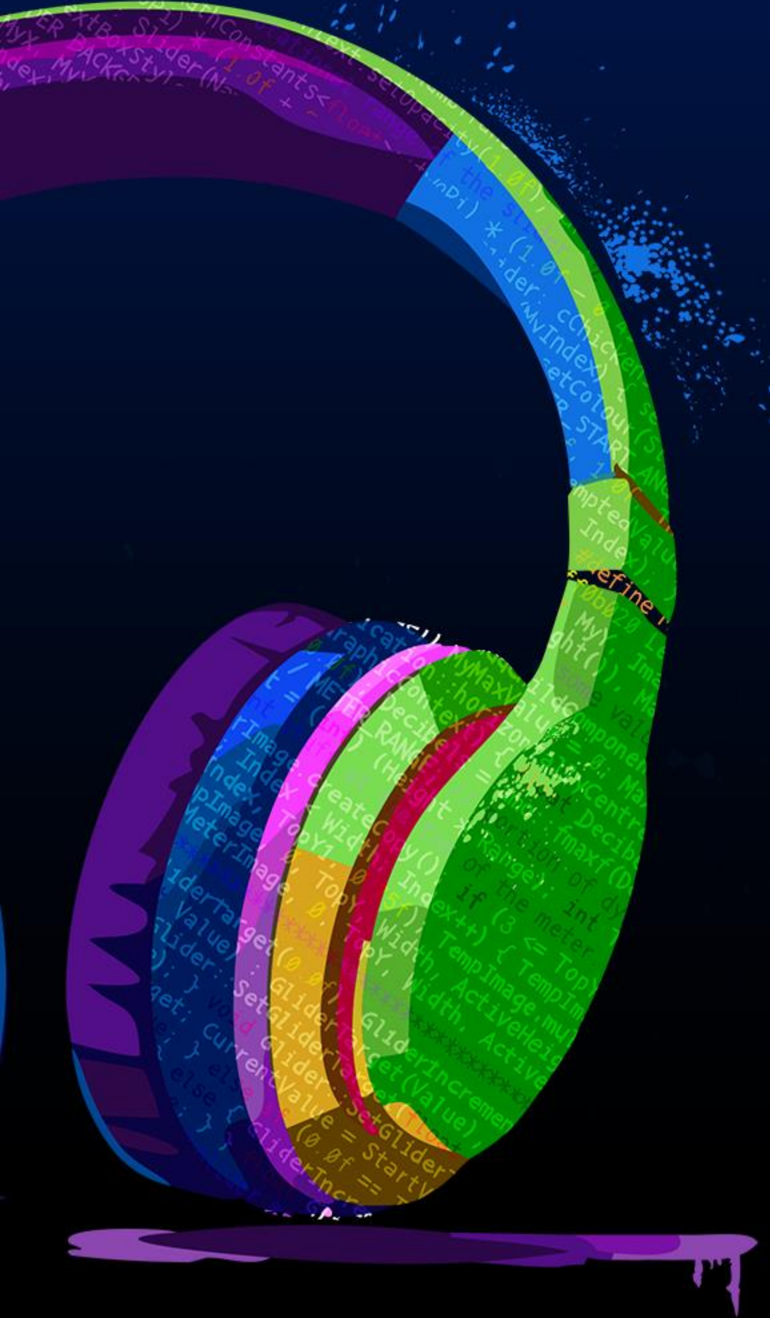


ADC24 Bristol

KNEE-DEEP LEARNING

PRACTICAL STEPS TO GET STARTED WITH AUDIO ML

MARTIN SWANHOLM

Get started…

Practical Steps to Get Started with Machine Learning for Audio

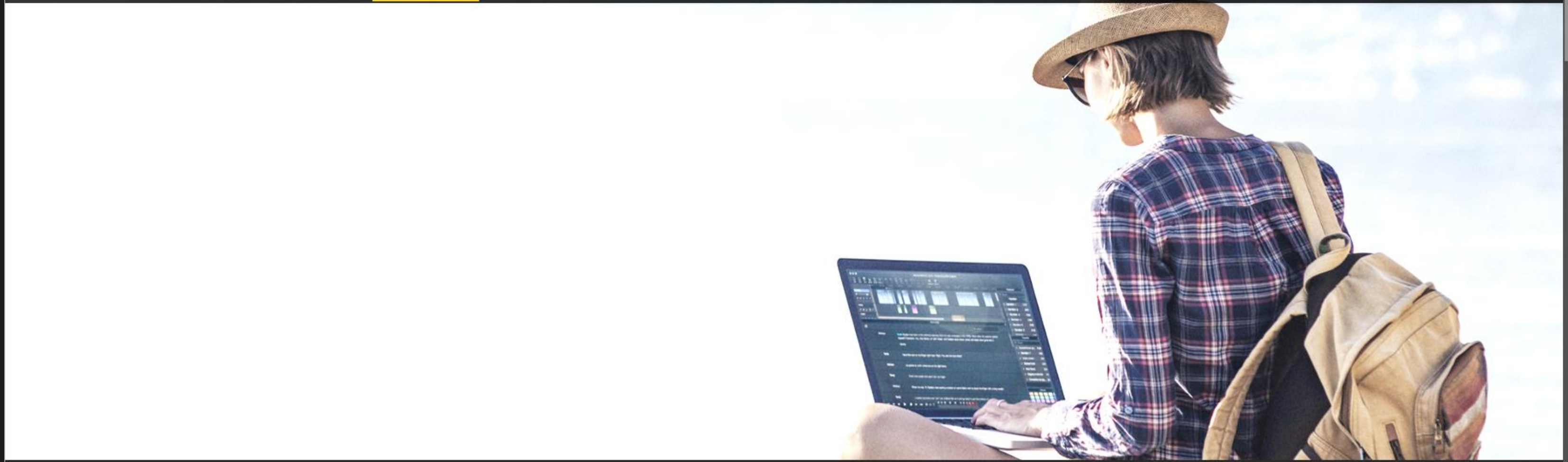
Get started…

Practical Steps to Get Started with Machine Learning for Audio

…and get creative!

Hindenburg

hindenburg.com



You're looking for an audio editor designed specifically for the spoken word?

Welcome to Hindenburg PRO!!

HINDENBURG PRO

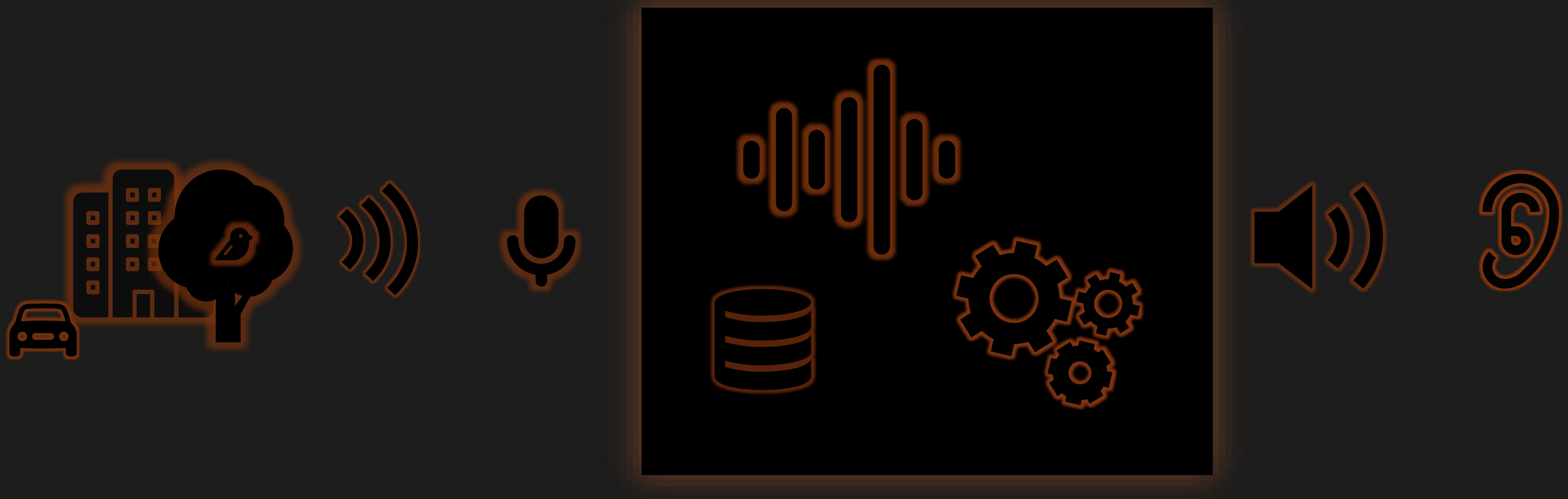
we get you and we've got you

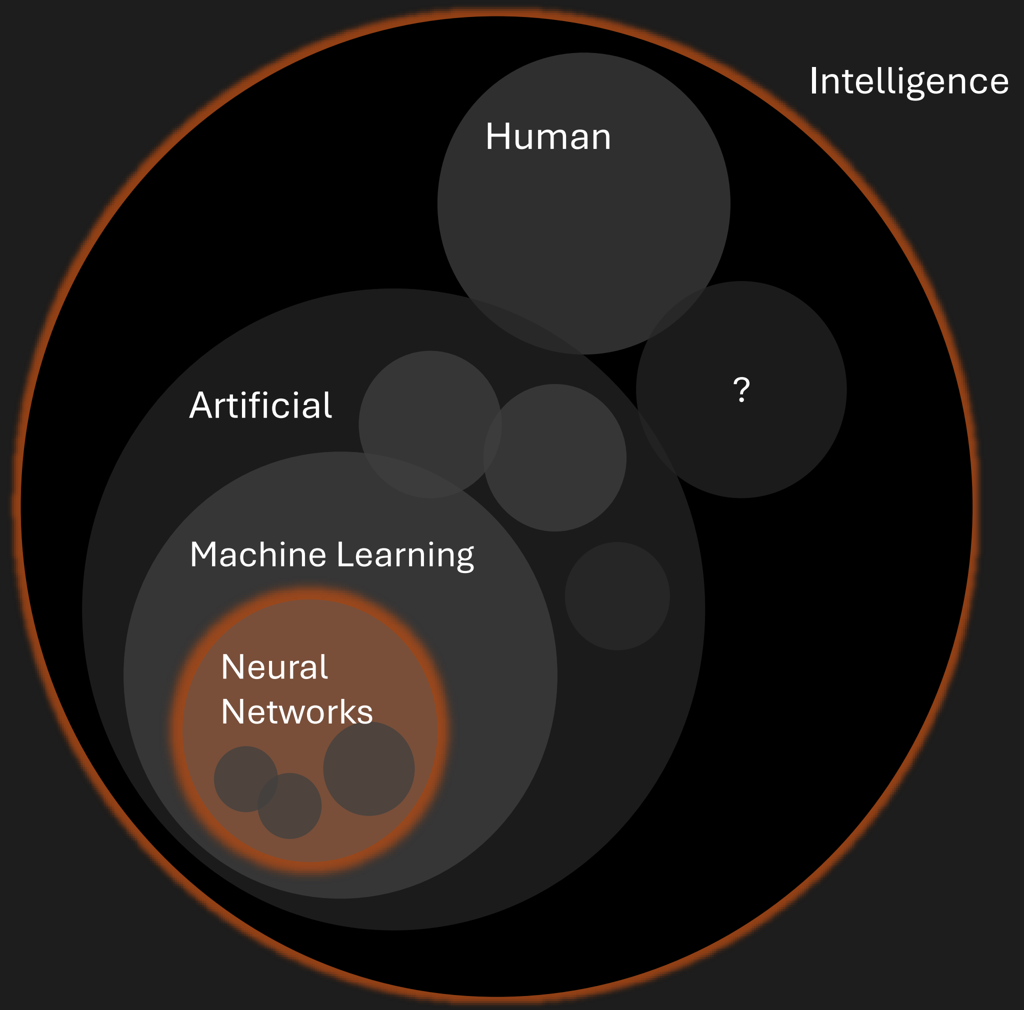
Hindenburg is all about easing your workflow and helping you craft great stories. The editor has every tool you need to record, transcribe, edit, and publish professional audio recordings – and is at the same time so intuitive to use that you won't need deep technical expertise to start producing your first piece.

