Image generation

AI for Media, Art & Design, Spring 2024

Prof. Perttu Hämäläinen
Aalto University

Text to Image in 5 minutes: Parti, Dall-E 2, Imagen

https://www.youtube.com/watch?v=GYyP7Ova8KA

Graphics in 5 Minutes
17,3 t. tilaajaa

Tilaa

363 | Jaa | Lataa | Klippi | Tallenna

Text to Image: Part 2 -- how image diffusion works in 5 minutes

https://www.youtube.com/watch?v=lyodbLwb2lY

Graphics in 5 Minutes
17,3 t. tilaajaa

Tilaa

345 | Jaa | Lataa | Klippi | Tallenna

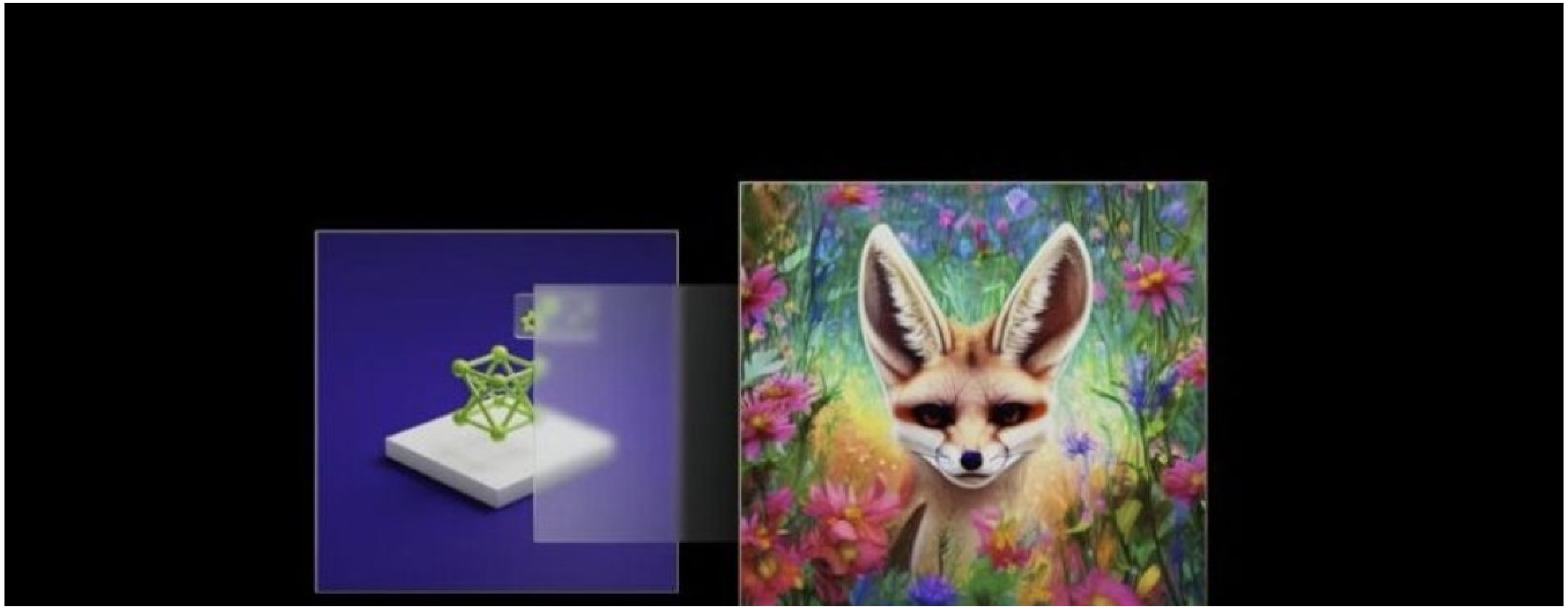
NVIDIA. DEVELOPER Home Blog Forums Docs Downloads Training

Technical Blog Search blog Filter

Generative AI Research Spotlight: Demystifying Diffusion-Based Models

Dec 14, 2023 +22 Like Discuss (0)

By Miika Aittala



https://developer.nvidia.com/blog/generative-ai-research-spotlight-demystifying-diffusion-based-models/

