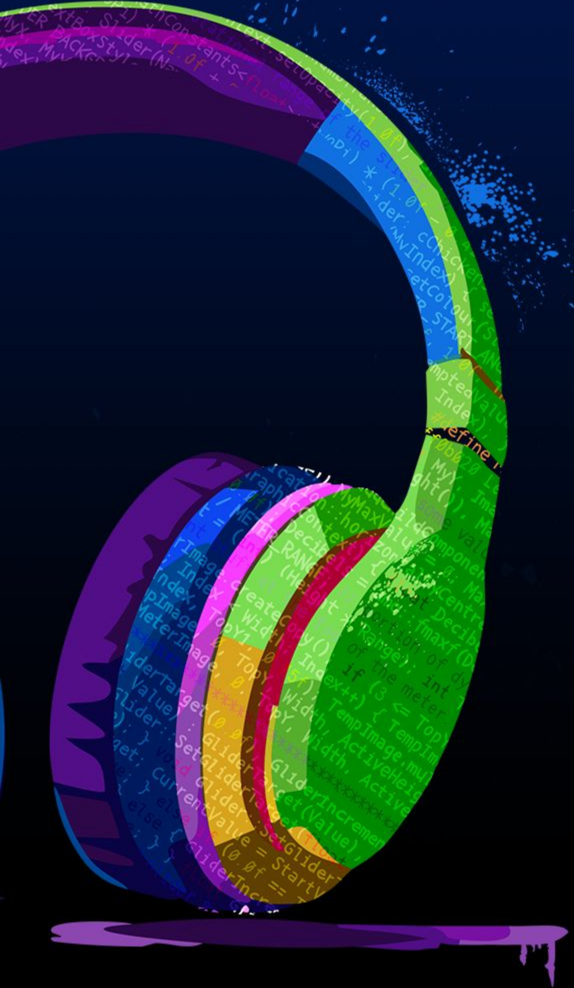
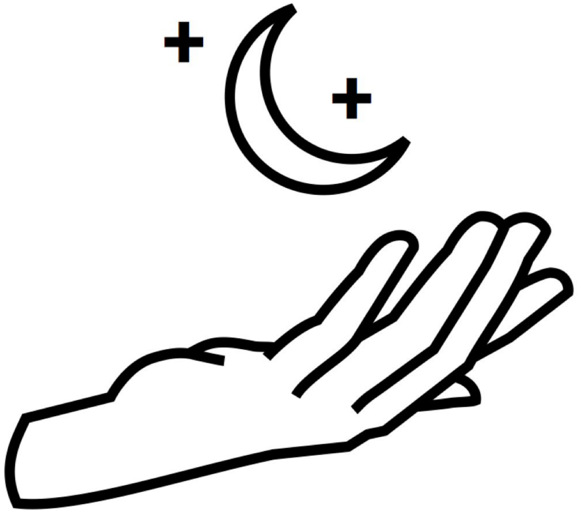


ADC24 Bristol

WORKSHOP:

PRACTICAL MACHINE LEARNING

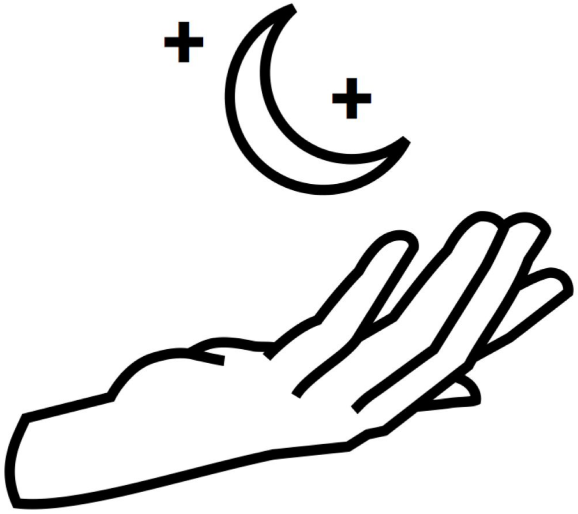
ANNA WSZEBOROWSKA
& HARRIET DRURY, SOHYUN IM,
JULIA LÄGER & PAULINE NEMCHAK



@dynamic_cast

Practical
Machine Learning

ADC 2024



@dynamic_cast

Harriet Drury

Julia Läger

Sohyun Im

Pauline Nemchak

Anna Wszeborowska

dynamic-cast.github.io

Presentation slides

https://github.com/dynamic-cast/ADC24/wiki

Workshop goals

- Demonstrate a practical example of using machine learning in an audio application
- Show you how to use an existing generative model and embed it in an application
- Walk you through the stages of training your own machine learning model
    - We will build a model which gives us the ability to control the generative model

Machine Learning

- Machine Learning is a field of Artificial Intelligence focusing on algorithms which can make sense of the data you feed them.
- These models can learn from data, and find patterns in that data without being explicitly programmed.

Generative models

- Generative - having the ability to generate new data
- Generative models try to understand the patterns and structures of the data they were fed (dataset) and generate new, yet similar examples

AI Music Generation

- Song generators
- Sample generators
- Instruments

AI Music Generation

- Sound generated in response to:
  - Text prompt - a description of what we want to hear
  - A set of features (style, mood, tempo, etc.)
  - No input - eg. hitting a button “generate”
  - Audio - providing a sound we want to transform

Style Transfer

- *"Style Transfer is a technique in computer vision and graphics that involves generating a new image by combining the content of one image with the style of another image. The goal of style transfer is to create an image that preserves the content of the original image while applying the visual style of another image."* [1]
- In the audio domain: generating a new sound in the style of the training data but preserving some characteristics of the input sound
  - Also called timbre transformation/transfer

[1] https://paperswithcode.com/task/style-transfer

Interacting with generative models in musical contexts

- What does a meaningful interaction with a generative model look like?
  - Mapping large parameter spaces
- Continuous exploration of interfaces which allow a sense of control and agency
  - Are we generating content or playing an instrument?
  - Embodied interaction

Challenges associated with building generative models

- Dataset collection
- Training times
  - Small models which train fast tend to work with low sampling rates
- Access to computing power
- Technical literacy
- Ethical considerations
  - Copyright, bias, cost (financial, environmental, cultural), access to the technology

Part I: Embedding a generative model in a music app

Demo

Overview of the app

- Desktop* app with web UI
- Audio is produced locally
- Uses Python and Flask
- Audio engine runs in a separate thread

Overview of the generative model

In the app we use **RAVE** (**R**ealtime **A**udio **V**ariational auto**E**ncoder)

- Developed at IRCAM
- Enables fast and high-quality neural audio synthesis
- A lot of tutorials and helper tools available to support training your own models

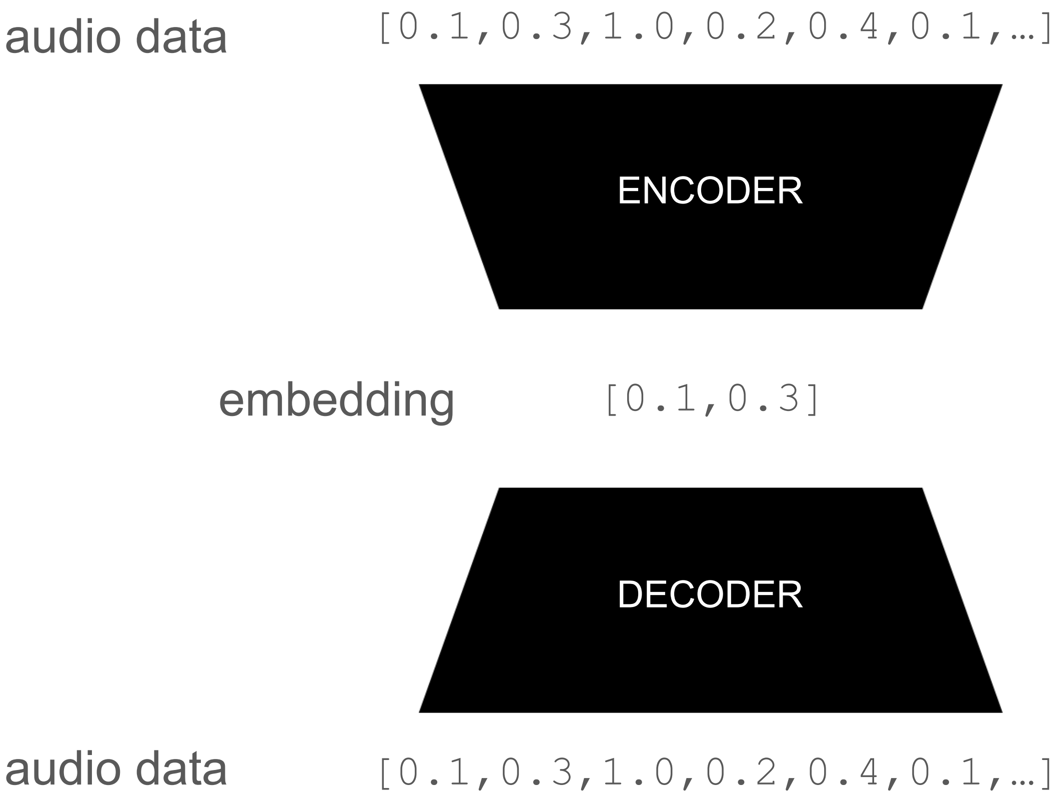
Variational AutoEncoder (VAE)

- Can learn important properties of the data it was exposed to
- Can use this knowledge to generate novel sounds which imitate the data the model was trained on
- Can be controlled - when generating new sounds, you can specify the desired direction