TRIAL > FEATURES > BUY

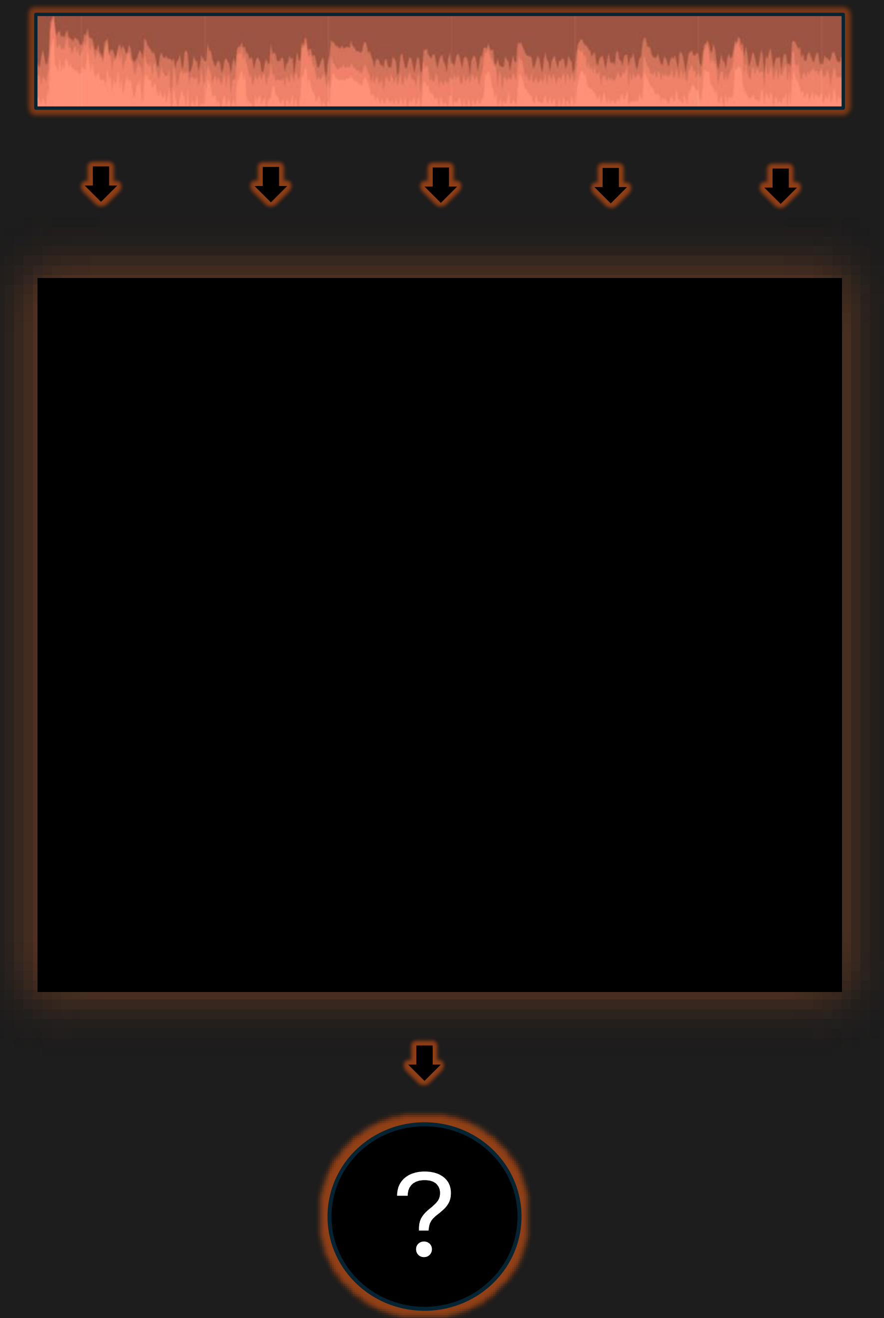
Audio Technology

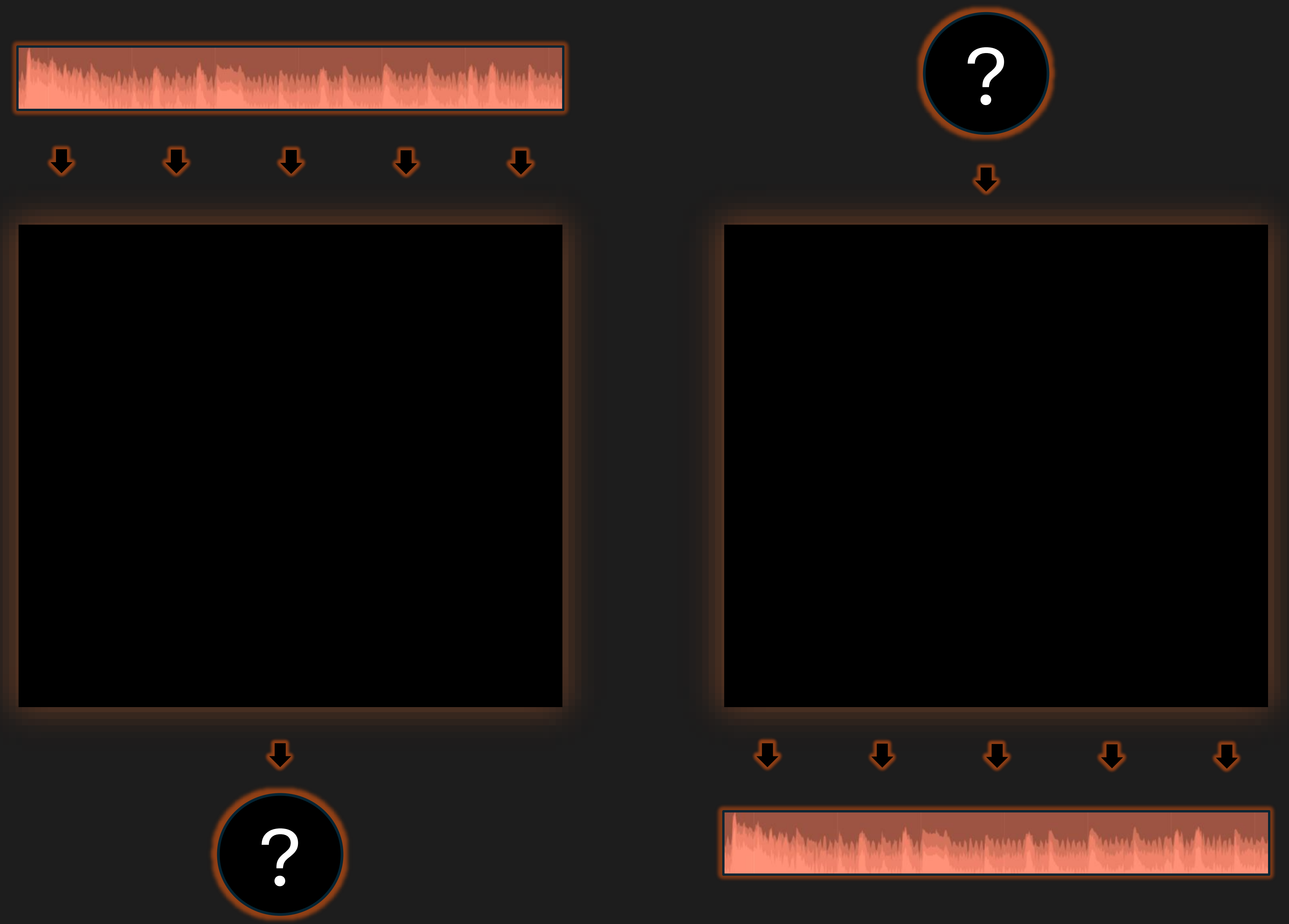
- what does it actually do?
- how does it operate?
- what processes are taking place?
- what are the inputs and outputs?
- how do we interact with systems or tools?
- how can modules be combined to create larger systems?

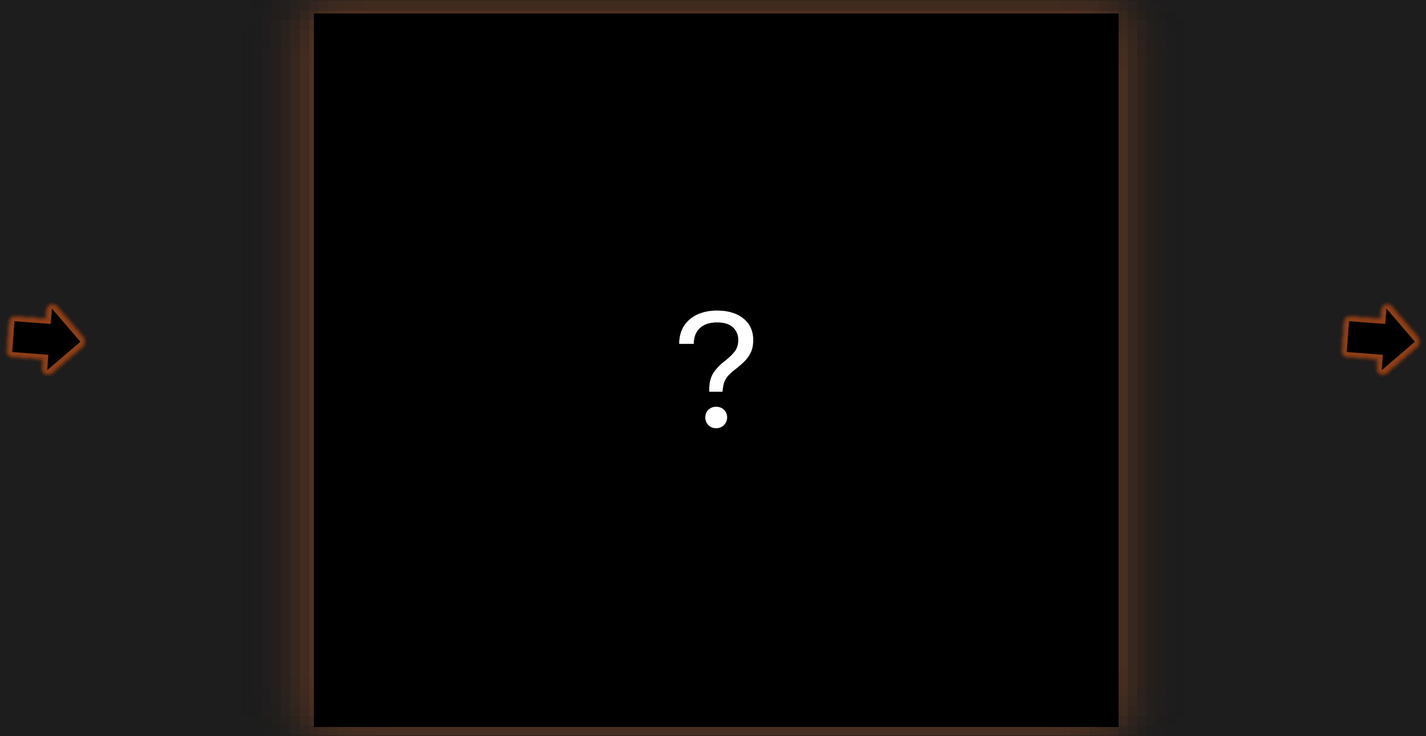



Intelligence
Human
?
Artificial
Machine Learning
Neural
Networks

Neural Network Models for Audio







?

$$y = f(x; \theta)$$

```
float* f(const float* x, size_t size);
```

```
def f(x: array) -> array:
```

Universal approximation theorem

From Wikipedia, the free encyclopedia

In the mathematical theory of artificial neural networks, **universal approximation theorems** are theorems[1][2] of the following form: Given a family of neural networks, for each function f from a certain function space, there exists a sequence of neural networks $\phi_1, \phi_2, \dots$ from the family, such that $\phi_n \to f$ according to some criterion. That is, the family of neural networks is dense in the function space.

The most popular version states that feedforward networks with non-polynomial activation functions are dense in the space of continuous functions between two Euclidean spaces, with respect to the compact convergence topology.

Universal approximation theorems are existence theorems: They simply state that there *exists* such a sequence $\phi_1, \phi_2, \dots \to f$, and do not provide any way to actually find such a sequence. They also do not guarantee any method, such as backpropagation, might actually find such a sequence. Any method for searching the space of neural networks, including backpropagation, might find a converging sequence, or not (i.e. the backpropagation might get stuck in a local optimum).

Universal approximation theorems are limit theorems: They simply state that for any f and a criteria of closeness $\epsilon > 0$, if there are *enough* neurons in a neural network, then there exists a neural network with that many neurons that does approximate f to within ϵ. There is no guarantee that any finite size, say, 10000 neurons, is enough.

Setup [edit]

Artificial neural networks are combinations of multiple simple mathematical functions that implement more complicated functions from (typically) real-valued vectors to real-valued vectors. The spaces of multivariate functions that can be implemented by a network are determined by the structure of the network, the set of simple functions, and its multiplicative parameters. A great deal of theoretical work has gone into characterizing these function spaces.

Most universal approximation theorems are in one of two classes. The first quantifies the approximation capabilities of neural networks with an arbitrary number of artificial neurons ("*arbitrary width*" case) and the second focuses on the case with an arbitrary number of hidden layers, each containing a limited number of artificial neurons ("*arbitrary depth*" case). In addition to these two classes, there are also universal approximation theorems for neural networks with bounded number of hidden layers and a limited number of neurons in each layer ("*bounded depth and bounded width*" case).

History [edit]

We show that standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators with respect to Lp(μ) performance criteria, for arbitrary finite input environment measures μ, provided only that sufficiently many hidden units are available. If the activation function is continuous, bounded and nonconstant, then continuous mappings can be learned uniformly over compact input sets. We also give very general conditions ensuring that networks with sufficiently smooth activation functions are capable of arbitrarily accurate approximation to a function and its derivatives.

Kurt Hornik, “Approximation capabilities of multilayer feedforward networks”, Neural Networks, Volume 4, Issue 2, 1991, Pages 251-257

"It is shown that standard multilayer feedforward networks with arbitrary squashing functions are capable of approximating any measurable function to arbitrary accuracy."

Universal approximation theorem

From Wikipedia, the free encyclopedia

In the mathematical theory of artificial neural networks, **universal approximation theorems** are theorems[1][2] of the following form: Given a family of neural networks, for each function f from a certain function space, there exists a sequence of neural networks $\phi_1, \phi_2, \ldots$ from the family, such that $\phi_n \to f$ according to some criterion. [illegible]

The most popular version states that feedforward networks with non-polynomial activation functions are dense in the space of continuous functions between two Euclidean spaces, with respect to the compact convergence topology.

Universal approximation [illegible] $\phi_2, \cdots \to f$, and do not provide any way to actually find such a sequence. They also do not guarantee any method, such as backpropagation, might actually find such a sequence. Any method for [illegible] a converging sequence, or not (i.e. the backpropagation might get stuck in a local optimum).

Universal approximation theorems are limit theorems: They simply state that for any f and a criteria of closeness $\epsilon > 0$, if there are *enough* neurons in a neural network, then there exists a neural network with that many neurons that does approximate f to within ϵ. There is no guarantee that any finite size, say, 10000 neurons, is enough.

Setup

Artificial neural networks are combinations of multiple simple mathematical functions that implement more complicated functions from (typically) real-valued vectors to real-valued vectors. The spaces of multivariate functions that can be implemented by a network are determined by the structure of the network, the set of simple functions, and its multiplicative parameters. A great deal of theoretical work has gone into characterizing these function spaces.

Most universal approximation theorems are in one of two classes. The first quantifies the approximation capabilities of neural networks with an arbitrary number of artificial neurons ("*arbitrary width*" case) and the second focuses on the case with an arbitrary number of hidden layers, each containing a limited number of artificial neurons ("*arbitrary depth*" case). In addition to these two classes, there are also universal approximation theorems for neural networks with bounded number of hidden layers and a limited number of neurons in each layer ("*bounded depth and bounded width*" case).

History

Universal approximation theorem

"A neural network with at least one hidden layer can approximate any continuous function to any desired accuracy, given enough neurons and proper activation functions."

From Wikipedia, the free encyclopedia

Setup

Artificial neural networks are combinations of multiple simple mathematical functions that implement more complicated functions from (typically) real-valued vectors to real-valued vectors. The spaces of multivariate functions that can be implemented by a network are determined by the structure of the network, the set of simple functions, and its multiplicative parameters. A great deal of theoretical work has gone into characterizing these function spaces.