Contents

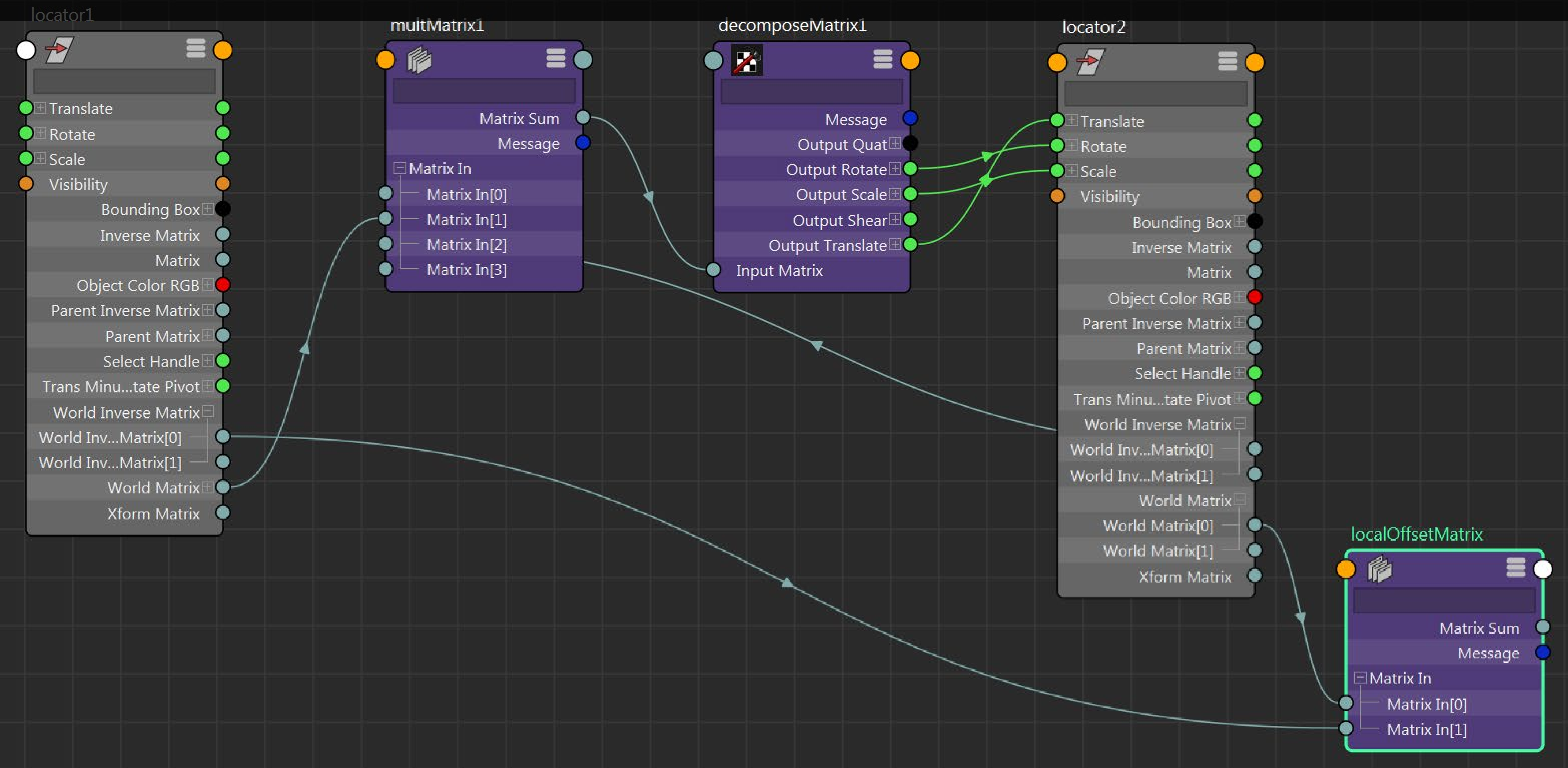
- **Preliminaries: compute graphs, artificial neurons, activation functions, loss functions**
- Understanding nonlinear activations
- Image generator architectures

