pYIN**. ICASSP 2014 · Camacho, Harris. **SWIPE**. JASA 2008
- Salamon et al. **An analysis/synthesis framework for automatic f0 annotation of multitrack datasets**. ISMIR 2017
- Raffel et al. **mir_eval**. ISMIR 2014
- Tutorial “Open-Source Tools & Data for Music Source Separation”: source-separation.github.io/tutorial
- Engel et al. **Self-supervised Pitch Detection by Inverse Audio Synthesis**. 2020

Thank you for your attention!

Time for discussion

Slides available at https://jeremy-e-cohen.jimdofree.com/