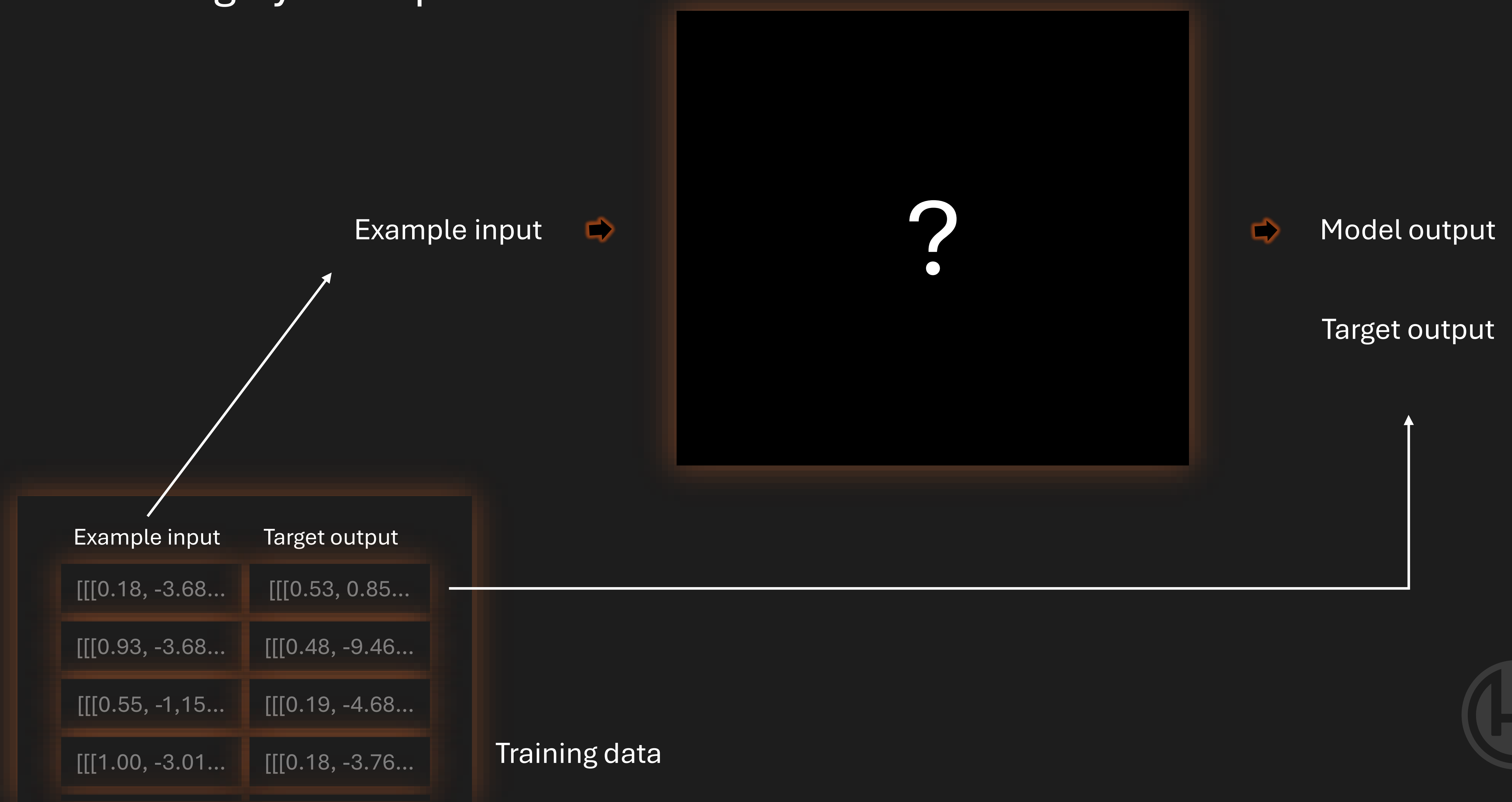
Most universal approximation theorems are in one of two classes. The first quantifies the approximation capabilities of neural networks with an arbitrary number of artificial neurons ("*arbitrary width*" case) and the second focuses on the case with an arbitrary number of hidden layers, each containing a limited number of artificial neurons ("*arbitrary depth*" case). In addition to these two classes, there are also universal approximation theorems for neural networks with bounded number of hidden layers and a limited number of neurons in each layer ("*bounded depth and bounded width*" case).