Preliminaries

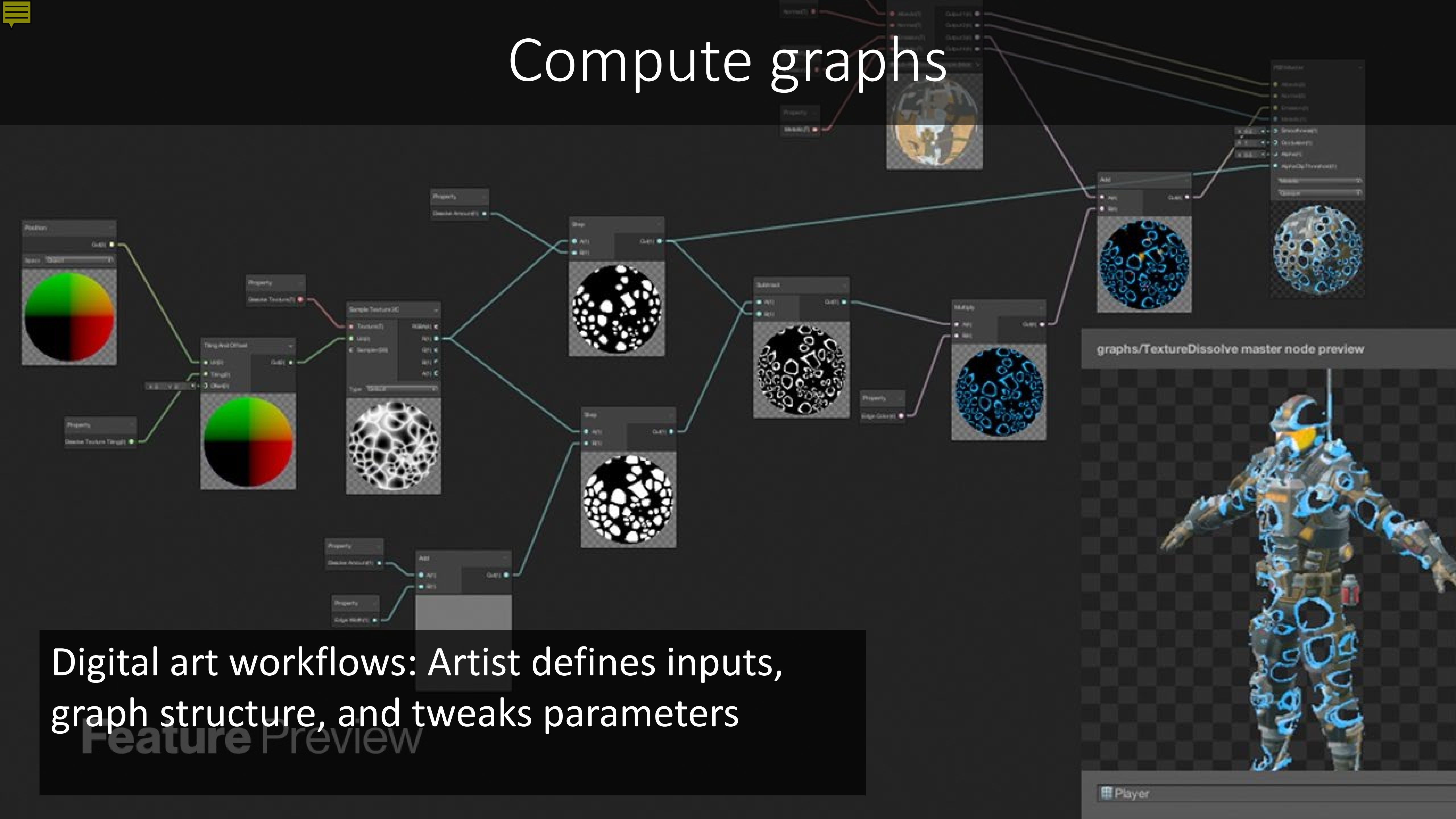
3Blue1Brown YouTube channel: Visual intuitions to math and neural networks

- https://www.youtube.com/watch?v=aircAruvnKk (what is a neural network)
- https://www.youtube.com/watch?v=IHZwWFHWa-w (how neural networks learn, i.e., gradient descend)
- https://www.youtube.com/watch?v=Ilg3gGewQ5U (what is backpropagation really doing)
- Essence of Linear Algebra (vectors, matrices, determinants etc., the branch of math at the heart of it all): https://www.youtube.com/watch?v=kjBOesZCoqc&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab