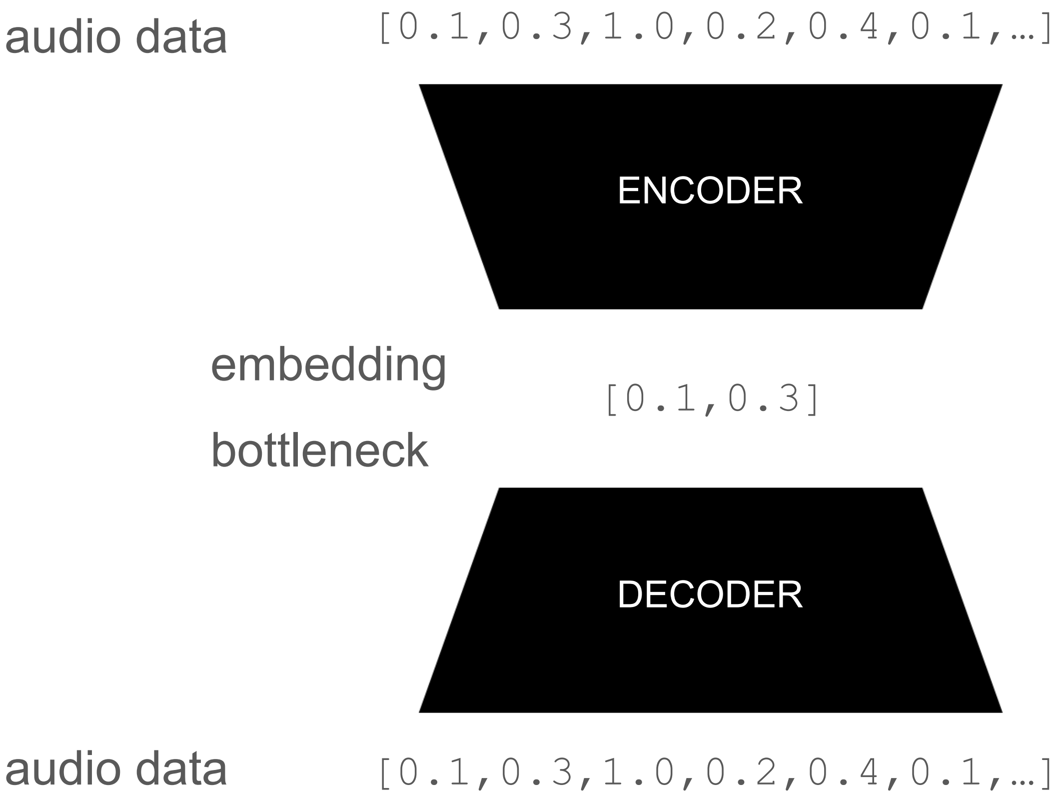
Autoencoder

- Encoder-Decoder architecture (two networks)
- Encoder takes an input and compresses it to a much smaller representation (the *encoding*)
- Decoder can convert the encoding back to the original input
- Conclusions:
    - The encoding contains enough information for the decoder network to reconstruct it.
    - The encoder learns the most important properties of the input data and discards irrelevant parts

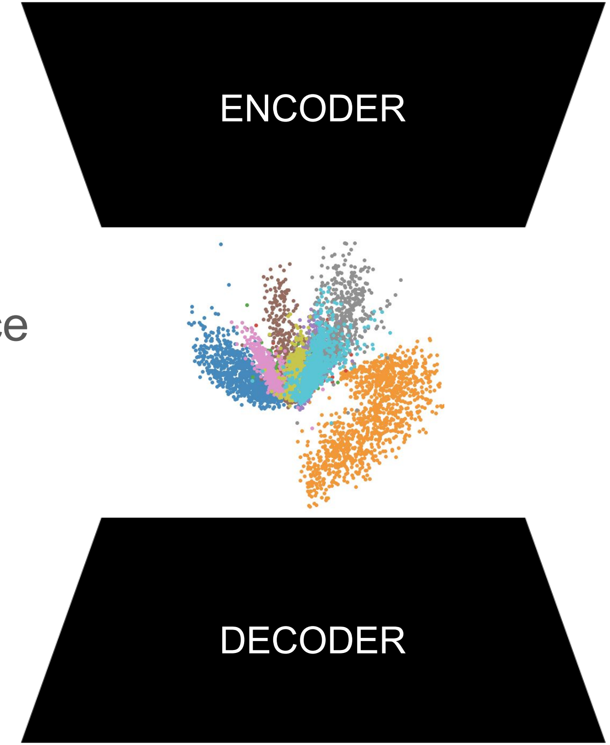
Autoencoder



Autoencoder



Autoencoder



embedding space

latent space

2D (in our example)

each embedding represents a coordinate on the latent space

Latent space coordinates

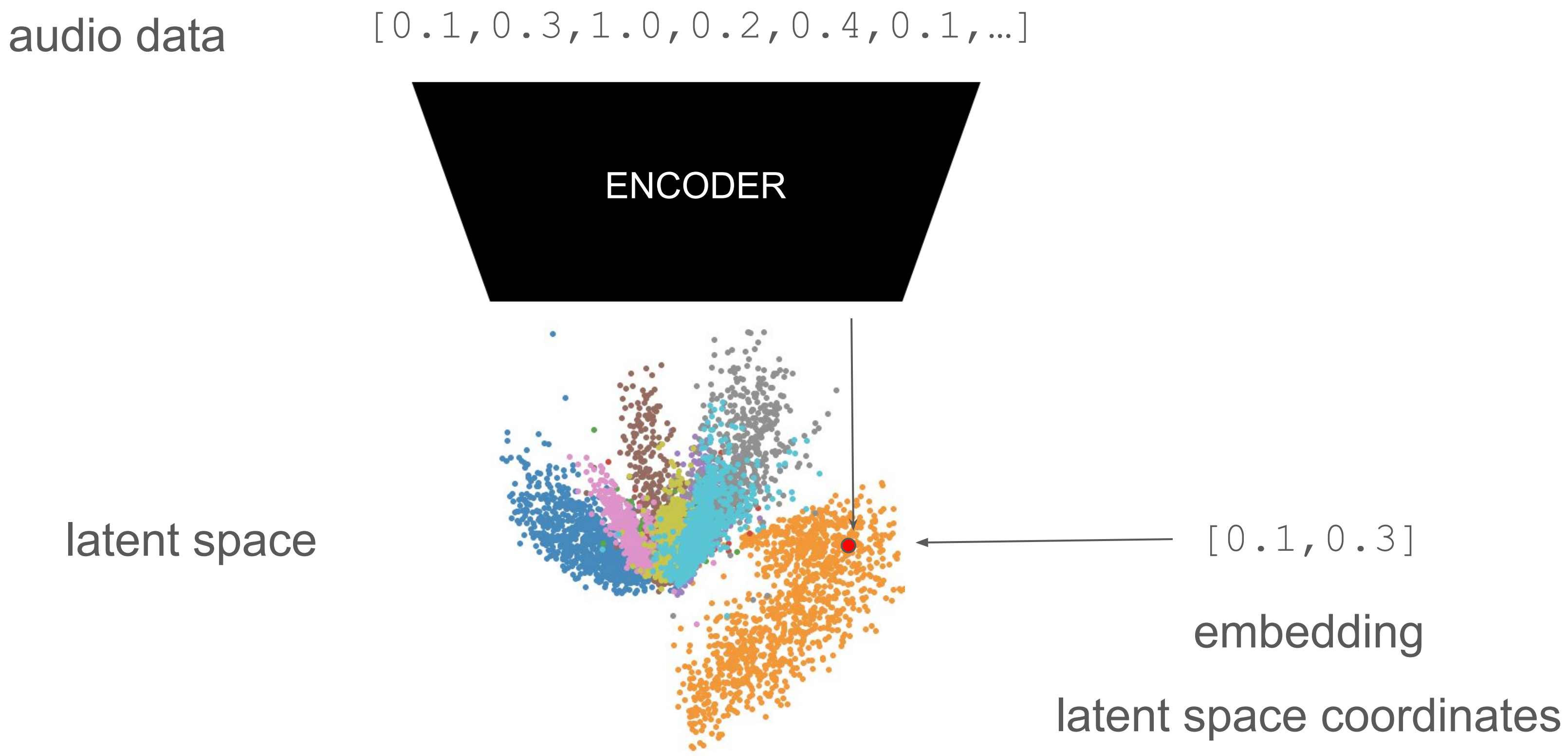
`[0.1, 0.3]` 2D latent space

`[0.1, 0.3, 0.2]` 3D latent space

`[0.1, 0.3, 0.2, 1.0]` 4D latent space

Each latent space dimension corresponds to a specific feature or characteristic learnt during the training process

Encoding

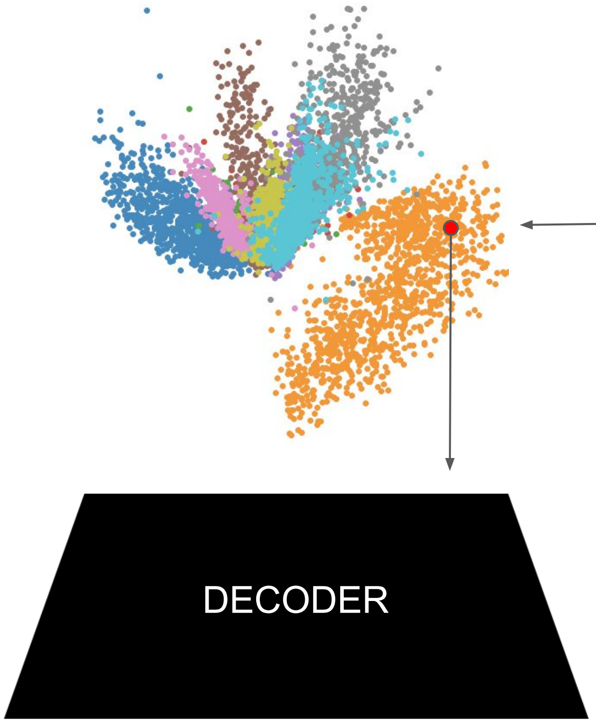

audio data
[0.1,0.3,1.0,0.2,0.4,0.1,…]
ENCODER
latent space
[0.1,0.3]
embedding
latent space coordinates