History

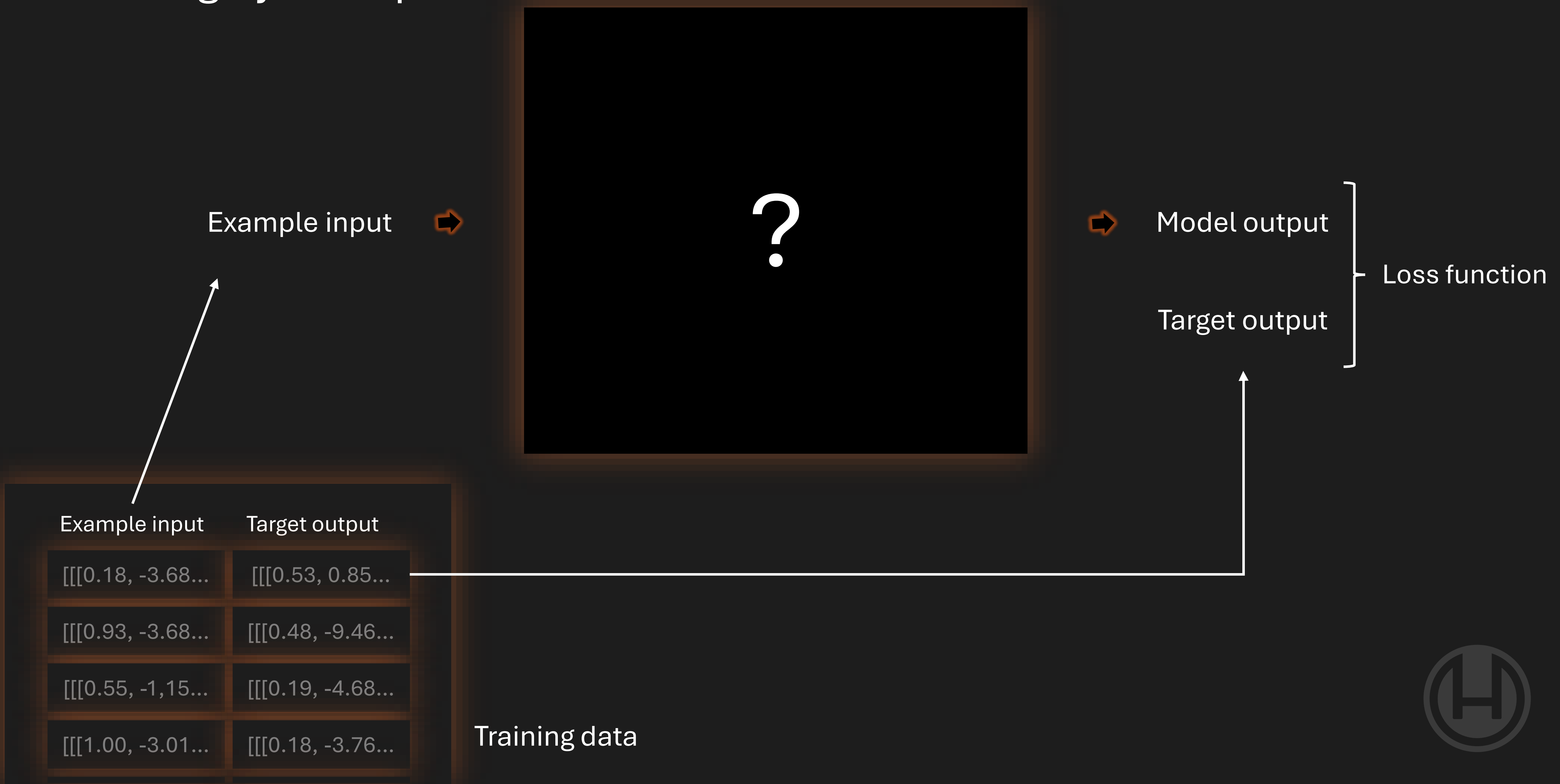
Learning by examples

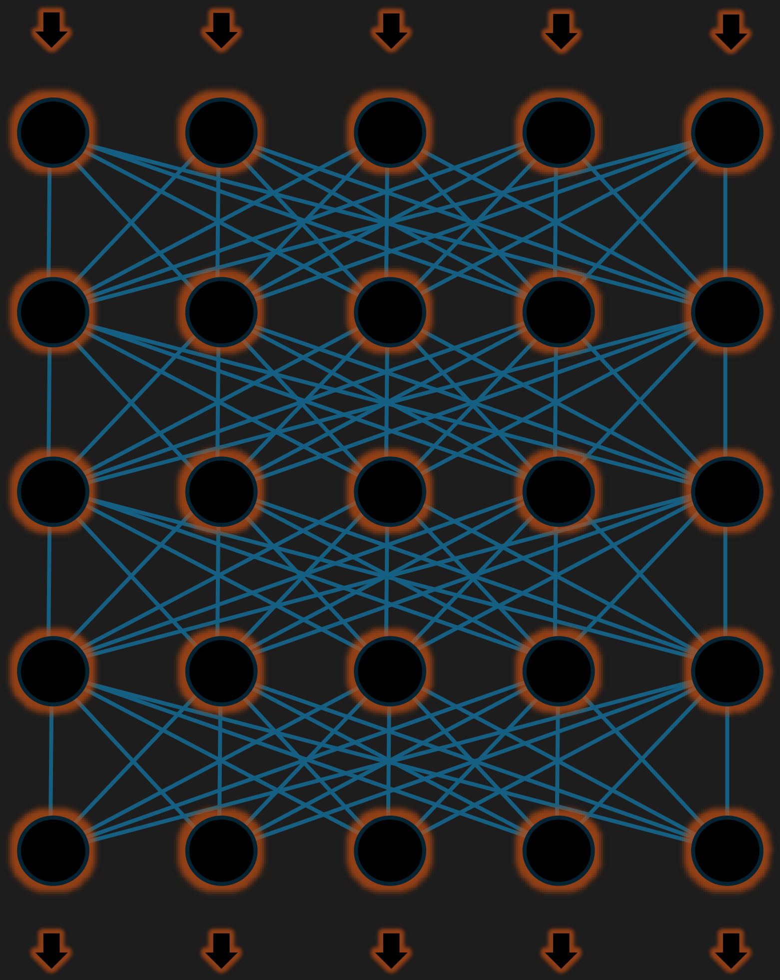
Example input → ? → Model output

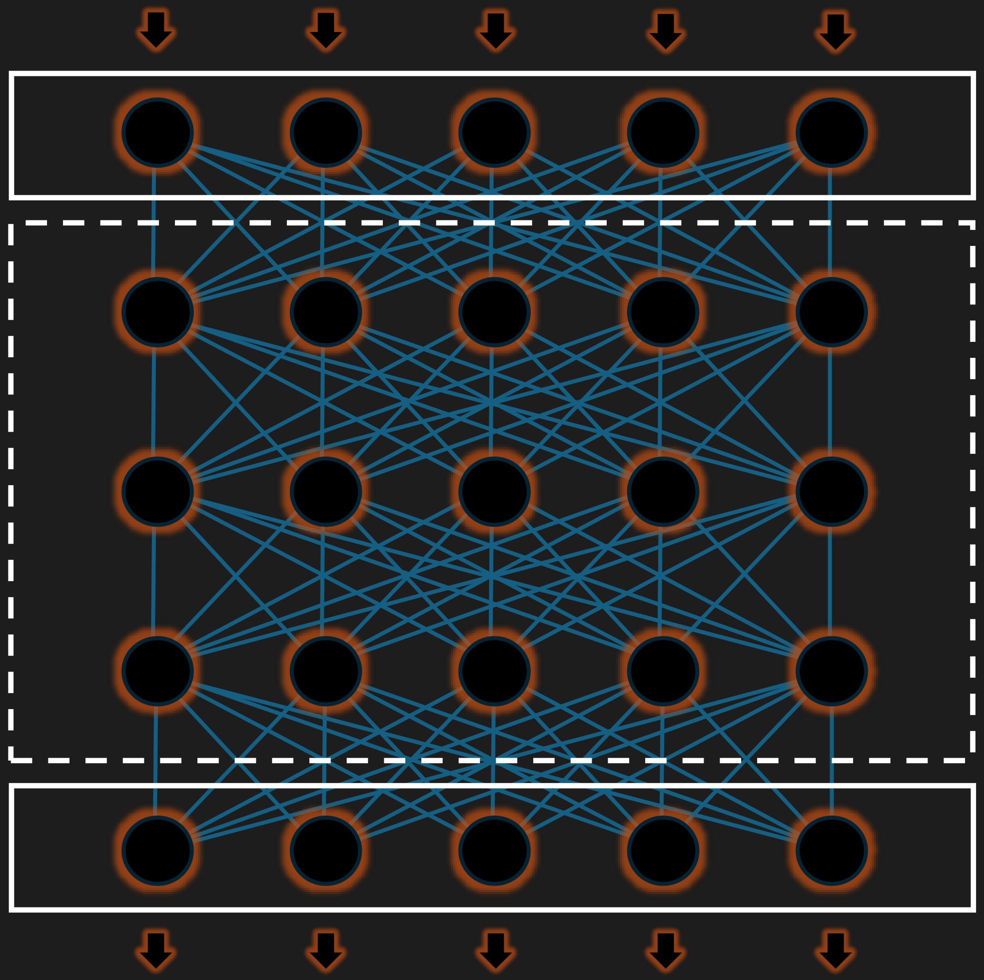
Learning by examples



Learning by examples



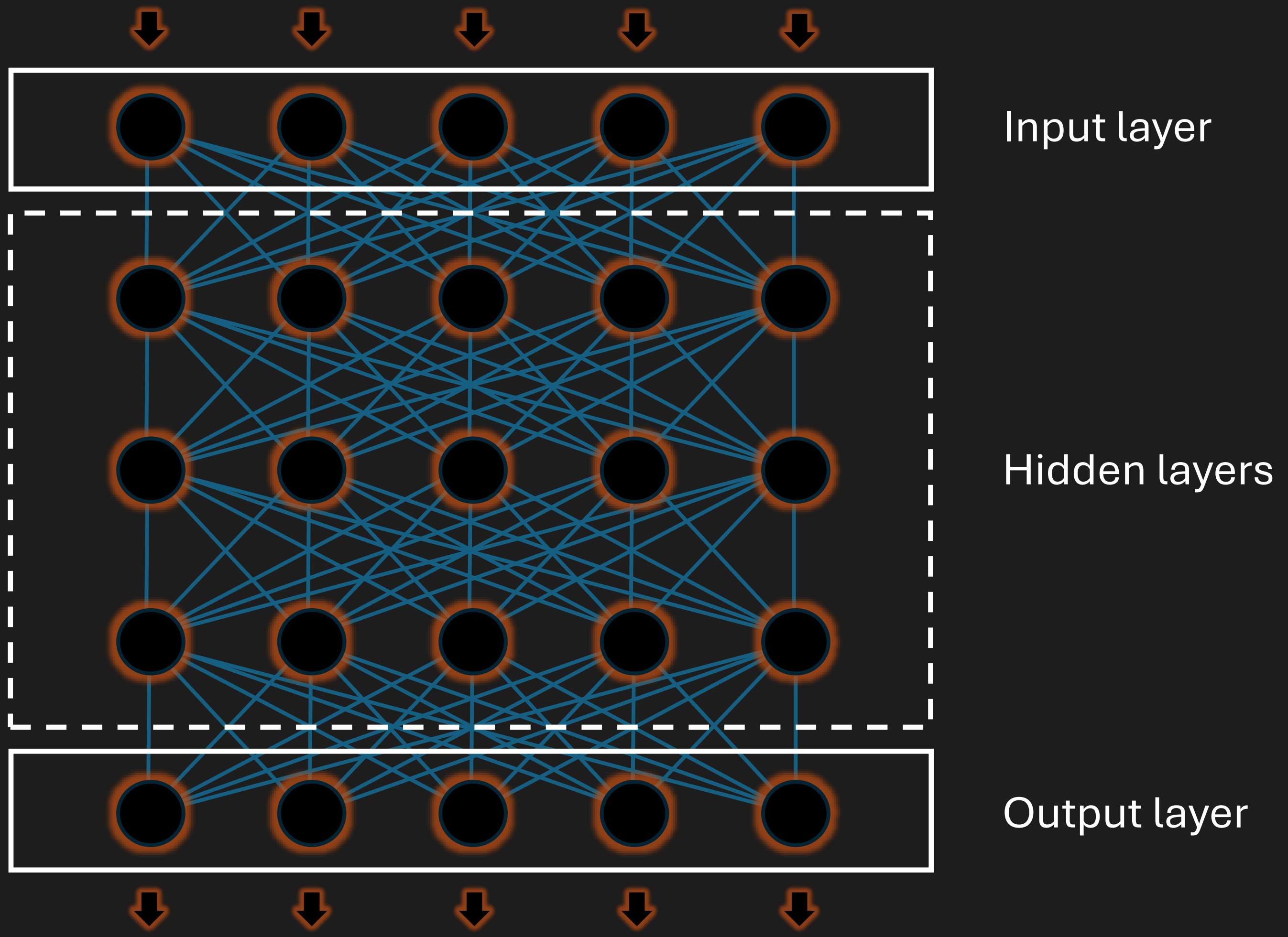


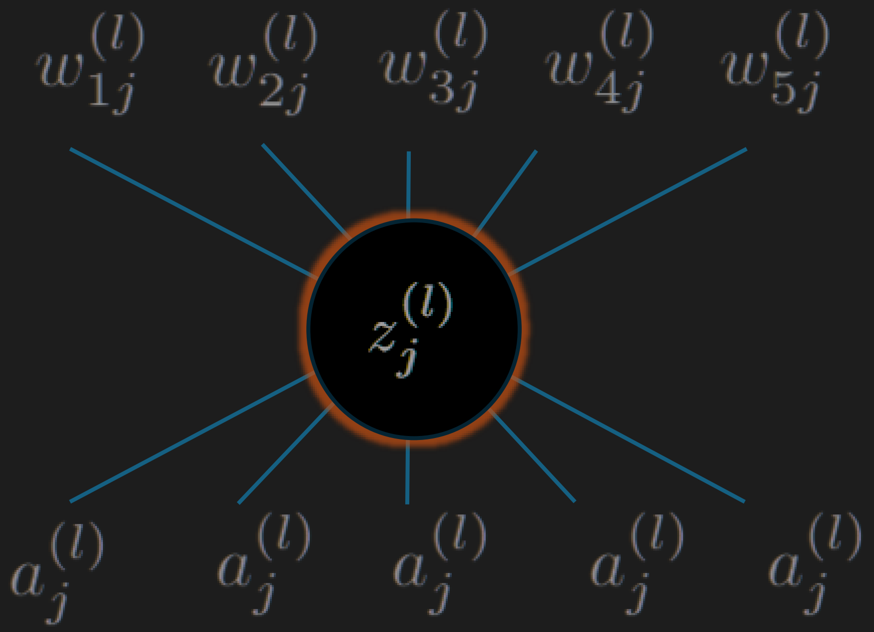


Input layer

Hidden layers

Output layer





$$z_j^{(l)} = \sum_i w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

$$a_j^{(l)} = \text{activation}(z_j^{(l)})$$

Loss functions

- Depends on the task and the type and shape of the data
- Determines what we want to optimize for
- Controls the optimization process

- Classification
    - Cross-Entropy loss
    - Activation: Softmax (+ argmax)

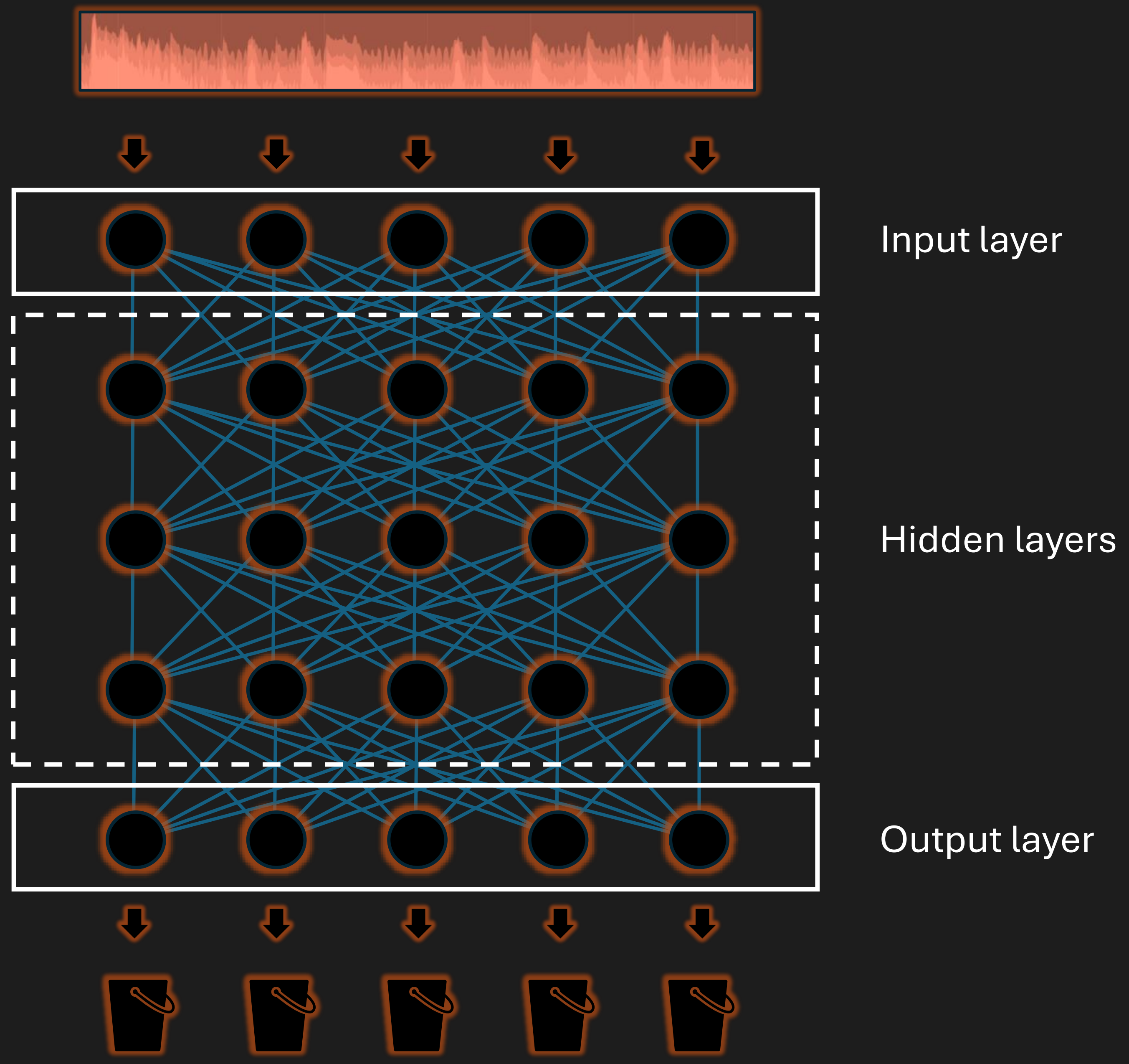
- Regression
    - L1, L2 loss

Activation function

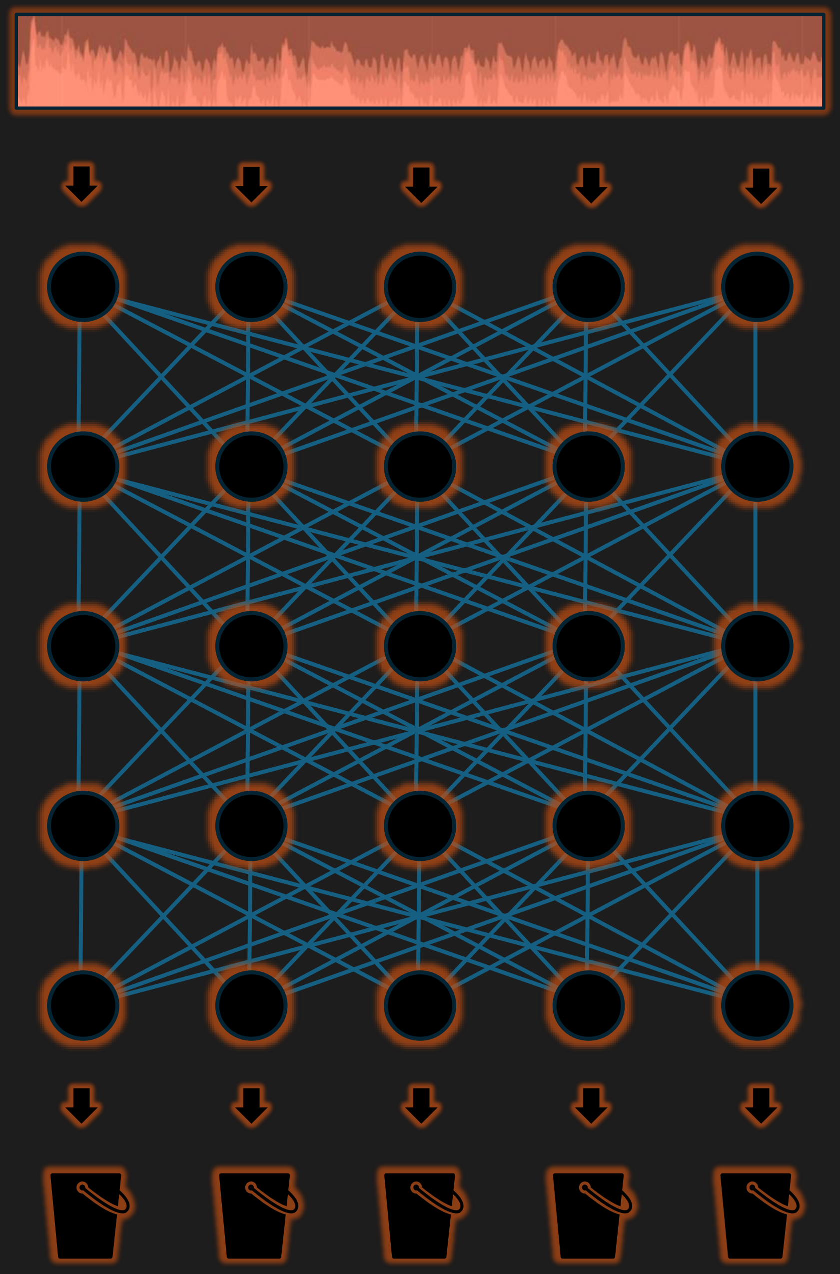
- ReLU
- Sigmoid

Optimization

- SGD
- Adam



Input layer
Hidden layers
Output layer



Learning by examples

Example input

?

Model output

Target output

Loss function

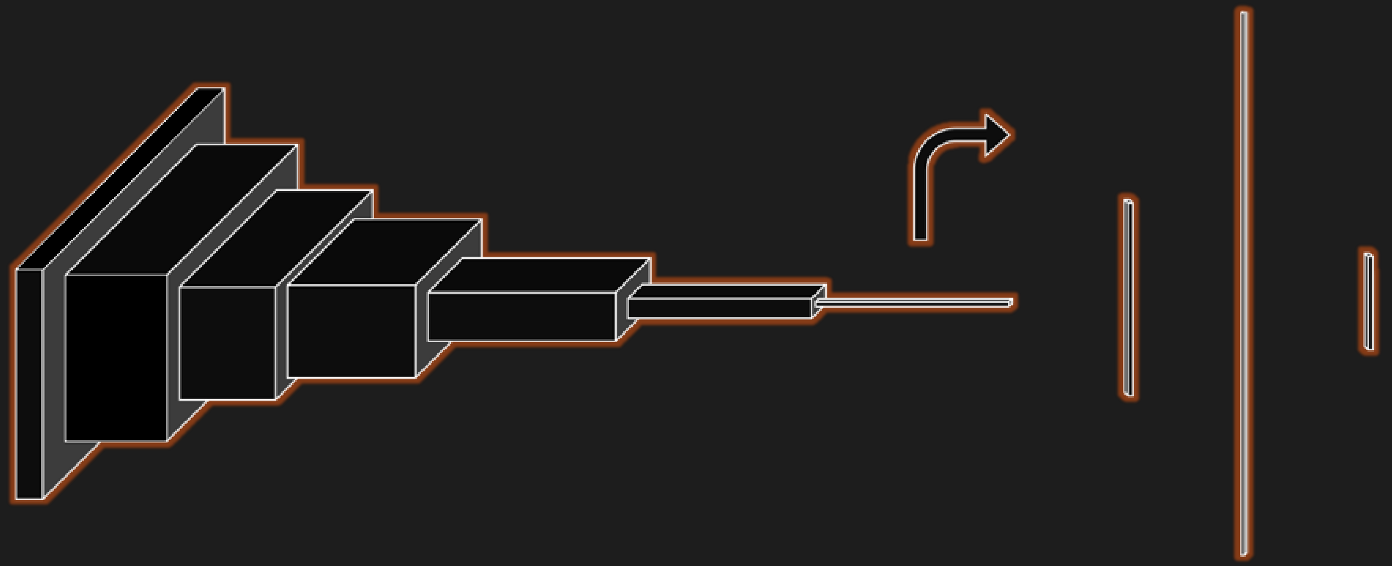
Example input	Target output
[[[0.18, -3.68...	[[[0.53, 0.85...
[[[0.93, -3.68...	[[[0.48, -9.46...
[[[0.55, -1,15...	[[[0.19, -4.68...
[[[1.00, -3.01...	[[[0.18, -3.76...