Compute graphs

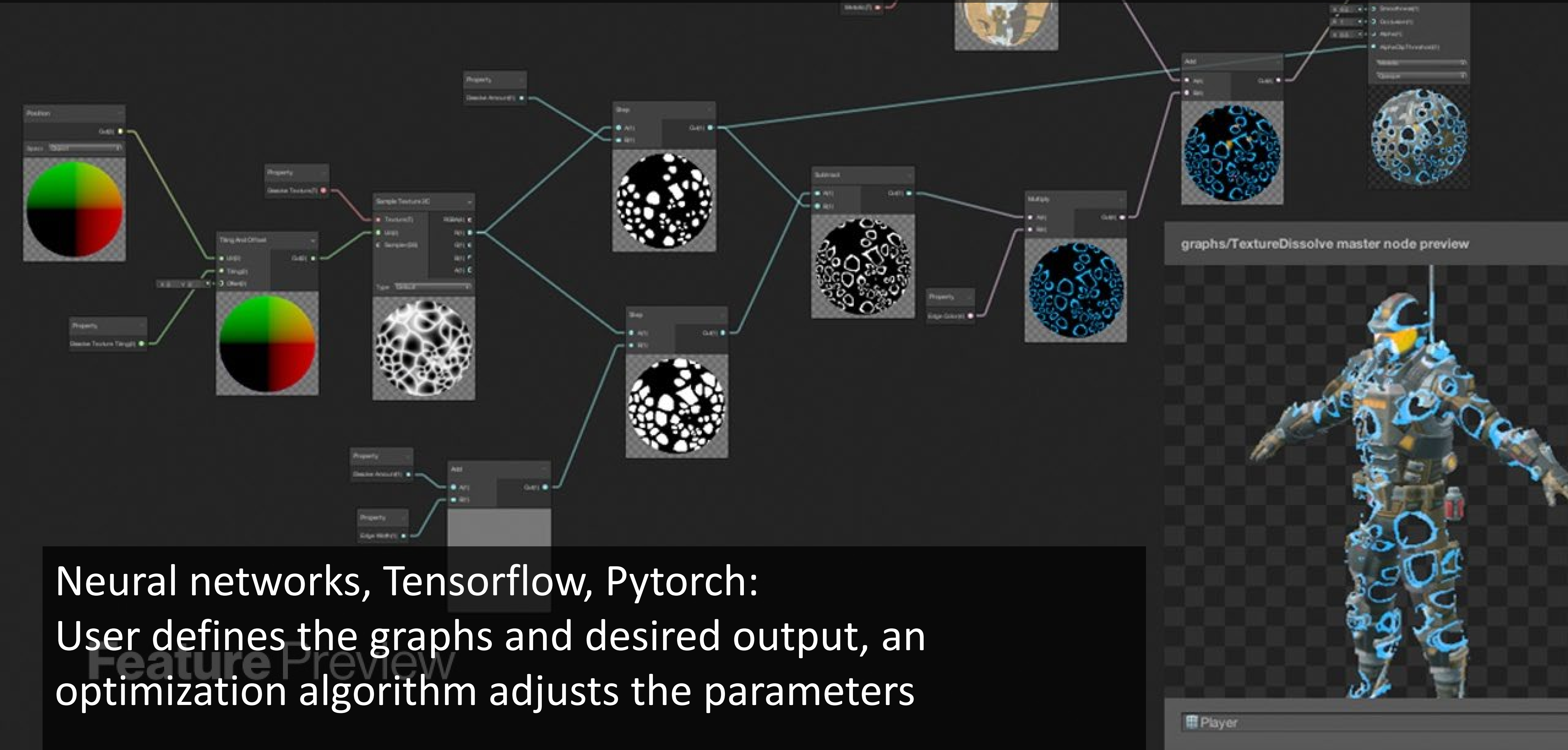


Compute graphs



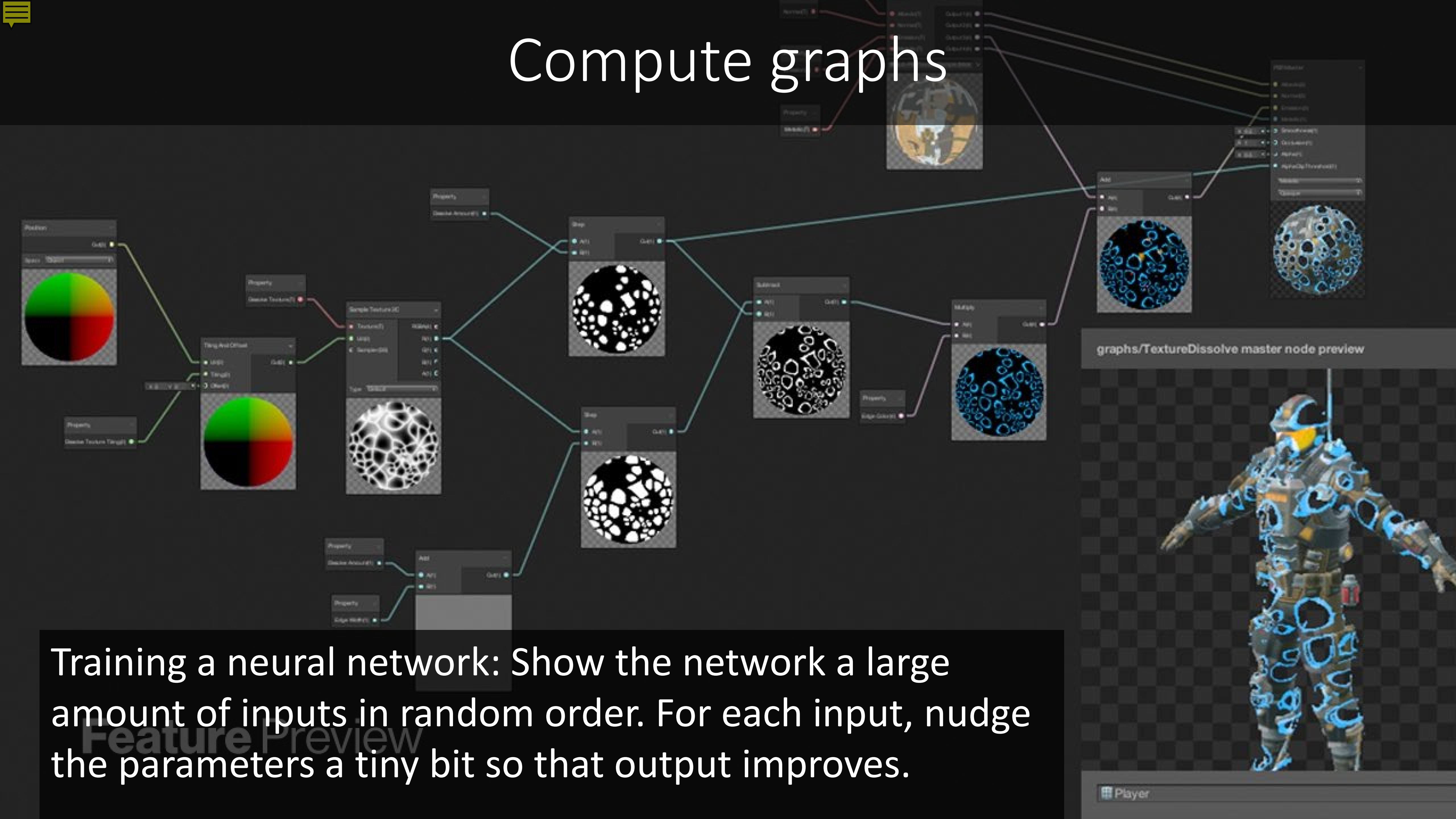
Digital art workflows: Artist defines inputs, graph structure, and tweaks parameters