Decoding

latent space

sample a point
from the latent space

[0.1,0.3]



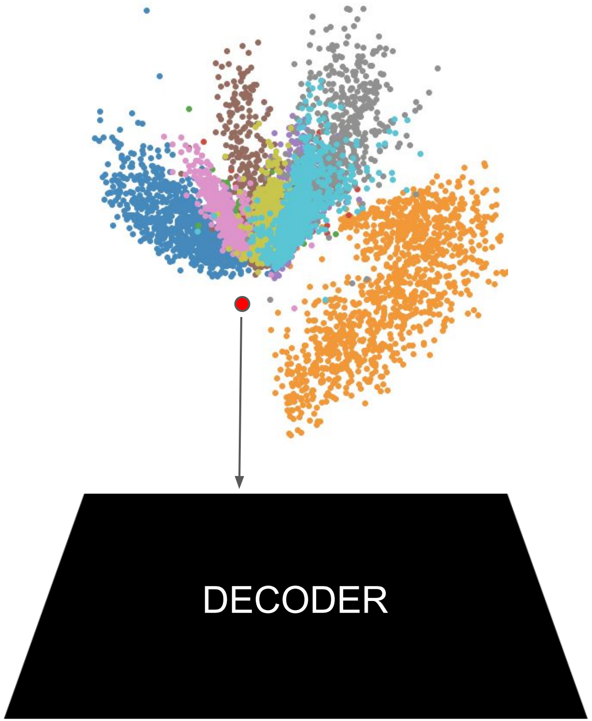
pass it through
the decoder

reconstruct input data

[0.1,0.3,1.0,0.2,0.4,0.1,…]

Generating novel sounds

latent space



sample a point
from the unpopulated
part of the latent space

Autoencoder - challenges

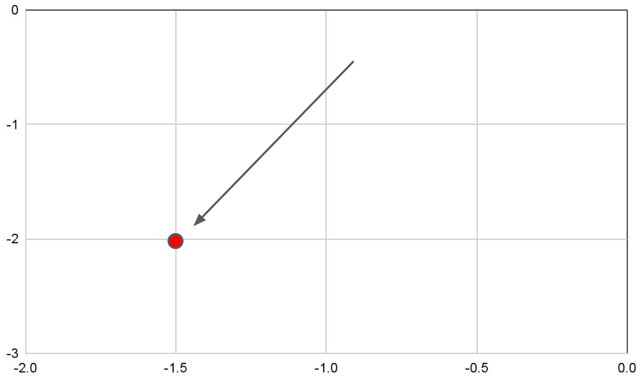
- The latent space is organised in clusters of similar things close to each other
- The latent space may not be continuous (has gaps between clusters). Sampling from the gaps will generate unrealistic output because the decoder has no idea how to deal with that region of the latent space
- In order to generate novel sounds we want to sample from the gaps and smoothly transition between different regions

Variational AutoEncoder (VAE)

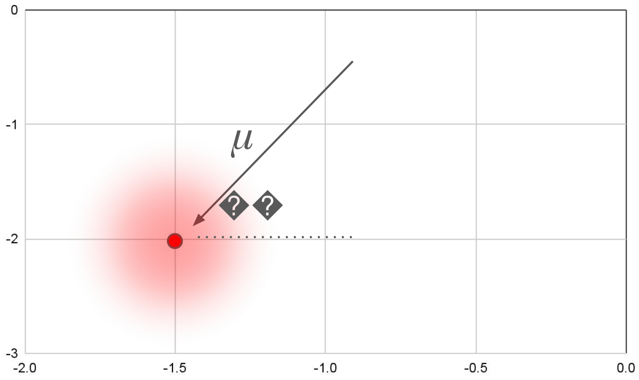
- Introduces two tricks to make VAEs suitable for generative modelling:
    - The latent space is continuous *by design*
    - The latent space is centered around the origin

Variational AutoEncoder (VAE)

- The latent space is continuous *by design* which makes VAEs suitable for generative modelling.



Autoencoder



Variational Autoencoder

μ and σ initialise a probability distribution

Variational AutoEncoder (VAE)

- The encoder does not output a single encoding vector but two vectors: a vector of means (μ) and a vector of standard deviations (σ) from which we sample to obtain the encoding we pass to the decoder
- Since encodings are generated at random from anywhere in the distribution (the “circle”), the decoder learns that not only a single point on the latent space represents the input sample but all nearby points refer to it as well
- Thanks to being exposed to a range of variations of the encoding of the same input during training, the decoder trains not only to decode specific encodings but ones that slightly vary, too

Variational AutoEncoder (VAE)

- Centering around the origin:
  - During training, clustering encodings apart into specific regions gets penalised
  - All encodings end up evenly distributed around the center of the latent space
- Optimising both reconstruction loss and divergence loss results in a latent space which maintains clusters of similar encodings nearby on the local scale, but globally it is densely packed around the latent space origin

Navigating the latent space

- Vector arithmetic
- Encoding an audio buffer and adding (or subtracting) from the embedding vector

Interacting with the model

- Interact with the model from Jupyter Notebook
- Instantiating the audio engine
- Starting / stopping the engine
- Toggling style transfer
- Navigating the latent space

Setting up the workshop project

- Go to the ADC24 workshop repository
  https://github.com/dynamic-cast/ADC24

Setting up the workshop project

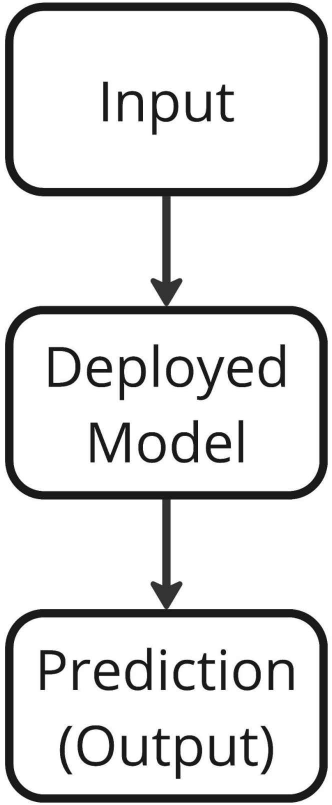
- Follow the steps in jupyter_setup.md
  https://github.com/dynamic-cast/ADC24/blob/main/jupyter_setup.md

Embedding the RAVE Model

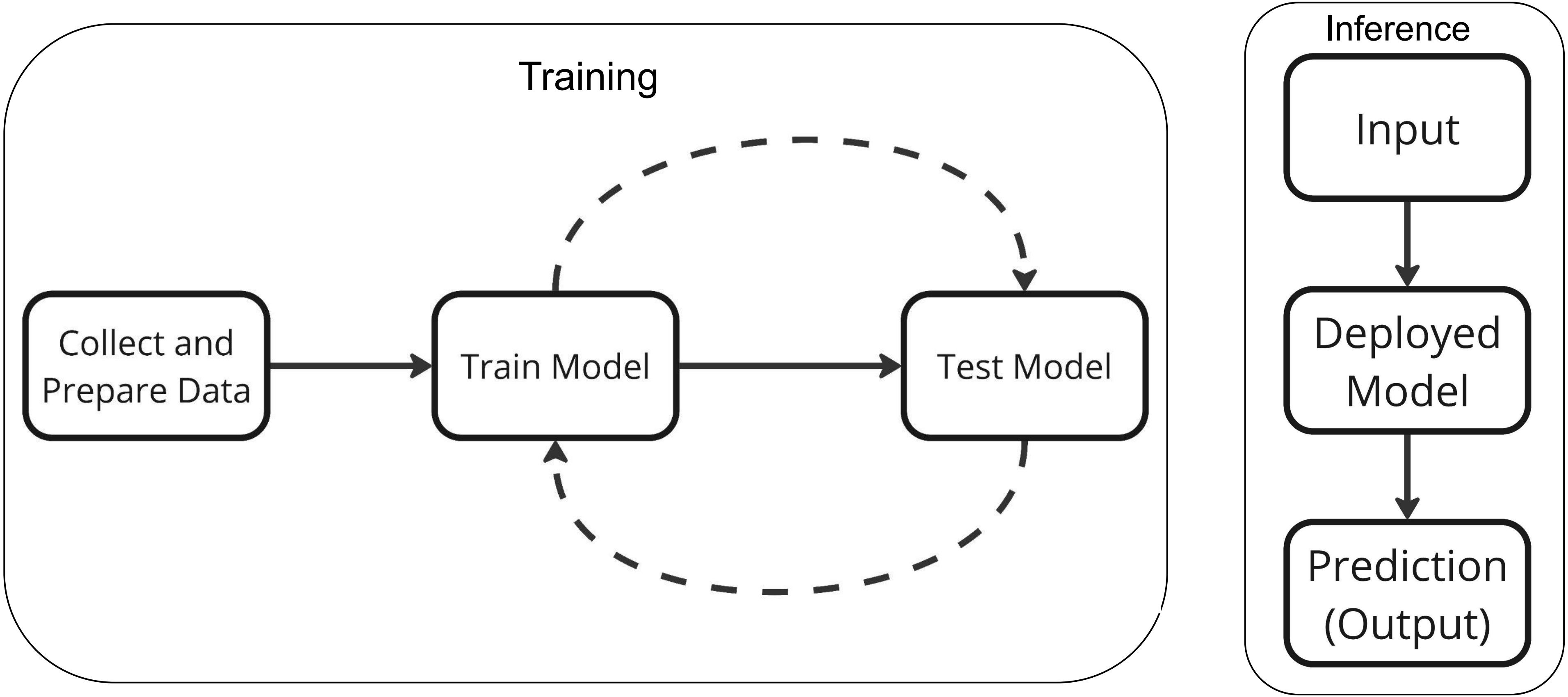
Inference - What is it?