Training data

$$y = f(x; \theta)$$

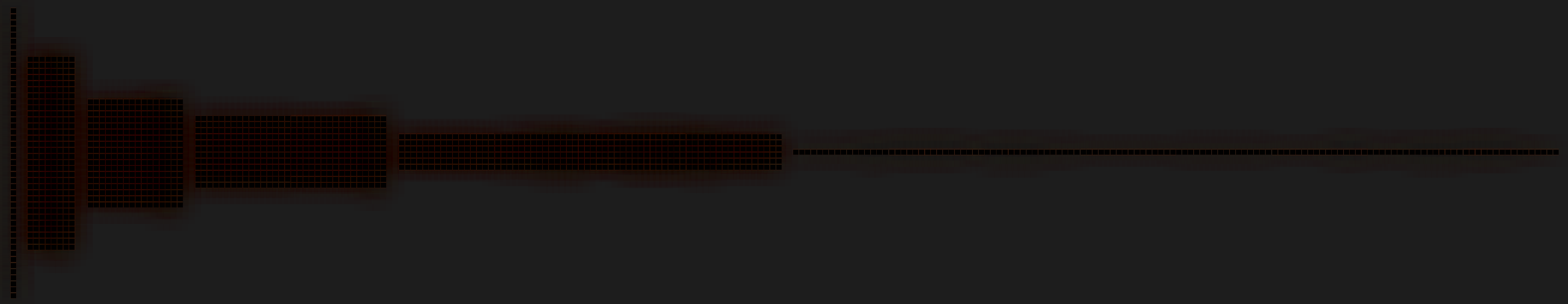
```
float* f(const float* x, size_t size);
```

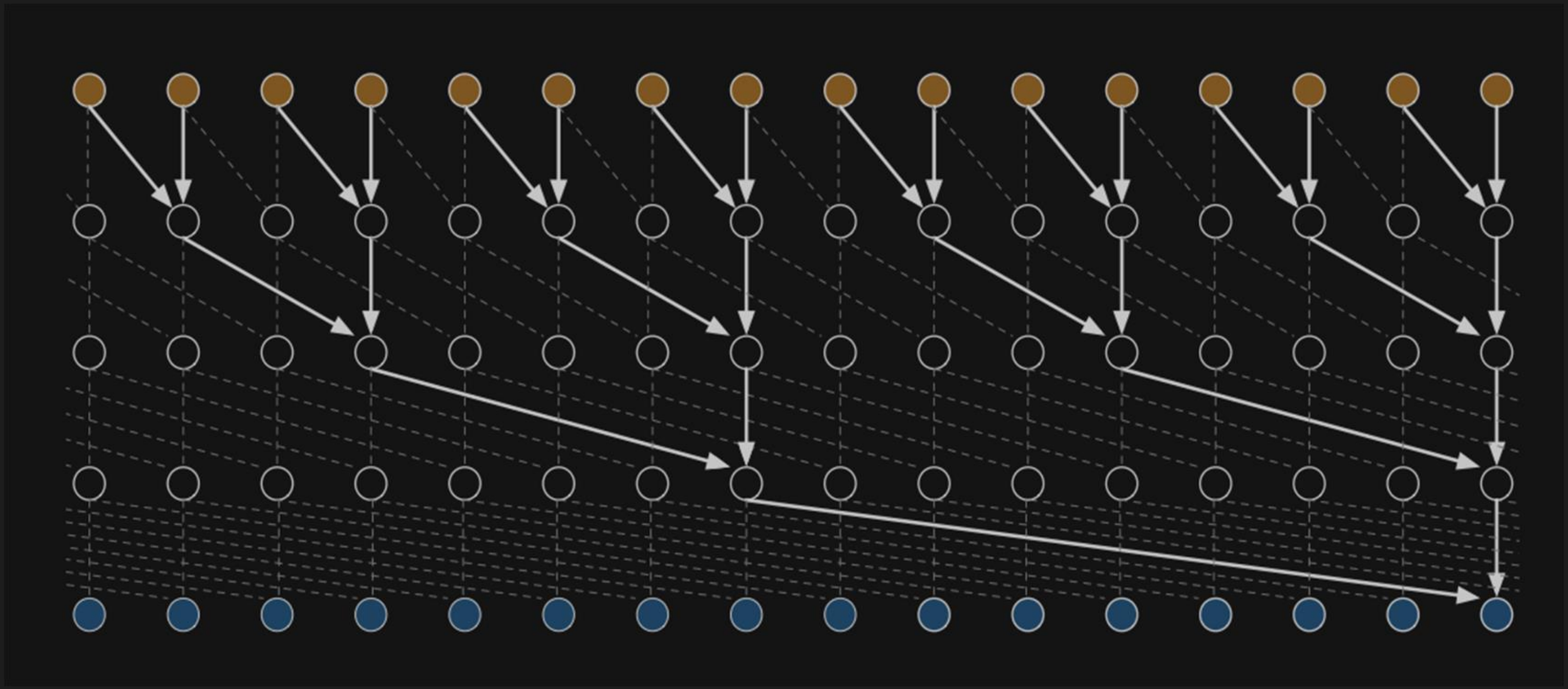
```
def f(x: array) -> array:
```



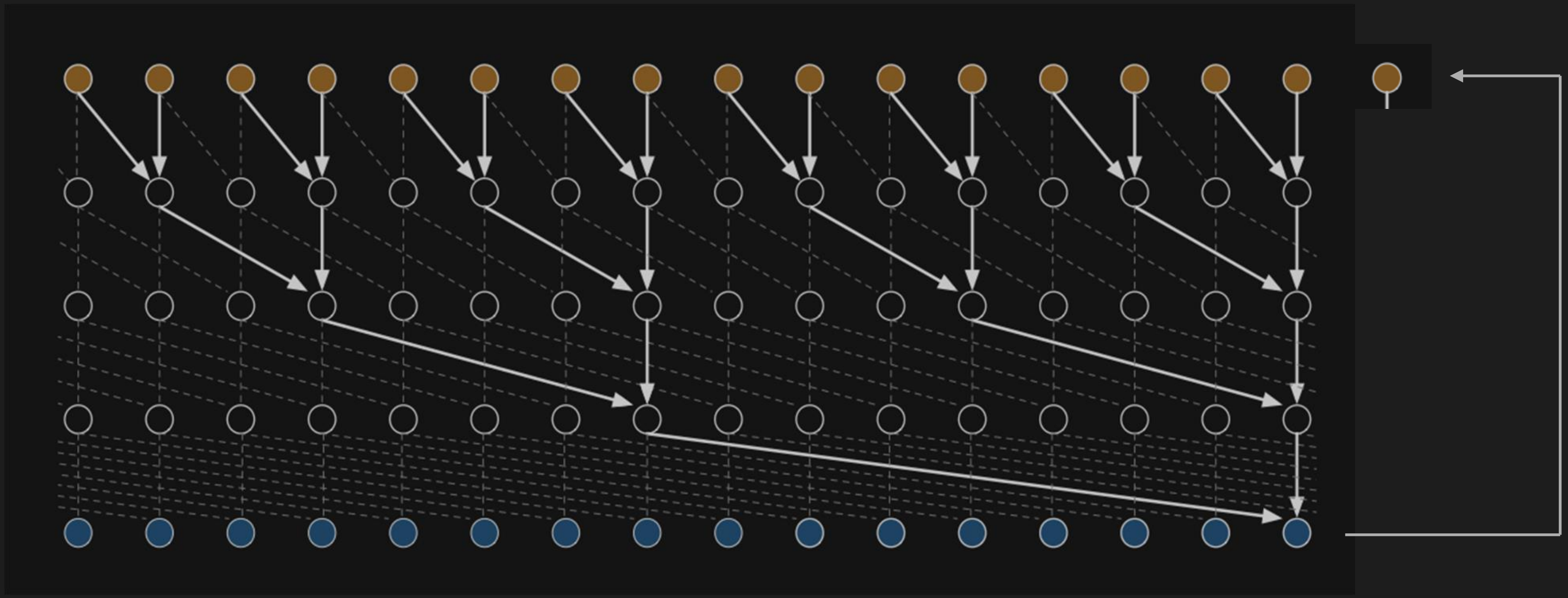
Pablo Ruiz, “Understanding and visualizing ResNets”, https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

Convolution



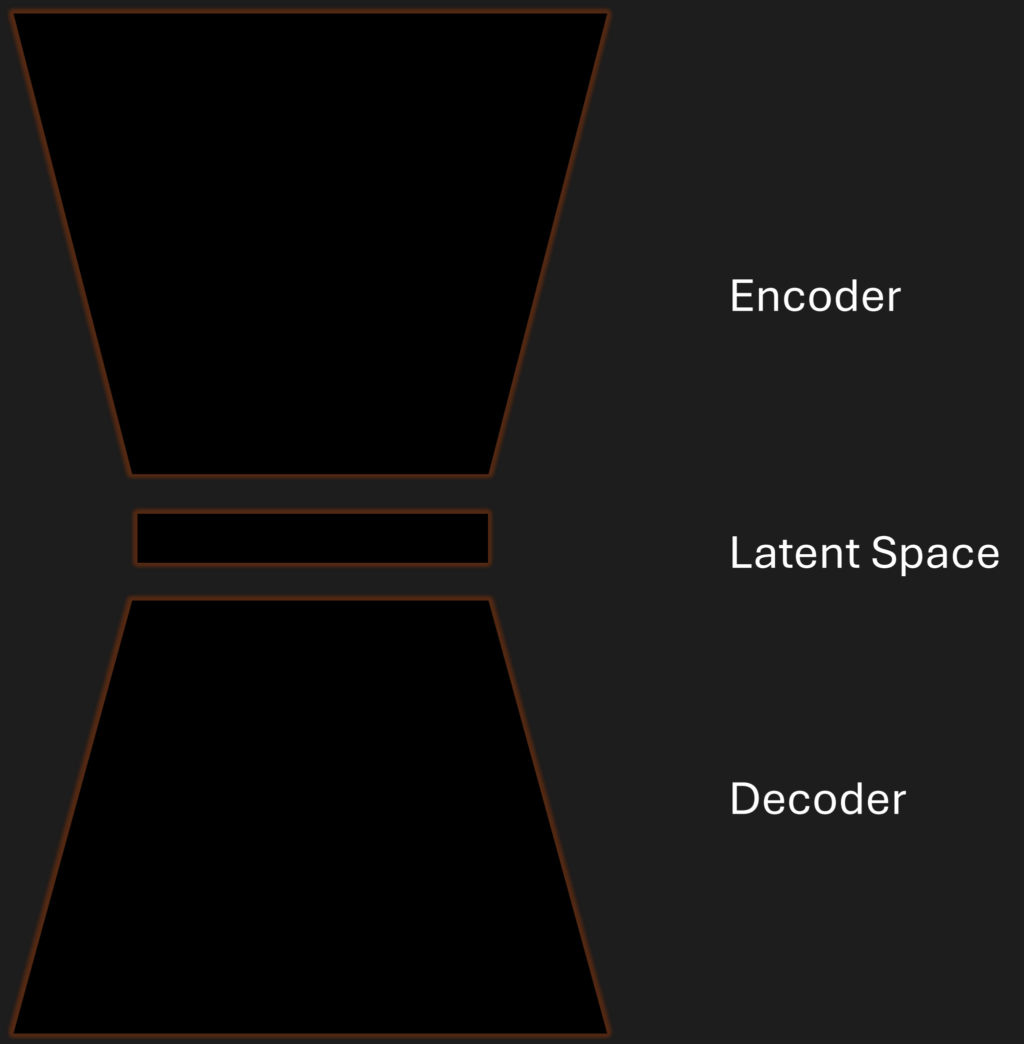


Aaron van den Oord, et al. “WaveNet: A Generative Model for Raw Audio”, https://arxiv.org/abs/1609.03499

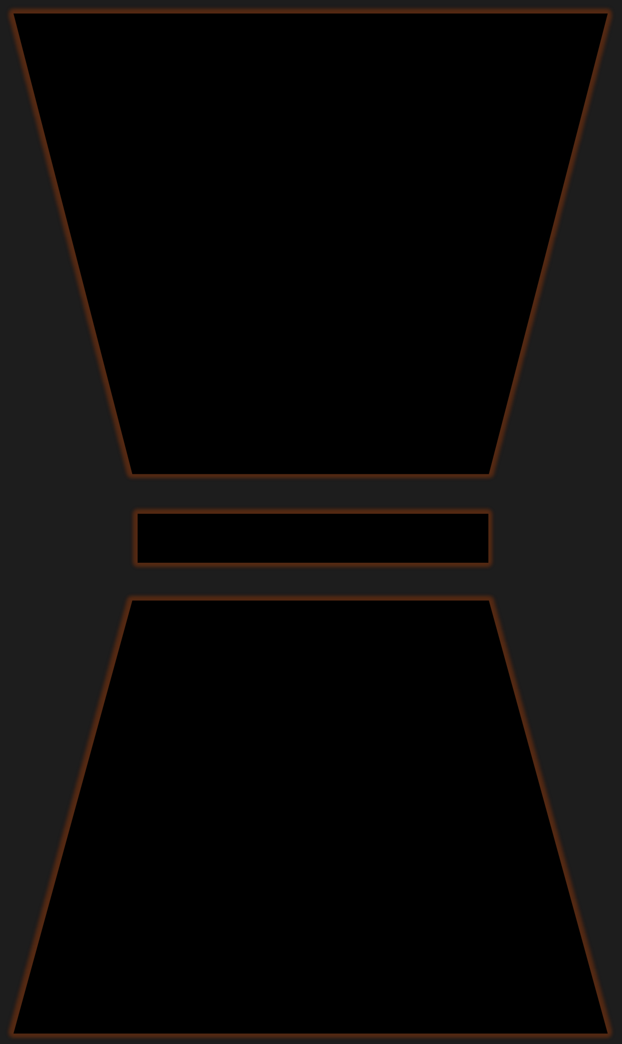


Aaron van den Oord, et al. “WaveNet: A Generative Model for Raw Audio”, https://arxiv.org/abs/1609.03499

Auto-Encoder



Variational Auto-Encoder (VAE)