Compute graphs



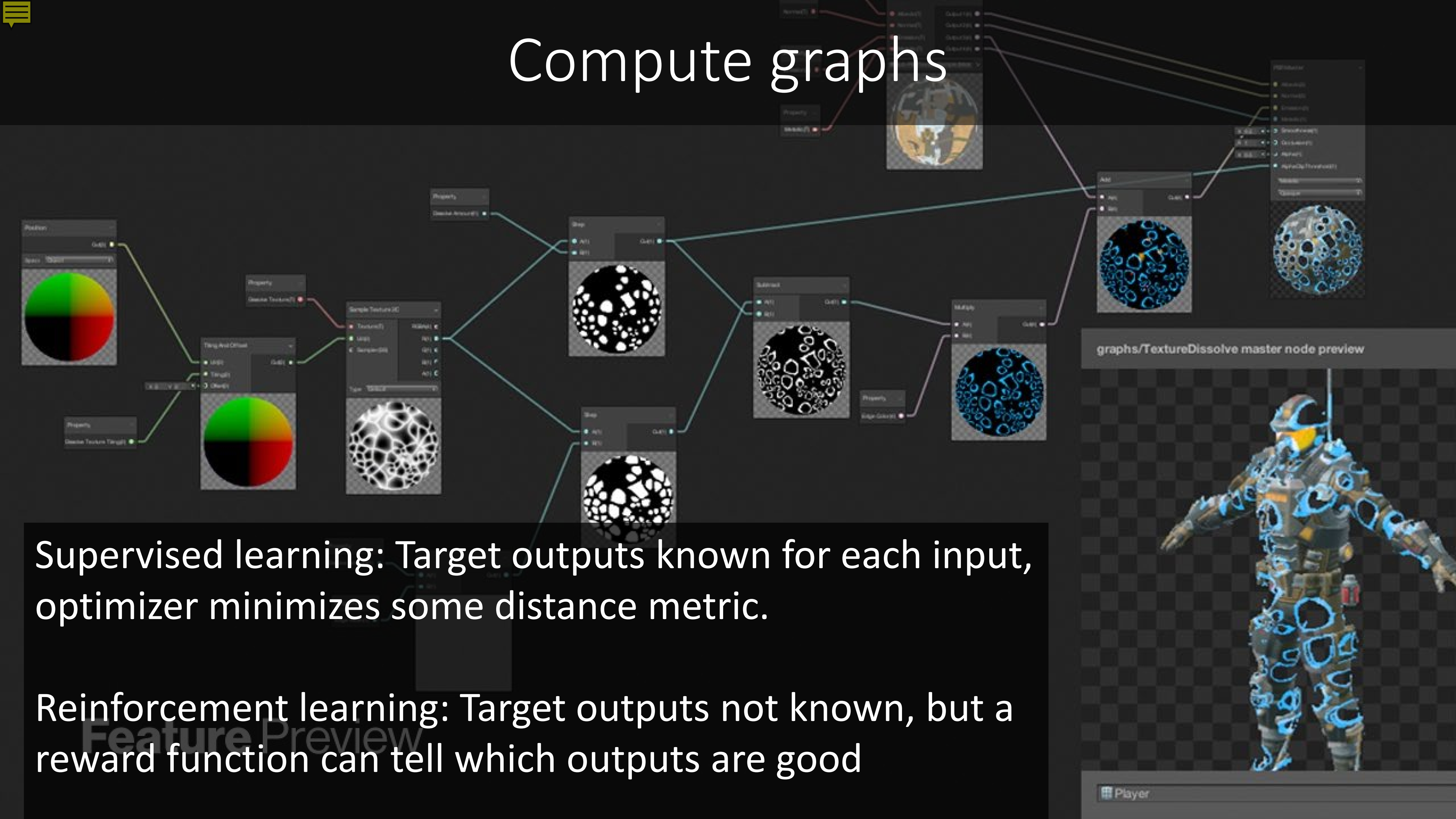
Neural networks, Tensorflow, Pytorch:
User defines the graphs and desired output, an optimization algorithm adjusts the parameters

Compute graphs



Training a neural network: Show the network a large amount of inputs in random order. For each input, nudge the parameters a tiny bit so that output improves.