The process of giving a machine learning model unseen data to output (predict) a value.

1. Load the Model
2. Preprocess Data
3. Run Inference
4. Interpret Output



Machine Learning Pipeline



Inference in Production Audio, Briefly

In audio contexts, we have to think about how quickly something can run. If we want things to run in ‘real time’, we need to consider:

- Audio Loading & Streaming
  - Avoid Memory Bottlenecks
- Real-Time Considerations
  - Buffers for Real-Time Streaming
  - Latency-Aware Code Structure
- Unbounded Execution Times
  - Allocations, etc
  - Garbage Collection

Note:- These are things to think about in production code

Interacting with Latent Space

We have several methods to interact with the **RAVE** model.

- encode: Encodes input data into a latent representation.
- forward: Runs the model's forward pass to process input data.
- decode: Decodes latent representation back into the original data space.

Interacting with Latent Space

encode

The encode method is responsible for transforming input data into a latent representation. This is often used in models that involve some form of compression or feature extraction. For example, in an autoencoder, the encoder part of the model compresses the input data into a lower-dimensional latent space.

```
latent_representation = rave_model.encode(input_audio)
```

Interacting with Latent Space

forward

The forward method is the main method that defines the computation performed at every call. In PyTorch, for example, the forward method is where the actual computation of the model happens. It takes input data and passes it through the model's layers to produce the output.

```
output_audio = rave_model.forward(input_audio)
```

Interacting with Latent Space

decode

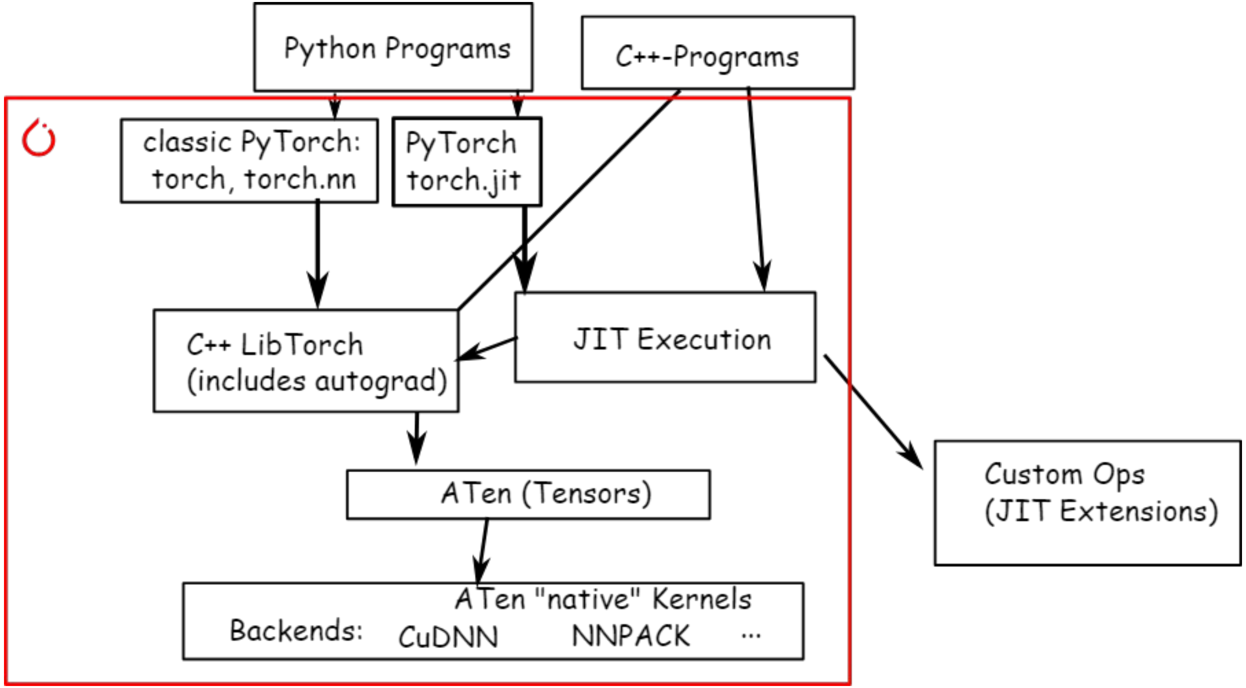
The decode method takes the latent representation produced by the encode method and transforms it back into the original data space. In an autoencoder, the decoder part of the model reconstructs the original data from the latent representation.

```
reconstructed_audio = rave_model.decode(latent_representation)
```

Embedding a Trained Model Using Pytorch

There are common tools out there that can run inference for us.

In this example we are using torchscript's JIT engine.



From: https://lernapparat.de/jit-optimization-intro

Let's Look at apply_transformation

```
def _apply_transformation(self, buffer):
    torch.set_grad_enabled(False)
    input_data = torch.Tensor(buffer)
    input_data = torch.reshape(input_data, (1, 1, buffer.size))
    encoded = self._rave_model.encode(input_data)
    encoded[0][:,0] += self._latent_coordinates
    decoded = self._rave_model.decode(encoded)
    return decoded[0][0][:buffer.size].numpy()
```

Part II: Training custom interactions with a generative model

Demo

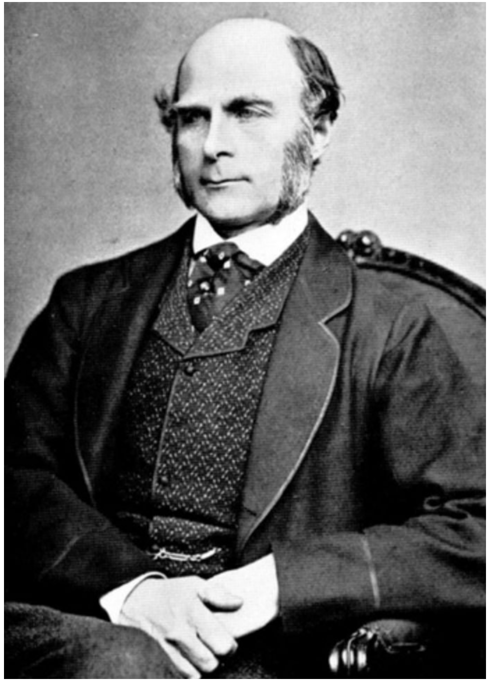
What was the demo showing?

The demo was all about training a regression neural network model based on the input/output data that you created yourselves.

What was the demo showing?

The demo was all about training a regression neural network model based on the input/output data that you created yourselves.

What is regression?



- Francis Galton (1822 - 1911)
- Statistician
- Regression analysis

"... While childrens' heights are influenced by their parents, they tend to regress toward the average height of the general population."

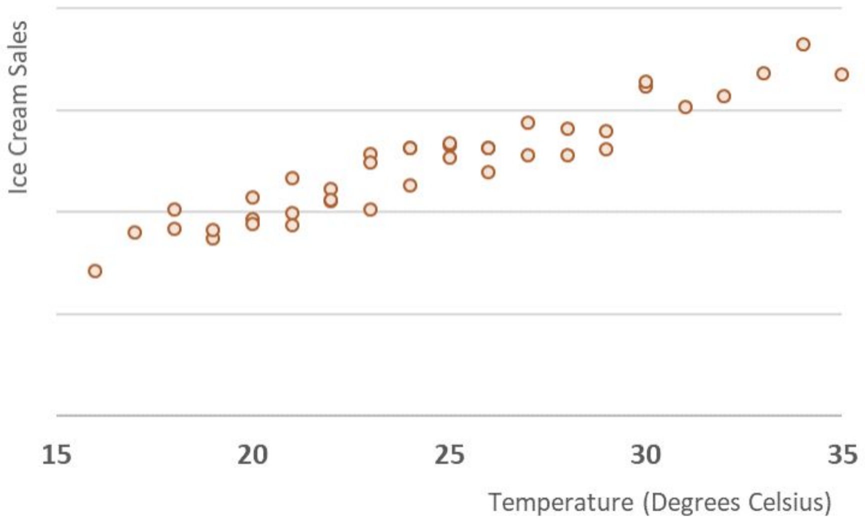
What is regression?

When you analyse a regression problem...

- To find a trend/pattern between certain factors and the corresponding outcomes

- To define this trend/pattern as a mathematical expression/model, and use it as a system to predict an outcome based on any given input.