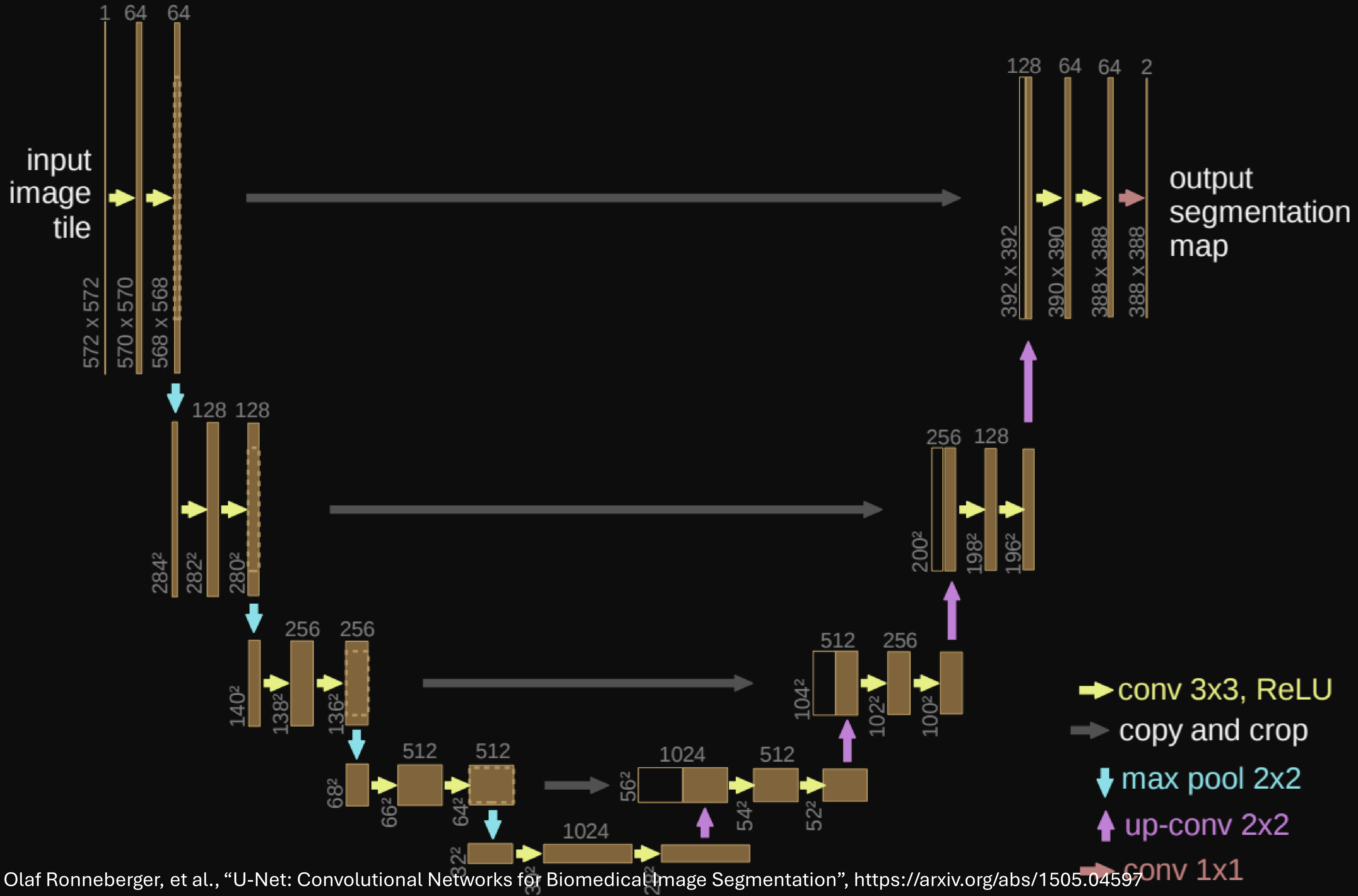
Encoder

Latent Space

Decoder

Generative Adversarial Network (GAN)

U-net



Olaf Ronneberger, et al., "U-Net: Convolutional Networks for Biomedical Image Segmentation", https://arxiv.org/abs/1505.04597

Frequency-Domain models

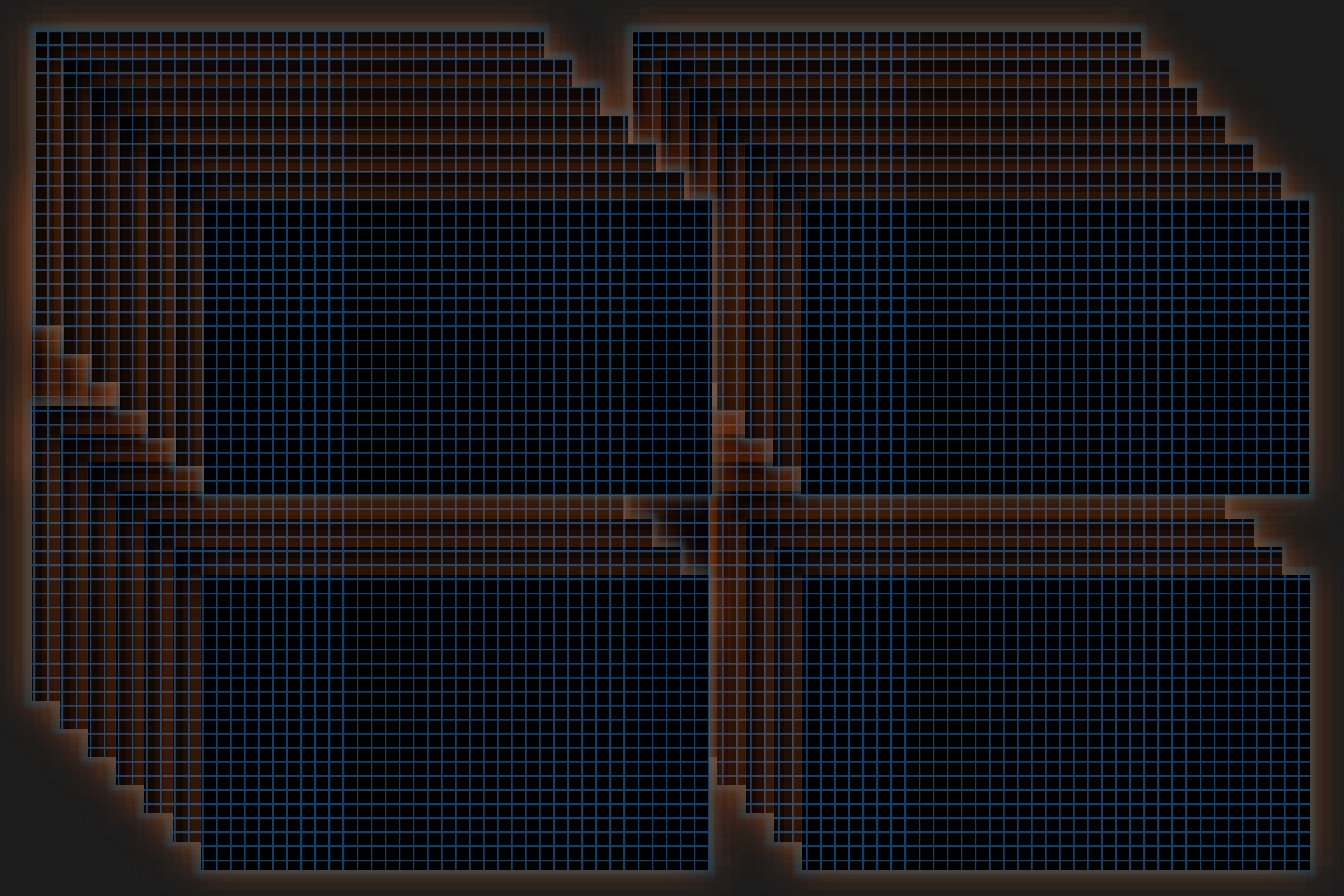
- Spectrograms or raw transform-domain data (stft)
- Other transforms

- 2D-CNNs
- Phase-coherence

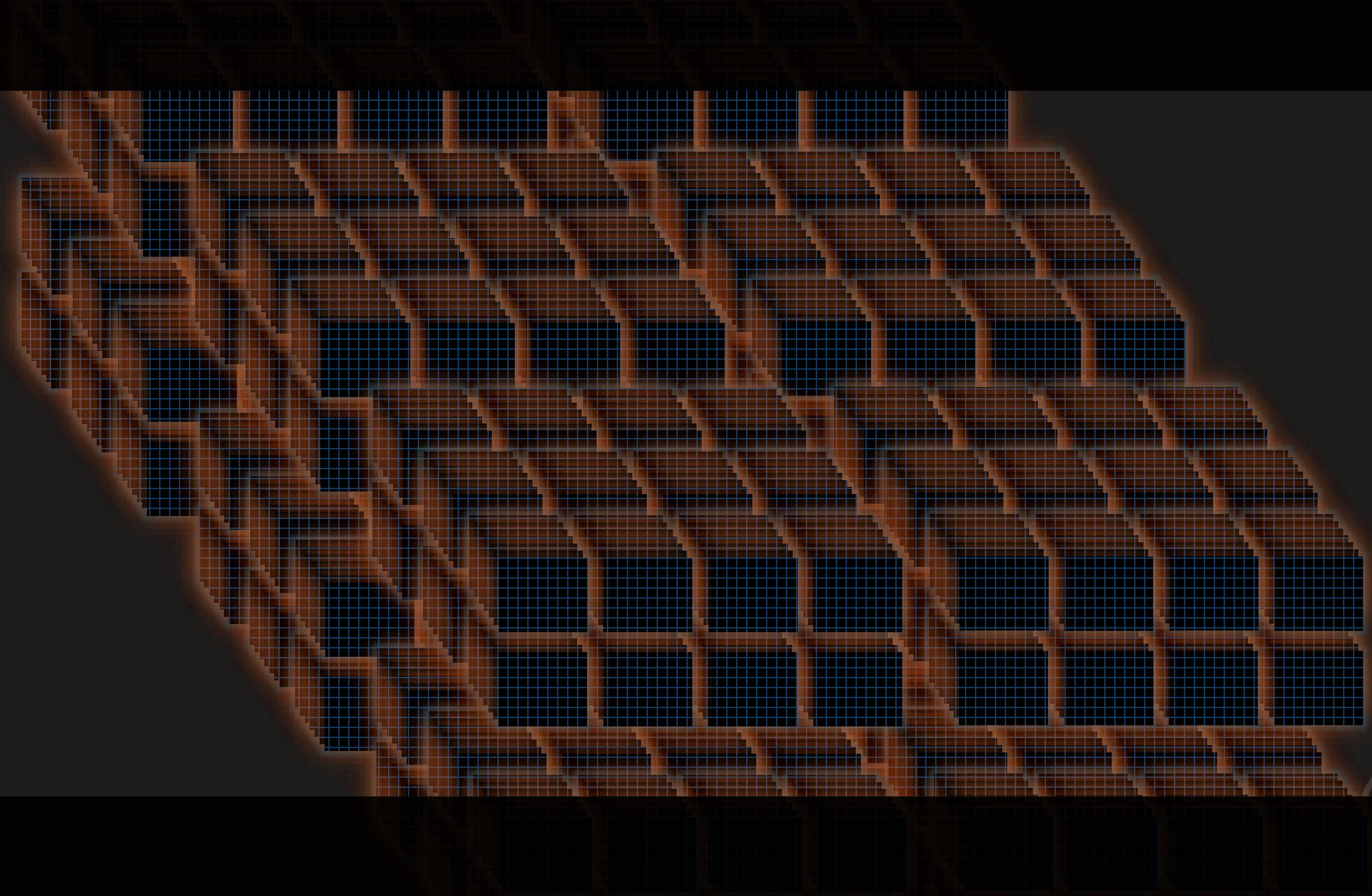
Other Types of Networks

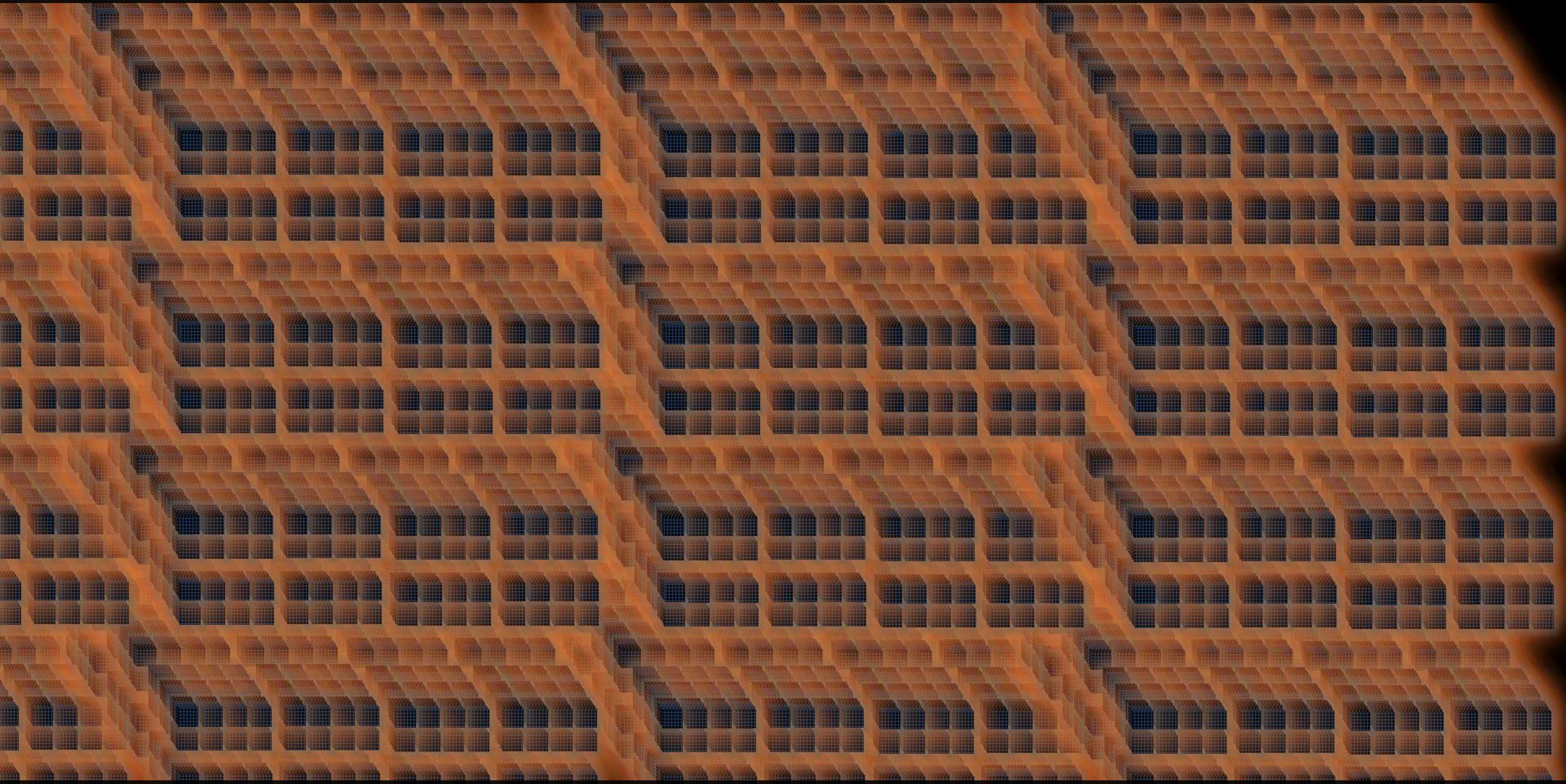
- Transformers
- Stable Diffusion

Create!