Compute graphs

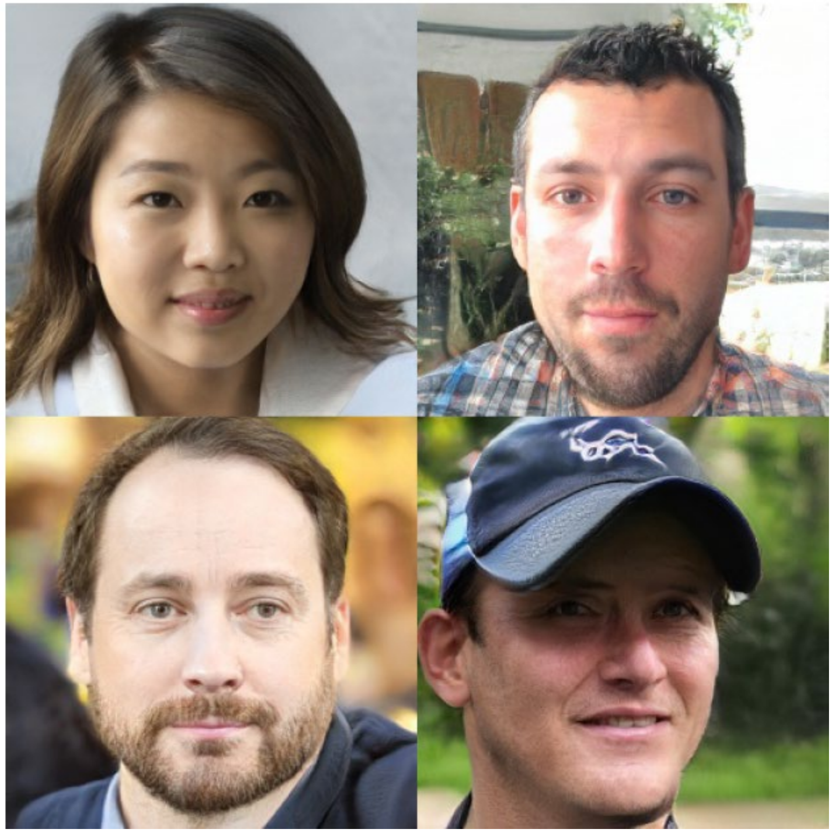


Supervised learning: Target outputs known for each input, optimizer minimizes some distance metric.

Reinforcement learning: Target outputs not known, but a reward function can tell which outputs are good

Working & thinking with compute graphs

- Case study: StyleGAN 2 “circuit bending”



NVIDIA’s pretrained StyleGAN2-ada neural network available through Github

The same network “broken” with unrelated images of rocks, trees, foliage

Working & thinking with compute graphs

- A generator network takes in "latent space" vectors, i.e., vectors of hidden variables that explain the results (face gender, age, rotation…)
- Training the network = based on training images, infer an approximation of the latent variables and the generative process

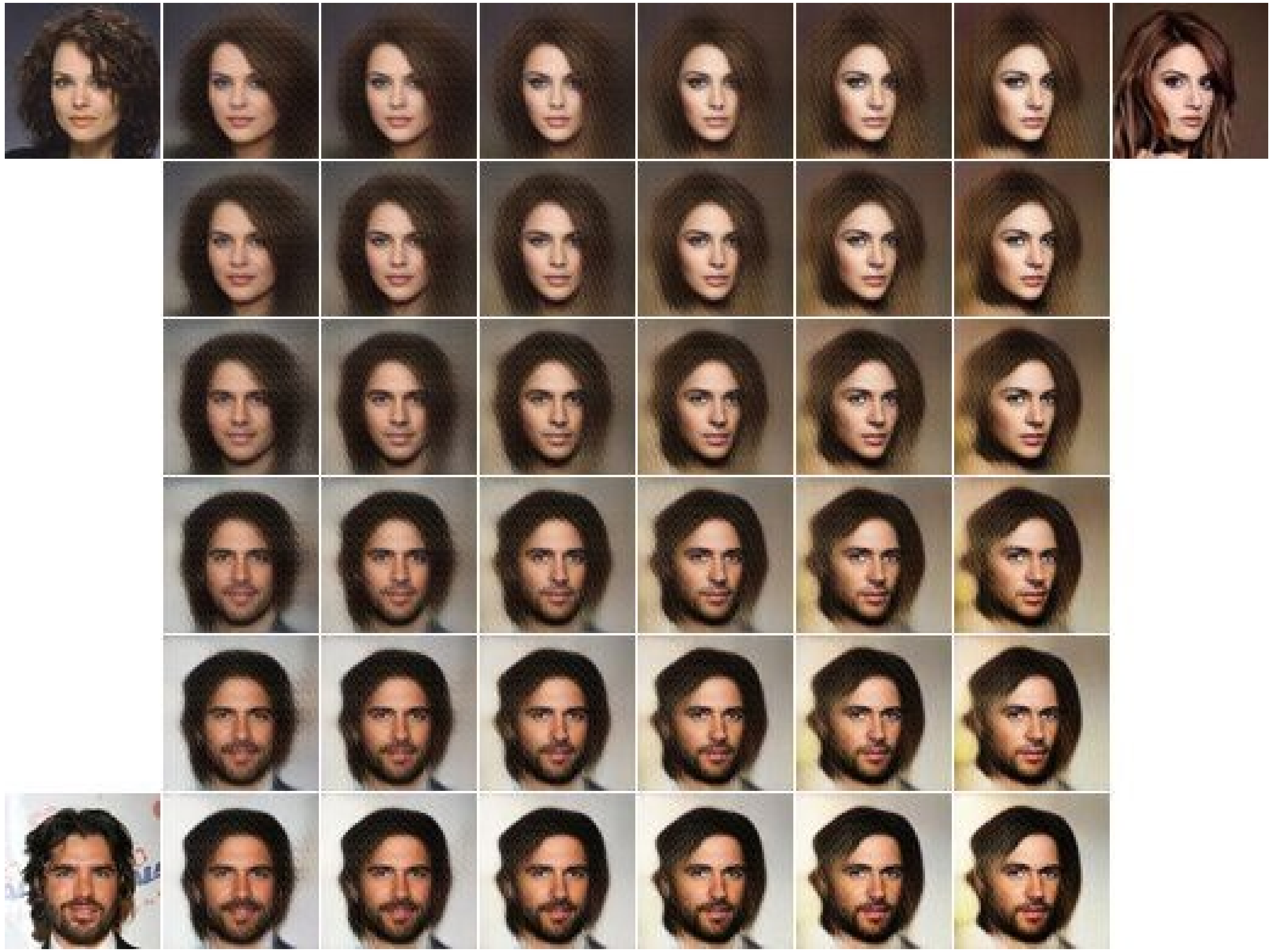
Output image: this is just an array of numbers (pixel RGB values)