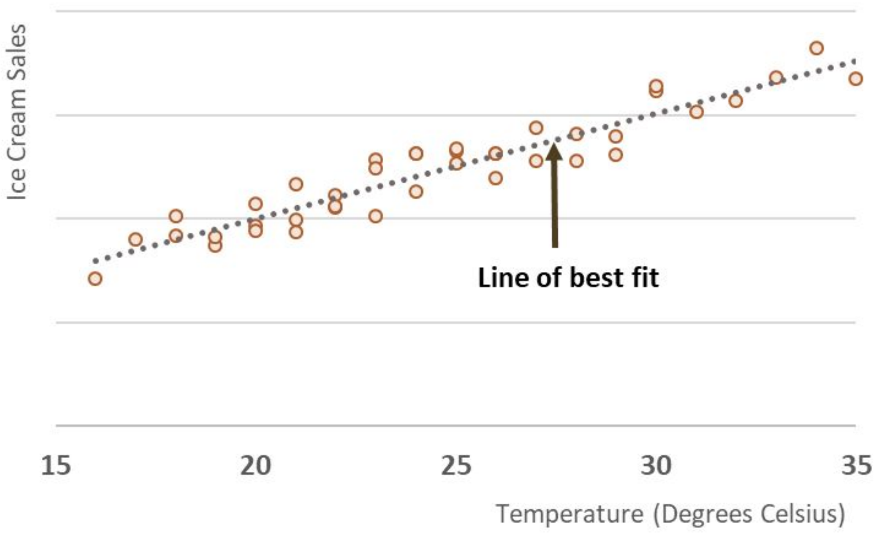
The simplest example of regression



- X-axis: Daily average temperature
- Y-axis: Ice Cream Sales

http://sensibleanalytics.co.uk/how-key-driver-analysis-help/

The simplest example of regression



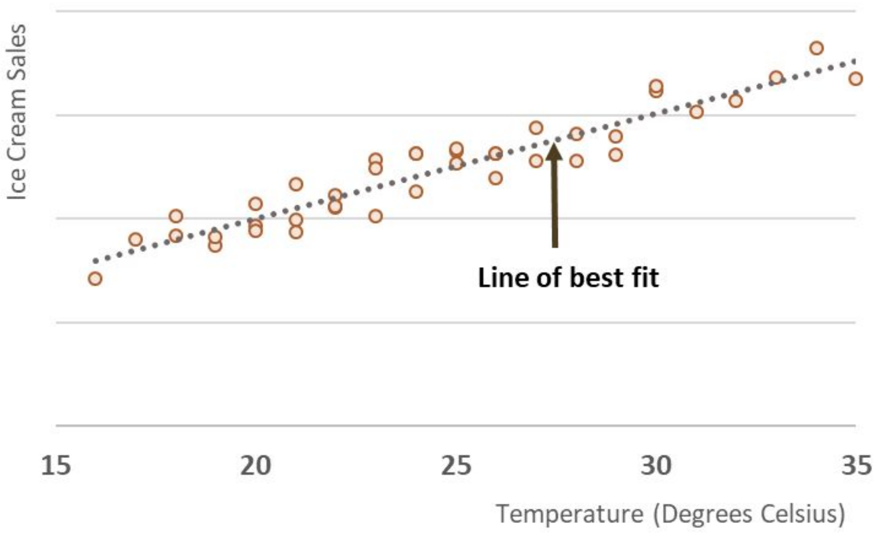
The "**Line of best fit**"...

- describes the data pattern well.
- can be represented by a simple mathematical expression.

$y=ax+b$ (a: slope / b: y-intercept)

http://sensibleanalytics.co.uk/how-key-driver-analysis-help/

The simplest example of regression

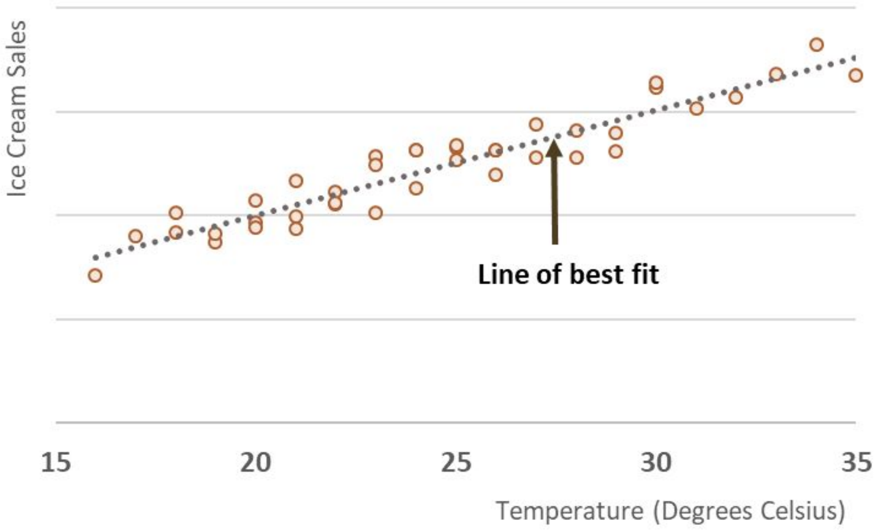


- One feature in one input
- One feature in one output

- The pattern of data is drawn as a linear line

http://sensibleanalytics.co.uk/how-key-driver-analysis-help/

The simplest example of regression



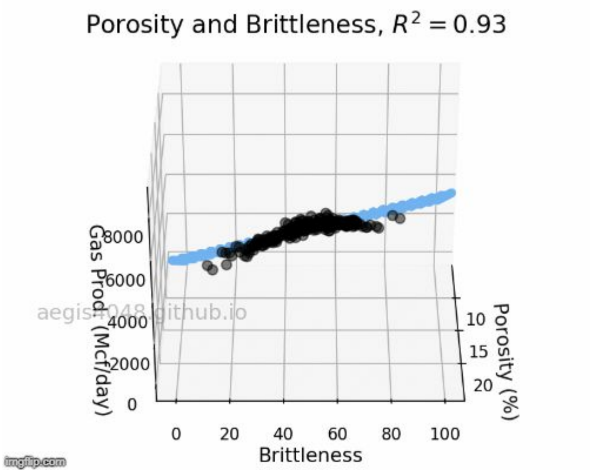
- One feature in one input
- One feature in one output
  → can be multiple features

- The pattern of data is drawn as a linear line
  → nonlinear data pattern is possible

http://sensibleanalytics.co.uk/how-key-driver-analysis-help/

Complex regression problems

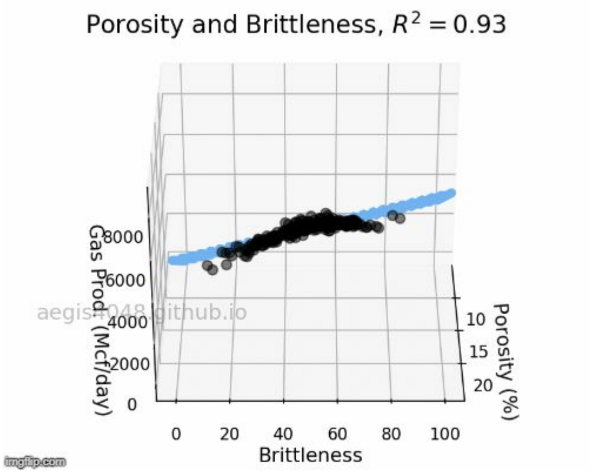
1. Multi-dimensionality



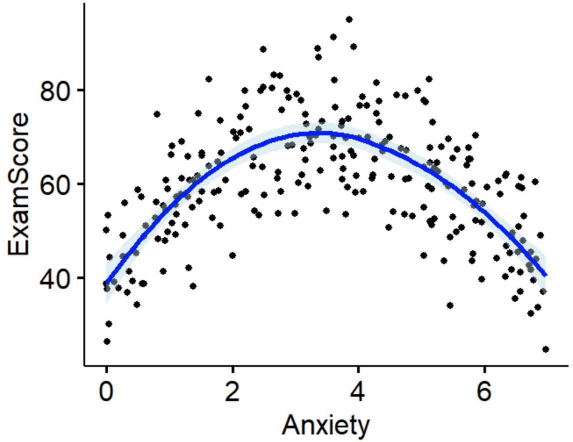
https://aegis4048.github.io/mutiple_linear_regression_and_visualization_in_python

Complex regression problems

1. Multi-dimensionality



2. Nonlinearity

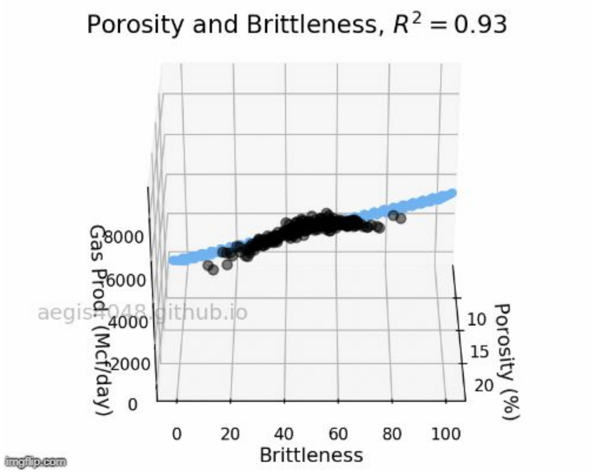


https://aegis4048.github.io/mutiple_linear_regression_and_visualization_in_python

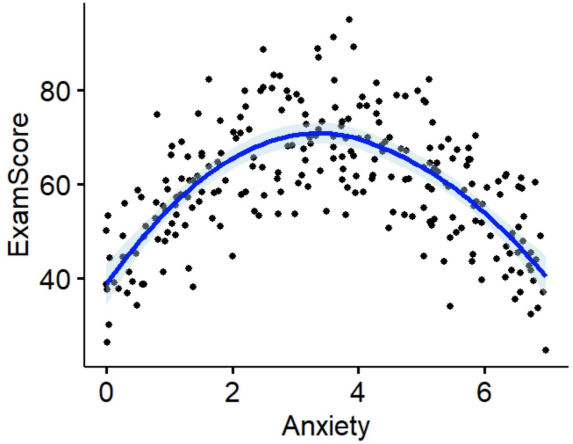
https://www.alexanderdemos.org/Class5.html