“All beginnings cement detrimental esoteric facts genuinely highlighting ingenuity — jokingly, knowledgeable, leveraging meaning”





$$$

- Azure
- AWS
- Google Cloud

$

- Lambda Labs
- Paperspace

? $

Local computer

- Gets you started quickly
- Supports all workflows
- Ease of use
- Good for light to moderate training
- Not efficient for large-scale training

Data

Data acquisition

- Make your own!
- Research datasets
- Use other data in the public domain
- Synthetic data
  - Synthesis
  - Acoustic simulations
  - Sequencing/layering of sounds
  - Output from other models

- Buy it / License it $$$

- Use any material for own research and learning

- Be creative!

Data acquisition

- Make your own!
- Research datasets
- Use other data in the public domain
- Synthetic data
    - Synthesis
    - Acoustic simulations
    - Sequencing/layering of sounds
    - Output from other models

- Buy it / License it $$$

- Use any material for own research and learning

- Be creative!

Observe rules, regulations and licensing terms!

Data acquisition

- Make your own!
- Research datasets
- Use other data in the public domain
- Synthetic data
    - Synthesis
    - Acoustic simulations
    - Sequencing/layering of sounds
    - Output from other models

- Buy it / License it $$$

- Use any material for own research and learning

- Be creative!

Observe rules, regulations and licensing terms!

Be mindful of ethical considerations!

Technology stack

Hardware

CPU

CUDA / Nvidia

ROCm / AMD

MPS / Apple

ARM ai

Low-level

BLAS/cuBLAS | ATen | Eigen | cuDNN

Hardware

CPU | CUDA / Nvidia | ROCm / AMD | MPS / Apple | ARM ai

PyTorch (Python framework and API)

Other languages

LibTorch (C++ API)

Low-level

BLAS/cuBLAS

ATen

Eigen

cuDNN

Hardware

CPU

CUDA / Nvidia

ROCm / AMD

MPS / Apple

ARM ai

Additional frameworks on top of PyTorch

PyTorch (Python framework and API)

Other languages

LibTorch (C++ API)

Low-level

BLAS/cuBLAS

ATen

Eigen

cuDNN

Hardware

CPU

CUDA / Nvidia

ROCm / AMD

MPS / Apple

ARM ai

Your application code

Application code

Additional frameworks on top of PyTorch

PyTorch (Python framework and API)

Other languages

LibTorch (C++ API)