Latent space vector

$\mathbf{x} = [x_1, x_2, x_3 \ldots]$ ⇨ Generator network ⇨

$$\begin{array}{l} [[r,g,b,\ldots] \\ \;[r,g,b,\ldots] \\ \quad\quad \vdots \\ \quad\quad \vdots \\ \quad\quad\quad\quad] \end{array} =$$

A 2D latent space for generating faces



https://medium.com/@juliendespois/latent-space-visualization-deep-learning-bits-2-bd09a46920df

Everything is just numbers

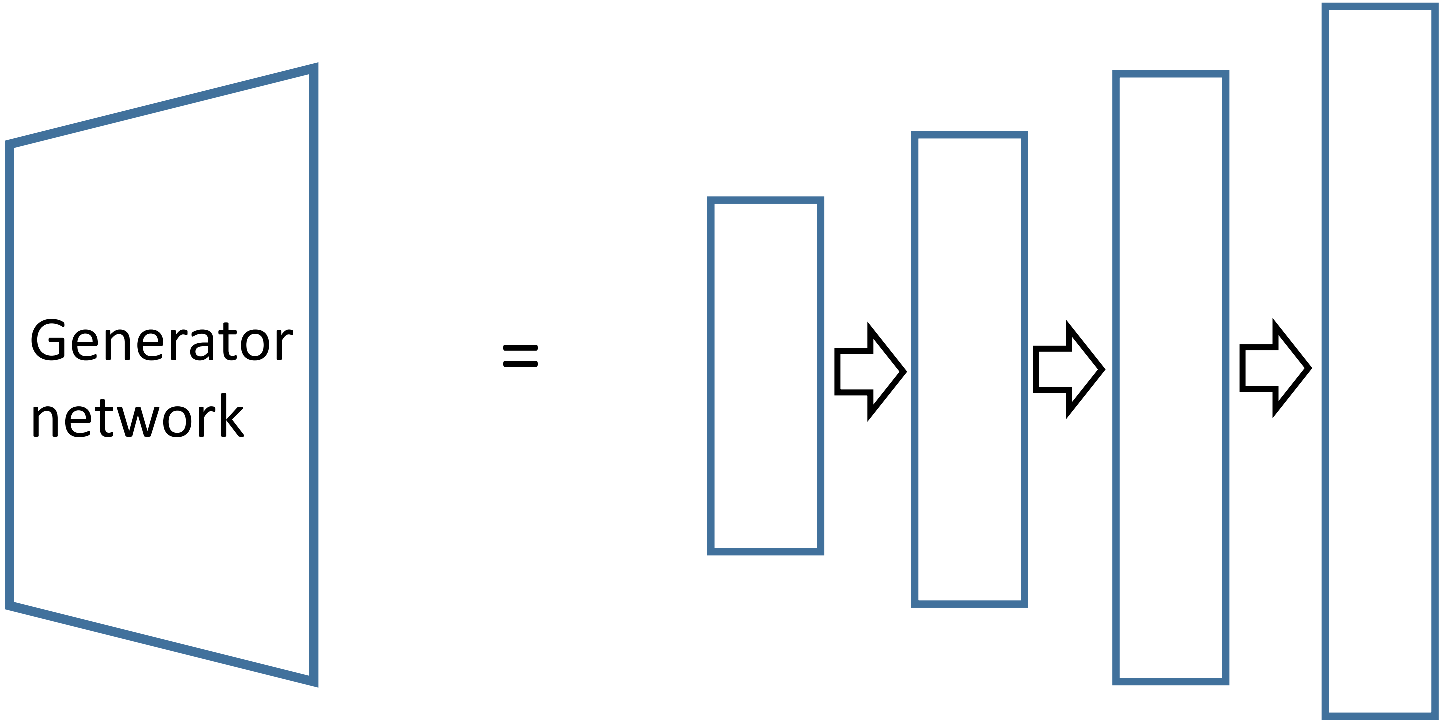
- 1-dimensional arrays: Text (each number corresponding to a character), 1-channel audio
- 2-dimensional arrays: grayscale images, stereo audio
- 3-dimensional arrays: color images (array shape [width,height,channels]), multiple stereo audio files
- 4-dimensional arrays: multiple color images, [index,width,height,channels]

…

The compute graphs usually don’t care about the actual data types. When working with someone else’s code, the first thing is to usually check how data is formatted into arrays, and convert one’s own data accordingly, if needed.