Complex regression problems

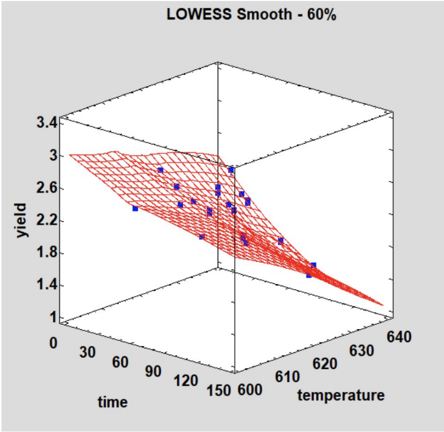
1. Multi-dimensionality



2. Nonlinearity



3. Multi-D + Nonlinear



https://aegis4048.github.io/mutiple_linear_regression_and_visualization_in_python

https://www.alexanderdemos.org/Class5.html

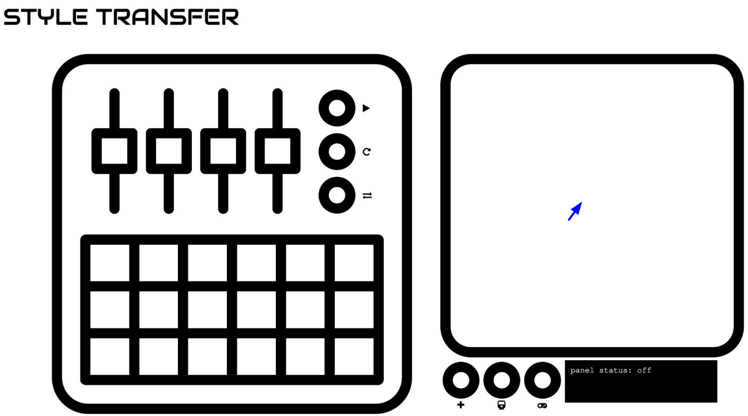
https://www.statgraphics.com/blog/nonlinear_regression

Why Neural Network?

- Multi-layer neural network can derive a mathematical model which is well fitting the nonlinear and complex pattern in the data.

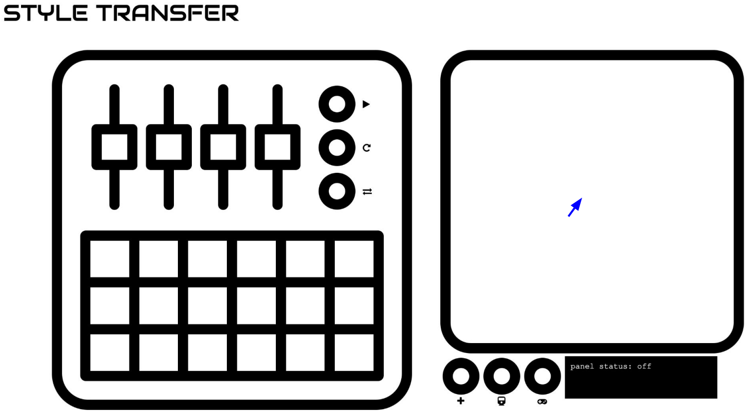
- How?

Inputs/outputs data



- Input features
  - x-coordinate
  - y-coordinate
- Output features
  - 4 sliders’ values

Inputs/outputs data



1 data point = 2 input features + 4 output features

(x,y coordinates) (4 sliders' positions)

Goals & Aims with using NN

Goal:

To gather a lot of this data point and come up with a mathematical model that best captures the nonlinear, multi-dimension, arbitrary patterns shown by the dataset.

Aim:

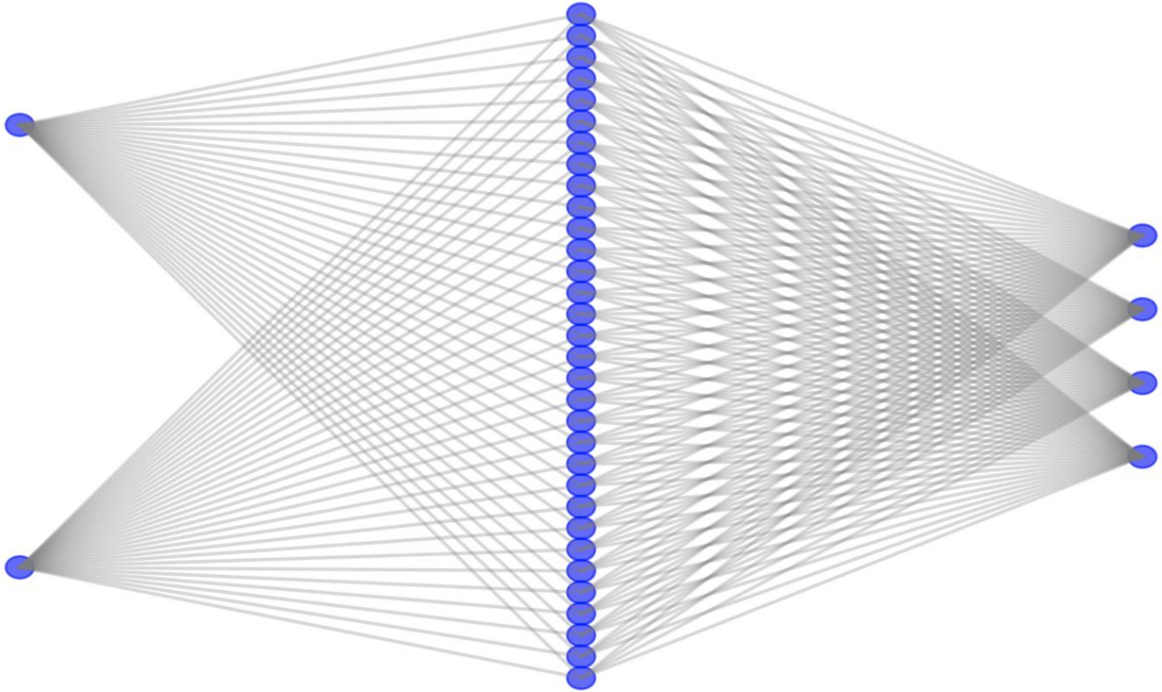
Use the mathematical model as a system that takes any given input, and predicts the corresponding output.

How it works?

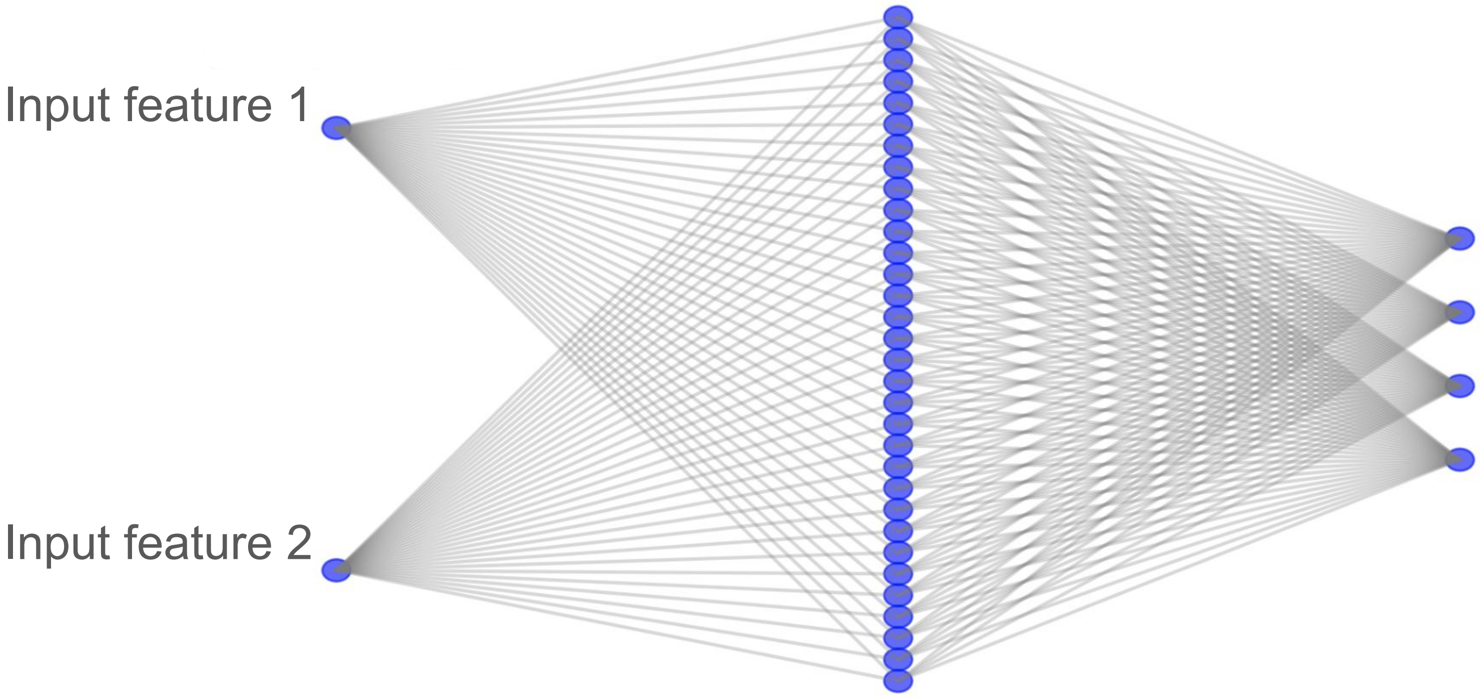
- Multi-layer neural network can capture the nonlinear and complex pattern in the data

How?