Low-level

BLAS/cuBLAS

ATen

Eigen

cuDNN

Hardware

CPU

CUDA / Nvidia

ROCm / AMD

MPS / Apple

ARM ai

```python
class ConvModelSimple(nn.Module):
    def __init__(self, num_classes=4, kernel_sizes=[33, 5, 5, 3], strides=[16, 8, 4, 2]):
        super(ConvModelSimple, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=16, kernel_size=kernel_sizes[0], stride=strides[0], dilation=4)
        self.conv2 = nn.Conv1d(in_channels=16, out_channels=32, kernel_size=kernel_sizes[1], stride=strides[1])
        self.conv3 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=kernel_sizes[2], stride=strides[2], dilation=2)
        self.conv4 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=kernel_sizes[3], stride=strides[3])
        self.pool = nn.AdaptiveAvgPool1d(1)
        self.fc = nn.Linear(128, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

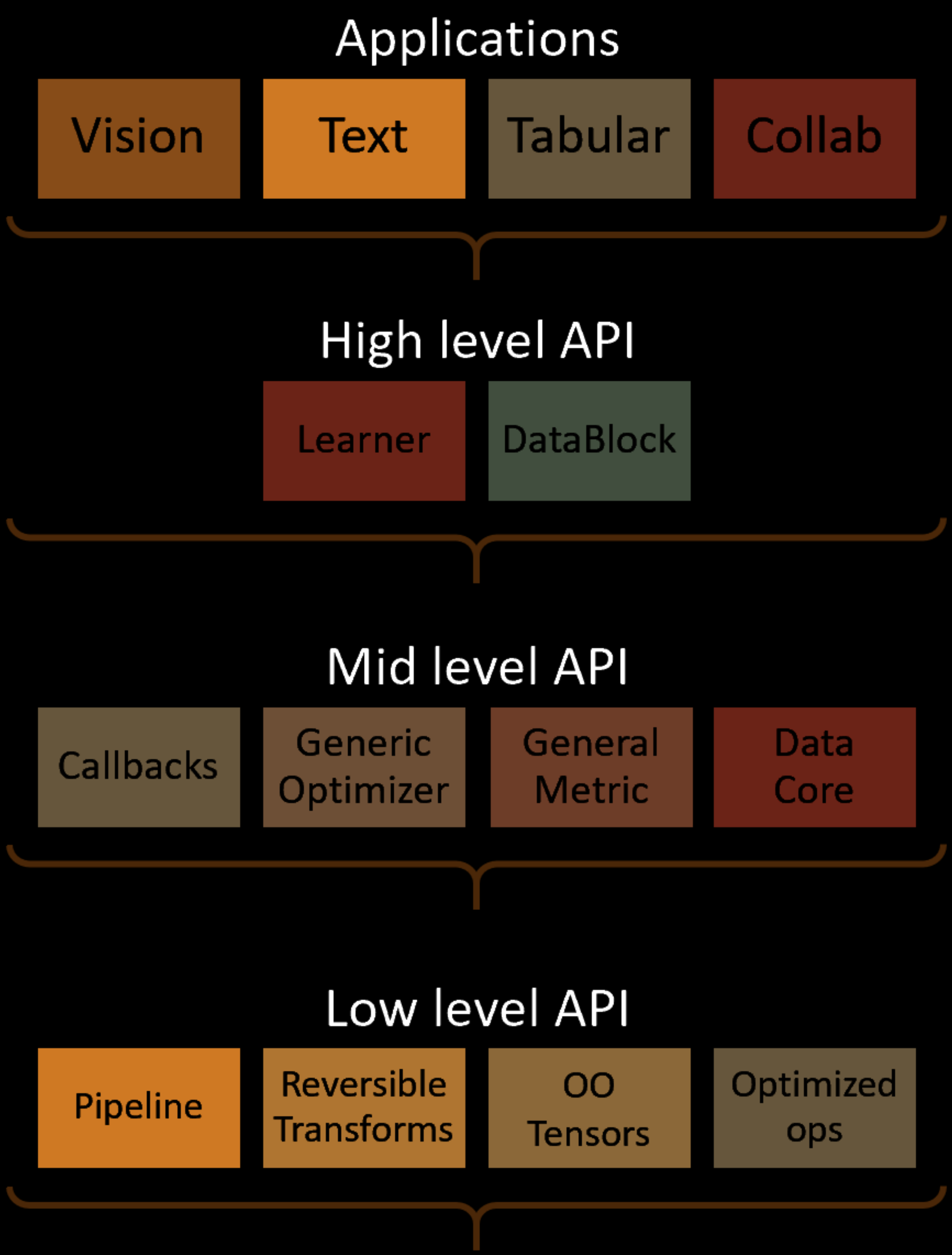
```
for x_batch, target_batch in data_loader:
    optimizer.zero_grad()
    output = model(x_batch)
    loss = torch.nn.functional.cross_entropy(output, target_batch)
    loss.backward()
    optimizer.step()
```

High-Level Frameworks

PyTorch:
- Fastai
- PyTorch Lightning

Other:
- Keras
- scikit-learn

fast.ai



https://fast.ai

Netron

https://netron.app/

Tensorboard

TensorFlow Install Learn API Resources Community Why TensorFlow Search

TensorBoard

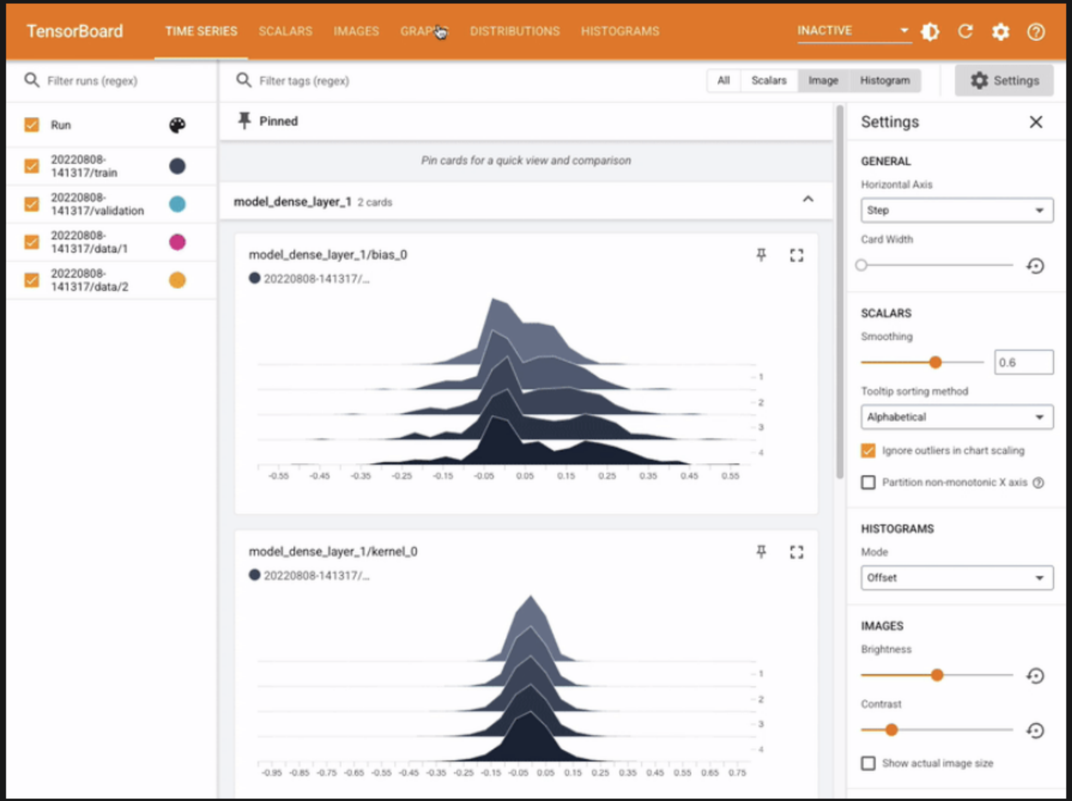
Overview Guide

TensorBoard: TensorFlow's visualization toolkit

TensorBoard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs
- And much more

Get started



Learning and Inspiration

RAVE: A variational autoencoder for fast and high-quality neural audio synthesis

Antoine Caillon & Philippe Esling
IRCAM - Sorbonne Université
CNRS UMR 9912
1, place Igor Stravinsky, Paris, France
{caillon,esling}@ircam.fr

Abstract

Deep generative models applied to audio have improved by a large margin the state-of-the-art in many speech and music related tasks. However, as raw waveform modelling remains an inherently difficult task, audio generative models are either computationally intensive, rely on low sampling rates, are complicated to control or restrict the nature of possible signals. Among those models, Variational AutoEncoders (VAE) give control over the generation by exposing latent variables, although they usually suffer from low synthesis quality. In this paper, we introduce a Realtime Audio Variational autoEncoder (RAVE) allowing both fast and high-quality audio waveform synthesis. We introduce a novel two-stage training procedure, namely *representation learning* and *adversarial fine-tuning*. We show that using a post-training analysis of the latent space allows a direct control between the reconstruction fidelity and the representation compactness. By leveraging a multi-band decomposition of the raw waveform, we show that our model is the first able to generate 48kHz audio signals, while simultaneously running 20 times faster than real-time on a standard laptop CPU. We evaluate synthesis quality using both quantitative and qualitative subjective experiments and show the superiority of our approach compared to existing models. Finally, we present applications of our model for *timbre transfer* and *signal compression*. All of our source code and audio examples are publicly available.

1 Introduction

Deep learning applied to audio signals proposes exciting new ways to perform speech generation, musical composition and sound design. Recent works in deep audio modelling have allowed novel types of interaction such as unconditional generation (Chung et al., 2015; Fraccaro et al., 2016; Oord et al., 2016; Vasquez & Lewis, 2019; Dhariwal et al., 2020) or timbre transfer between instruments (Mor et al., 2018). However, these approaches remain computationally intensive, as modeling audio raw waveforms requires dealing with extremely large temporal dimensionality. To cope with this issue, previous approaches usually rely on very low sampling rates (16 to 24kHz), which can be

3 Method

3.1 Two-stage training procedure

Ideally, the representation learned by a variational autoencoder should contain *high-level* attributes of the dataset. However, two perceptually similar audio signals may contain subtle phase variations that produce dramatically different waveforms. Hence, estimating the reconstruction term in equation (2) using the raw waveform penalizes the model if those subtle variations are not included in the learned representation. This might both hamper the learning process and include in the latent space those *low-level* variations about audio signal that are not relevant perceptually. To address this problem, we split the training process in two stages, namely *representation learning* and *adversarial fine-tuning*.

3.1.1 Stage 1: Representation learning

The first stage of our procedure aims to perform *representation learning*. We leverage the multiscale spectral distance $S(\cdot,\cdot)$ proposed by Engel et al. (2019) in order to estimate the distance between real and synthesized waveforms, defined as

$$S(\mathbf{x},\mathbf{y}) = \sum_{n\in\mathcal{N}} \left[\frac{\|\text{STFT}_n(\mathbf{x}) - \text{STFT}_n(\mathbf{y})\|_F}{\|\text{STFT}_n(\mathbf{x})\|_F} + \log\left(\|\text{STFT}_n(\mathbf{x}) - \text{STFT}_n(\mathbf{y})\|_1\right) \right], \tag{5}$$

where $\mathcal{N}$ is a set of scales, STFT_n is the amplitude of the Short-Term Fourier Transform with window size n and hop size $n/4$, and $\|\cdot\|_F$, $\|\cdot\|_1$ are respectively the Frobenius norm and L_1 norm. Using an amplitude spectrum-based distance does not penalize the model for inaccurately reconstructed phase, but encompasses important perceptual features about the signal. We train the *encoder* and *decoder* with the following loss derived from the ELBO

$$\mathcal{L}_{\text{vae}}(\mathbf{x}) = \mathbb{E}_{\hat{\mathbf{x}}\sim p(\mathbf{x}|\mathbf{z})}[S(\mathbf{x},\hat{\mathbf{x}})] + \beta \times \mathcal{D}_{\text{KL}}[q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})], \tag{6}$$

We start by training the model solely with $\mathcal{L}_{\text{vae}}$, and once this loss converges, we switch to the next training phase.

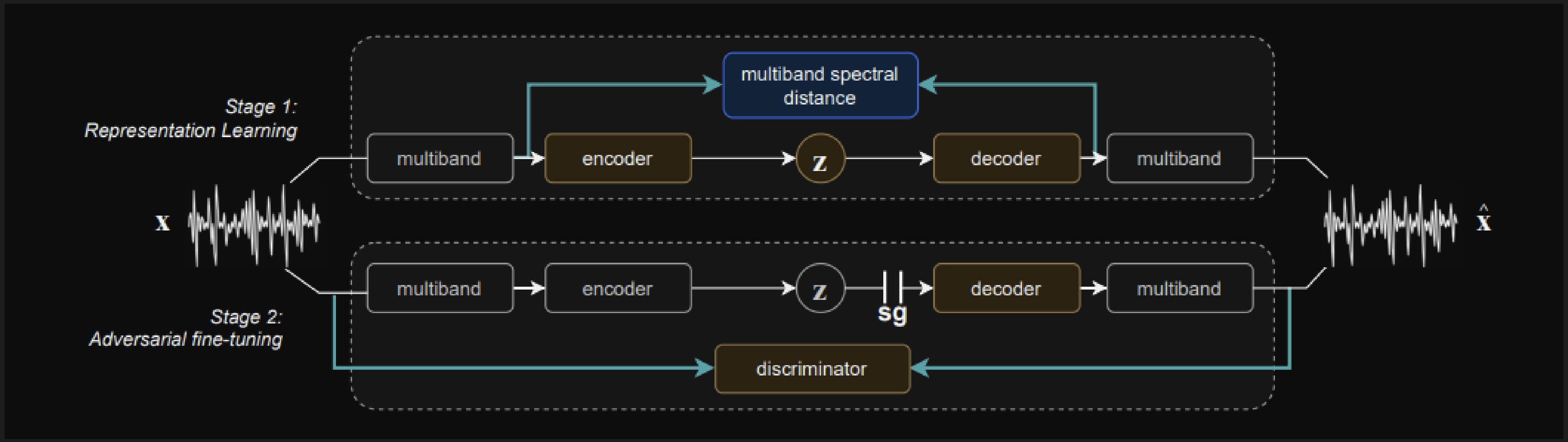
3.1.2 Stage 2: Adversarial fine-tuning

The second training stage aims at improving the synthesized audio quality and naturalness. As we consider that the learned representation has reached a satisfactory state at this point, we freeze the encoder and only train the decoder using an adversarial objective.

GANs are *implicit* generative models allowing to sample from a complex distribution by transforming a simpler one, called the *base distribution*. Here, we use the learned latent space in the first stage as the base distribution, and train the decoder to produce synthesized signals similar to the real ones by relying on a *discriminator* D. We use the hinge loss version of the GAN objective, defined as

$$\begin{aligned}\mathcal{L}_{\text{dis}}(\mathbf{x},\mathbf{z}) &= \max(0, 1 - D(\mathbf{x})) + \mathbb{E}_{\hat{\mathbf{x}}\sim p(\mathbf{x}|\mathbf{z})}[\max(0, 1 + D(\hat{\mathbf{x}}))],\\ \mathcal{L}_{\text{gen}}(\mathbf{z}) &= -\mathbb{E}_{\hat{\mathbf{x}}\sim p(\mathbf{x}|\mathbf{z})}[D(\hat{\mathbf{x}})].\end{aligned} \tag{7}$$

In order to ensure that the synthesized signal $\hat{\mathbf{x}}$ does not diverge too much from the ground truth $\mathbf{x}$, we keep minimizing the spectral distance defined in equation (5), but also add the feature matching



Antoine Caillon, et al., IRCAM, “RAVE: A variational autoencoder for fast and high-quality neural audio synthesis”, acids-ircam / RAVE Public

master | Go to file | Code

domkirke Update README.md ✓ 44498a0 · 7 months ago 484 Commits

.github/workflows	Update actions.yml	last year
.vscode	Add 1d spectral discriminator	last year
docs	update README	last year
rave	Passing to 2.3.1	last year
scripts	fixed tests + n_channels for wassers...	last year
tests	fixed tests + n_channels for wassers...	last year
.gitignore	split ci runs	last year
LICENSE	change LICENSE to CC-BY-NC-4.0	2 years ago
MANIFEST.in	include augmentation path in build ...	last year
README.md	Update README.md	7 months ago
requirements.txt	last fixes	last year
setup.py	attaching versioning to code	last year

README | License

RAVE: Realtime Audio Variational autoEncoder

About

Official implementation of the RAVE model: a Realtime Audio Variational autoEncoder

audio ai deep-learning neural-network generative-model

Readme
View license
Activity
1.3k stars
43 watching
182 forks
Report repository

Releases 1

v2.3.1 Latest
on Dec 19, 2023

Packages

No packages published

Contributors 4

caillonantoine Antoine Caillon
domkirke Axel Chemla--Romeu-Santos
capital-G Dennis Scheiba
acids-ircam ACIDS

Languages

Python 100.0%

RAVE: Realtime Audio Variational autoEncoder

Official implementation of *RAVE: A variational autoencoder for fast and high-quality neural audio synthesis* (article link) by Antoine Caillon and Philippe Esling.

If you use RAVE as a part of a music performance or installation, be sure to cite either this repository or the article !

If you want to share / discuss / ask things about RAVE you can do so in our discord server !

Please check the FAQ before posting an issue!

RAVE VST RAVE VST for Windows, Mac and Linux is available as beta on the corresponding Forum IRCAM webpage. For problems, please write an issue here or on the Forum IRCAM discussion page.

Tutorials : new tutorials are available on the Forum IRCAM webpage, and video versions are coming soon!

- Tutorial: Neural Synthesis in a DAW with RAVE
- Tutorial: Neural Synthesis in Max 8 with RAVE
- Tutorial: Training RAVE models on custom data

The latest in Machine Learning

paperswithcode.com

asdf

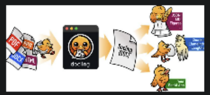
Browse State-of-the-Art | Datasets | Methods | More

Sign In

Top | New | Greatest

Trending Research

Subscribe



Docling Technical Report

DS4SD/docling • • 19 Aug 2024

This technical report introduces Docling, an easy to use, self-contained, MIT-licensed open-source package for PDF document conversion.

★ 7,866

3.07 stars / hour

Paper

Code

Tencent Hunyuan

Hunyuan-Large: An Open-Source MoE Model with 52 Billion Activated Parameters by Tencent

tencent/tencent-hunyuan-large • • 4 Nov 2024

In this paper, we introduce Hunyuan-Large, which is currently the largest open-source Transformer-based mixture of experts model, with a total of 389 billion parameters and 52 billion activation parameters, capable of handling up to 256K tokens.

Logical Reasoning | Mathematical Problem-Solving

★ 976

2.39 stars / hour

Paper

Code

ADOPT: Modified Adam Can Converge with Any β_2 with the Optimal Rate

ishohei220/adopt • • 5 Nov 2024

Adam is one of the most popular optimization algorithms in deep learning.

Image Classification

★ 192

1.65 stars / hour

Paper

Code

TableGPT2: A Large Multimodal Model with Tabular Data Integration

tablegpt/tablegpt-agent • 4 Nov 2024

In response, we introduce TableGPT2, a model rigorously pre-trained and fine-tuned with over 593. 8K tables and 2. 36M high-quality query-table-output tuples, a scale of table-related data unprecedented in prior research.

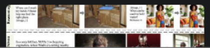
Benchmarking | Data Integration

★ 139

1.65 stars / hour

Paper

Code



OmniGen: Unified Image Generation

vectorspacelab/omnigen • • 17 Sep 2024

★ 2,301

The Whole Is Greater than the Sum of Its Parts: Improving Music Source Separation by Bridging Network

13 May 2023 · Ryosuke Sawata, Naoya Takahashi, Stefan Uhlich, Shusuke Takahashi, Yuki Mitsufuji · Edit social preview

This paper presents the crossing scheme (X-scheme) for improving the performance of deep neural network (DNN)-based music source separation (MSS) with almost no increasing calculation cost. It consists of three components: (i) multi-domain loss (MDL), (ii) bridging operation, which couples the individual instrument networks, and (iii) combination loss (CL). MDL enables the taking advantage of the frequency- and time-domain representations of audio signals. We modify the target network, i.e., the network architecture of the original DNN-based MSS, by adding bridging paths for each output instrument to share their information. MDL is then applied to the combinations of the output sources as well as each independent source; hence, we called it CL. MDL and CL can easily be applied to many DNN-based separation methods as they are merely loss functions that are only used during training and do not affect the inference step. Bridging operation does not increase the number of learnable parameters in the network. Experimental results showed that the validity of Open-Unmix (UMX), densely connected dilated DenseNet (D3Net) and convolutional time-domain audio separation network (Conv-TasNet) extended with our X-scheme, respectively called X-UMX, X-D3Net and X-Conv-TasNet, by comparing them with their original versions. We also verified the effectiveness of X-scheme in a large-scale data regime, showing its generality with respect to data size. X-UMX Large (X-UMXL), which was trained on large-scale internal data and used in our experiments, is newly available at https://github.com/asteroid-team/asteroid/tree/master/egs/musdb18/X-UMX.

PDF Abstract

Code

Edit

asteroid-team/asteroid official ★ 2,269

Tasks

Edit

Music Source Separation

Datasets

Edit

MUSDB18

Results from the Paper

Edit

Submit results from this paper to get state-of-the-art GitHub badges and help the community compare results to other papers.

Methods

Edit

1x1 Convolution • Average Pooling • Batch Normalization • Concatenated Skip Connection • Convolution • Dense Block • Dense Connections • Dropout • Global Average Pooling • Kaiming Initialization • Max Pooling • MDL • ReLU • Softmax

Show Me the Instruments: Musical Instrument Retrieval from Mixture Audio

15 Nov 2022 · KyungSu Kim, Minju Park, Haesun Joung, Yunkee Chae, Yeongbeom Hong, SeongHyeon Go, Kyogu Lee · Edit social preview

As digital music production has become mainstream, the selection of appropriate virtual instruments plays a crucial role in determining the quality of music. To search the musical instrument samples or virtual instruments that make one's desired sound, music producers use their ears to listen and compare each instrument sample in their collection, which is time-consuming and inefficient. In this paper, we call this task as Musical Instrument Retrieval and propose a method for retrieving desired musical instruments using reference music mixture as a query. The proposed model consists of the Single-Instrument Encoder and the Multi-Instrument Encoder, both based on convolutional neural networks. The Single-Instrument Encoder is trained to classify the instruments used in single-track audio, and we take its penultimate layer's activation as the instrument embedding. The Multi-Instrument Encoder is trained to estimate multiple instrument embeddings using the instrument embeddings computed by the Single-Instrument Encoder as a set of target embeddings. For more generalized training and realistic evaluation, we also propose a new dataset called Nlakh. Experimental results showed that the Single-Instrument Encoder was able to learn the mapping from the audio signal of unseen instruments to the instrument embedding space and the Multi-Instrument Encoder was able to extract multiple embeddings from the mixture of music and retrieve the desired instruments successfully. The code used for the experiment and audio samples are available at: https://github.com/minju0821/musical_instrument_retrieval

PDF Abstract

Code

Edit

minju0821/musical_instrument_retrie... ★ 27 PyTorch

official

Tasks

Edit

Retrieval

Datasets

Edit

Introduced in the Paper:

Nlakh

Used in the Paper:

NSynth

Results from the Paper

Edit

Submit results from this paper to get state-of-the-art GitHub badges and help the community compare results to other papers.

musical_instrument_retrieval Public

Watch 1 Fork 4 Star 27

main Go to file + Code

minju0821 Update README.md 375e28f · last year

Multi_Instrument_En...	codes for multi_inst_...	2 years ago
Single_Instrument_E...	path arguments mod...	2 years ago
dataset	Update README.md	last year
evaluation	dataset for eval. mod...	2 years ago
models	uploade requirement...	2 years ago
LICENSE	Update LICENSE	2 years ago
README.md	Update README.md	2 years ago
requirements.txt	uploade requirement...	2 years ago

README MIT license

Show Me the Instruments: Musical Instrument Retrieval from Mixture Audio

This repository contains the code and the dataset for our paper

About

No description, website, or topics provided.

Readme

MIT license

Activity

27 stars

1 watching

4 forks

Report repository

Releases

No releases published

Packages

No packages published

Contributors 2

Haessun

minju0821 Minju Park

Languages

Inference vs. Training

Deployment – use of trained models

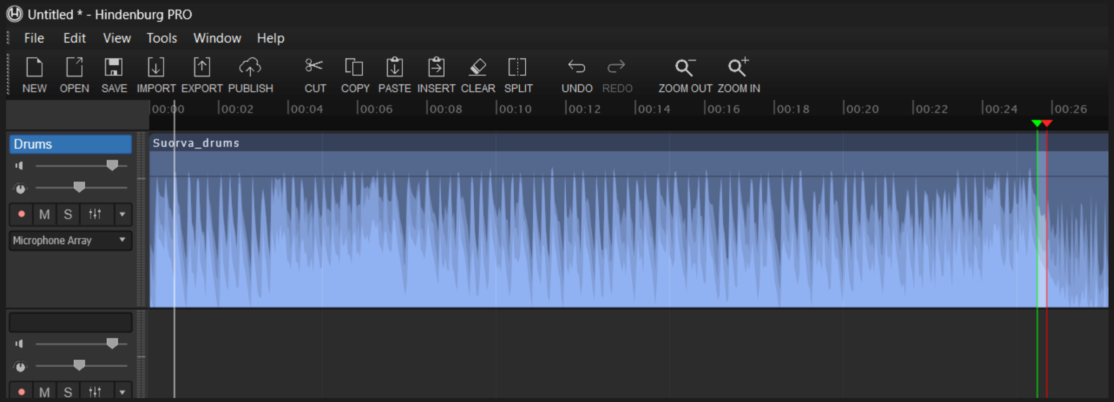
Saving and converting a trained model

Inference runtimes
- ONNX
- RT Neural
- TF Lite

Get started...!

- Regular Computer (CPU) + Cloud or Computer with GPU (Laptop/Desktop) or Mac with M3 / M4
- VS Code
    - Python plugins
- Python
    - Miniconda
- Check out other projects
- First use research datasets to learn
- Make you own data! - or collaborate with someone...
- Write your own training code or use a high-level framework (Fastai, Lightning)
- Create!

Cyano



https://open.spotify.com/album/0r02xBsC2yJxStKxhvFd5m?si=c3BW5We5R7eJcQk30Ie8-g