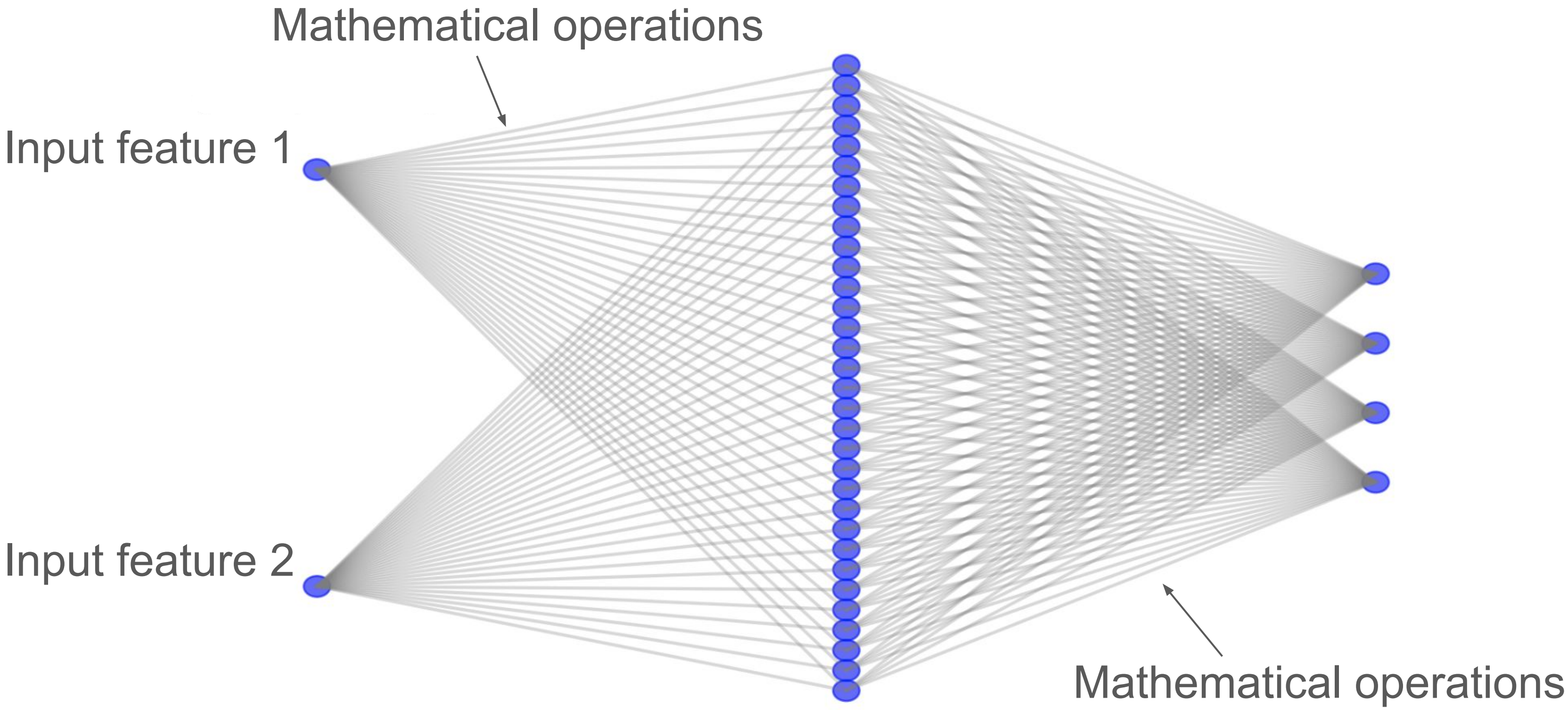
Visualization of Neural Network Architecture



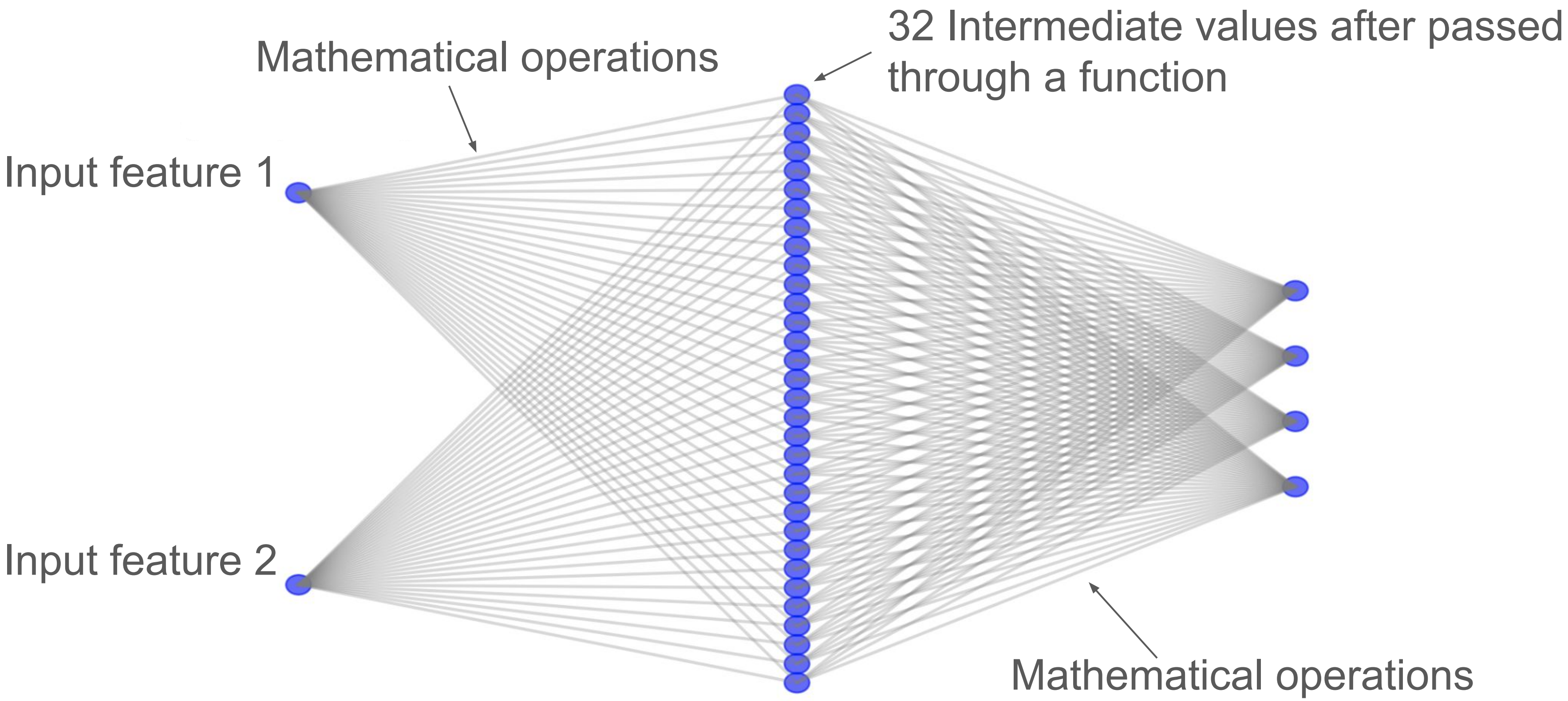
Visualization of Neural Network Architecture



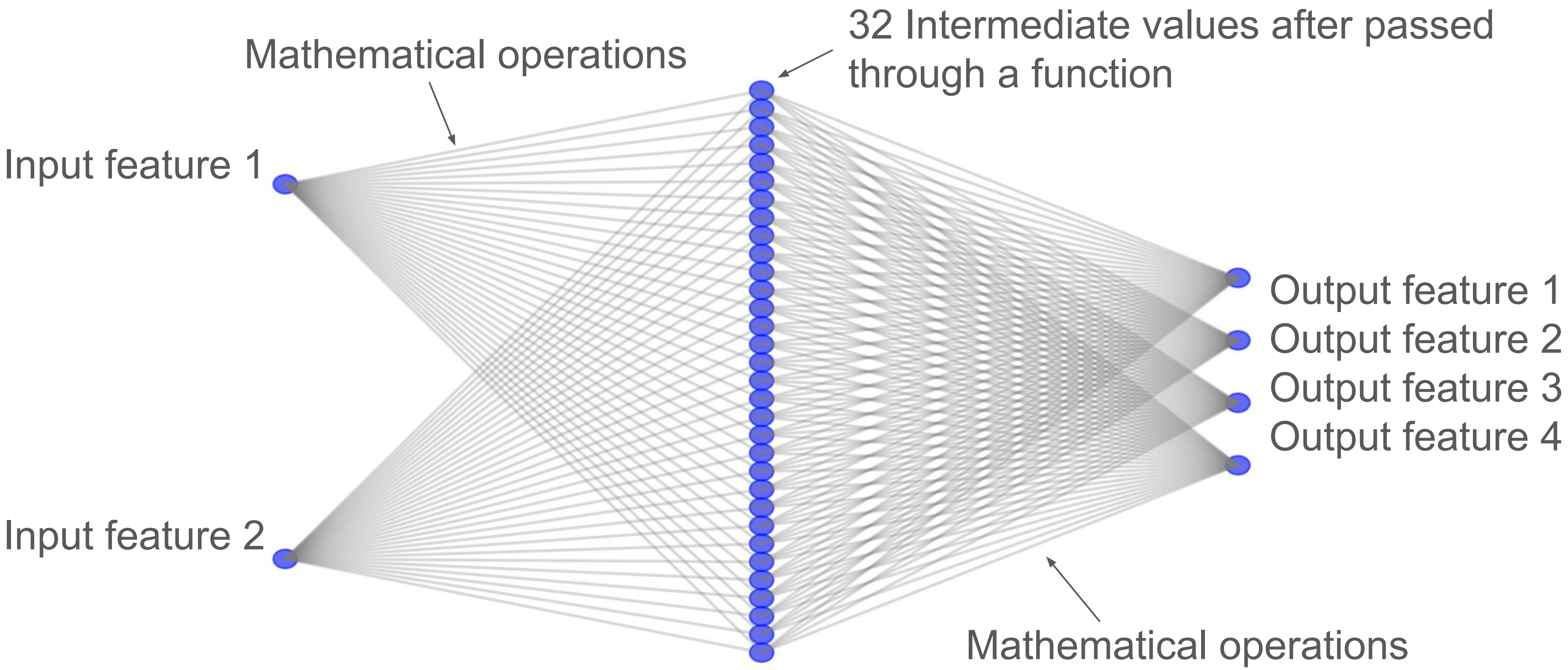
Visualization of Neural Network Architecture



Visualization of Neural Network Architecture

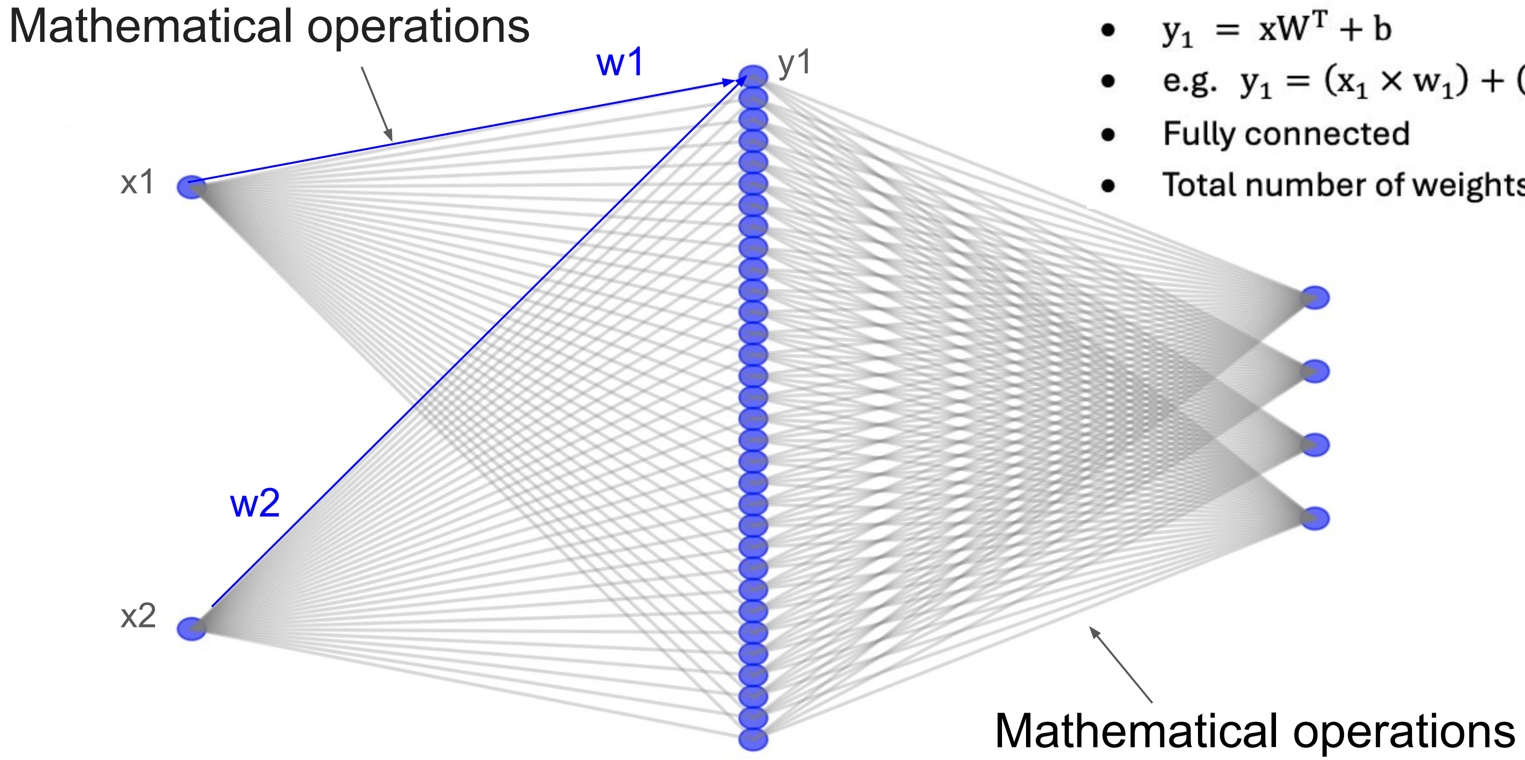


Visualization of Neural Network Architecture



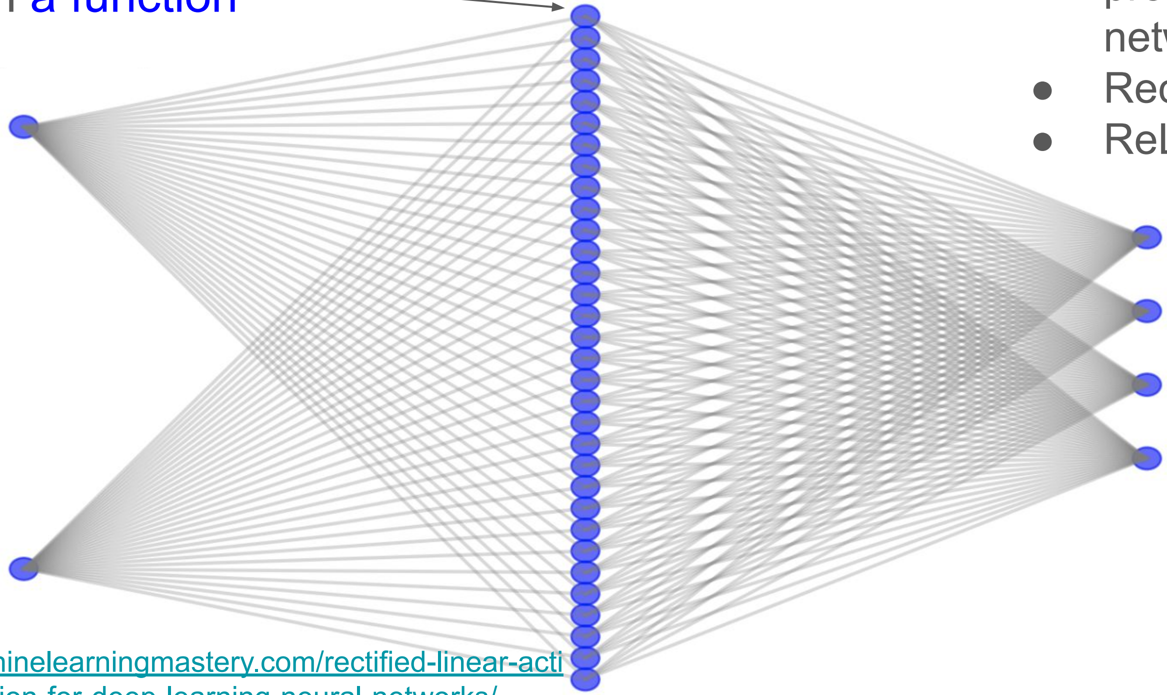
Linear layer

- 2 → 32 → 2
- $y_1 = xW^T + b$
- e.g. $y_1 = (x_1 \times w_1) + (x_2 \times w_2) + b$
- Fully connected
- Total number of weights = (32x2) + (32x4) = 192



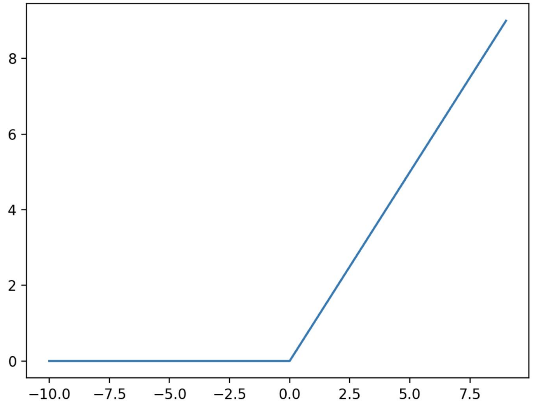
ReLU: An activation function

32 Intermediate values after passed through a function



https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

- An activation function that provides nonlinearity to neural networks.
- Rectified Linear Unit
- ReLU(x)=max(0,x)



What is training?

Training is an iterative process to find the optimized values for the weights and biases that make up the network.

A process of training 1: Forward pass

- Passing the input data through the neural network
- We don’t know what the values of the weights and biases should be, so the network only has randomly assigned values at the beginning stage.
- The results after the first forward pass must be a lot differ from the actual output values.

A process of training 2: MSE-based Loss Calculation

MSE calculates a cost per epoch/batch:

the average of the squared differences between the actual output values from dataset and the model's predicted values via forward pass.

$$\text{Mean Squared Error (MSE)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- N is the number of data points,
- y_i is the actual output value in the dataset,
- $\hat{y}_i$ is the predicted value from the model.

A process of training 3: Backpropagation to optimize params in NN

- Parameters in NN: weights, biases
- Backpropagation (of error) , chain rule
- Gradient descent
- How do you know when the training is finished?

Regression model

- `$ git checkout main`
- open Jupyter Notebook

Controlling the generative model

- See how the regression model is used in the app to control the generative model
- Play time!
    - Try training different interactions

Recap

- We have embedded a model which applies a style transfer effect to the audio sample loaded in our instrument.
- We have interacted with the generative model by exposing its latent space (compressed and organised representation of the training data) and navigating it with the sliders.
  - The values of the four sliders form a point in the 4D latent space, eg. (-1, 0, 1.5, 2)
- We have defined how to generate custom gestural interactions with the generative model
  - Adding the possibility to create a training dataset which translates a mouse XY position (2 values) to a point in the latent space dimension (4 values)
  - Training a regression model on the dataset
  - Using the regression model to translate our position on the XY pad to a point in the latent space

Thank you