Working & thinking with compute graphs

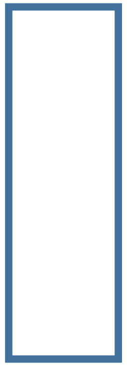
- Usually, a neural network can be broken down into layers
- What data moves between layers and how is defined by the network designer
- This is like an electronics device consisting of a few integrated circuits like a microcontroller and a Bluetooth chip.



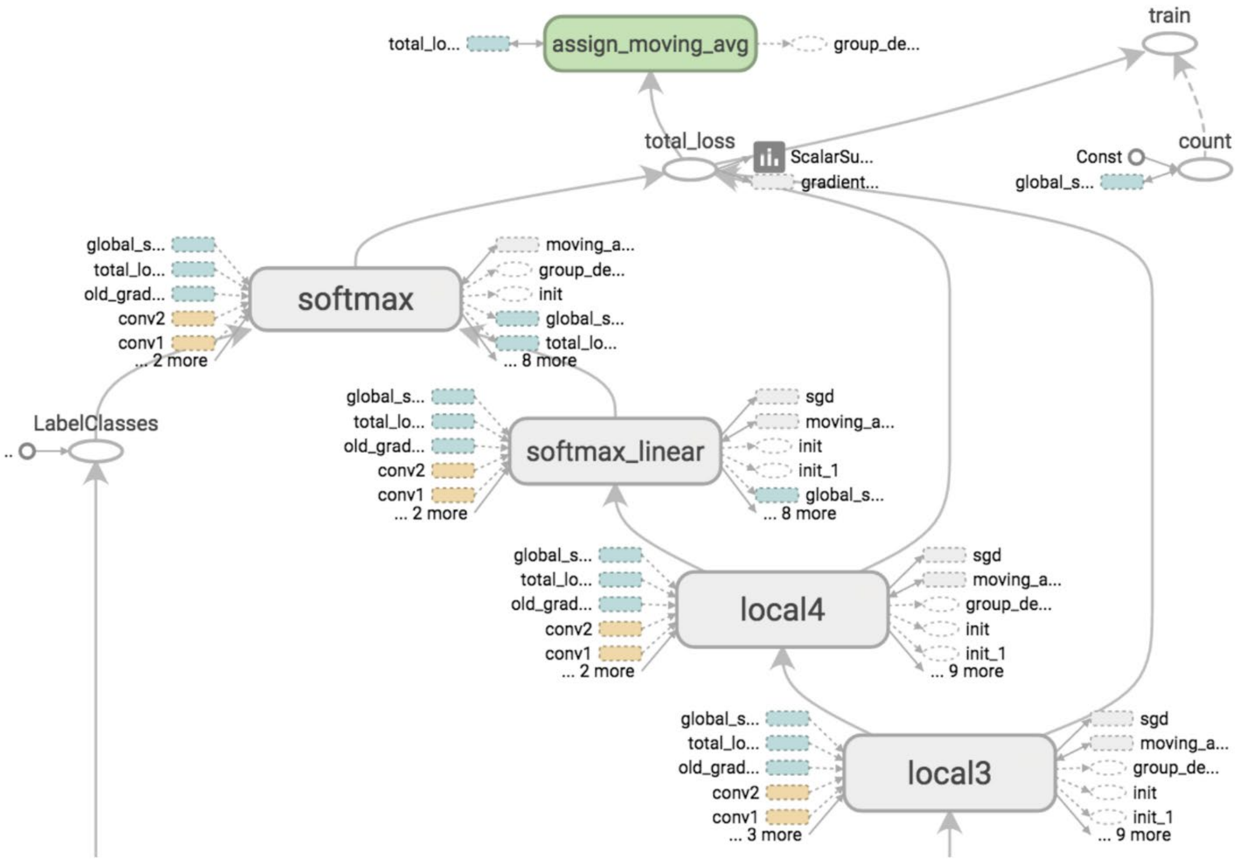
Working & thinking with compute graphs

- Layers can be further broken down into individual neurons and math operations
- This is like the Bluetooth chip consisting of individual transistors

Network layer



=



Working & thinking with compute graphs

- Usually, no need to work on the level of individual neurons, and only rarely on the level of layers
- Rather: Connecting readymade networks, like connecting an Arduino board to a movement tracking sensor
- Main questions: Which building blocks to use, where to get data, how to format it correctly?

Code written by artist/designer

Cloned from Github

Collect data (e.g., scrape from some website) ⇨ Read data & convert to the format expected by the neural network or its training code ⇨ Neural network(s) (as is, or trained with your data) ⇨ Interesting output

Working & thinking with compute graphs

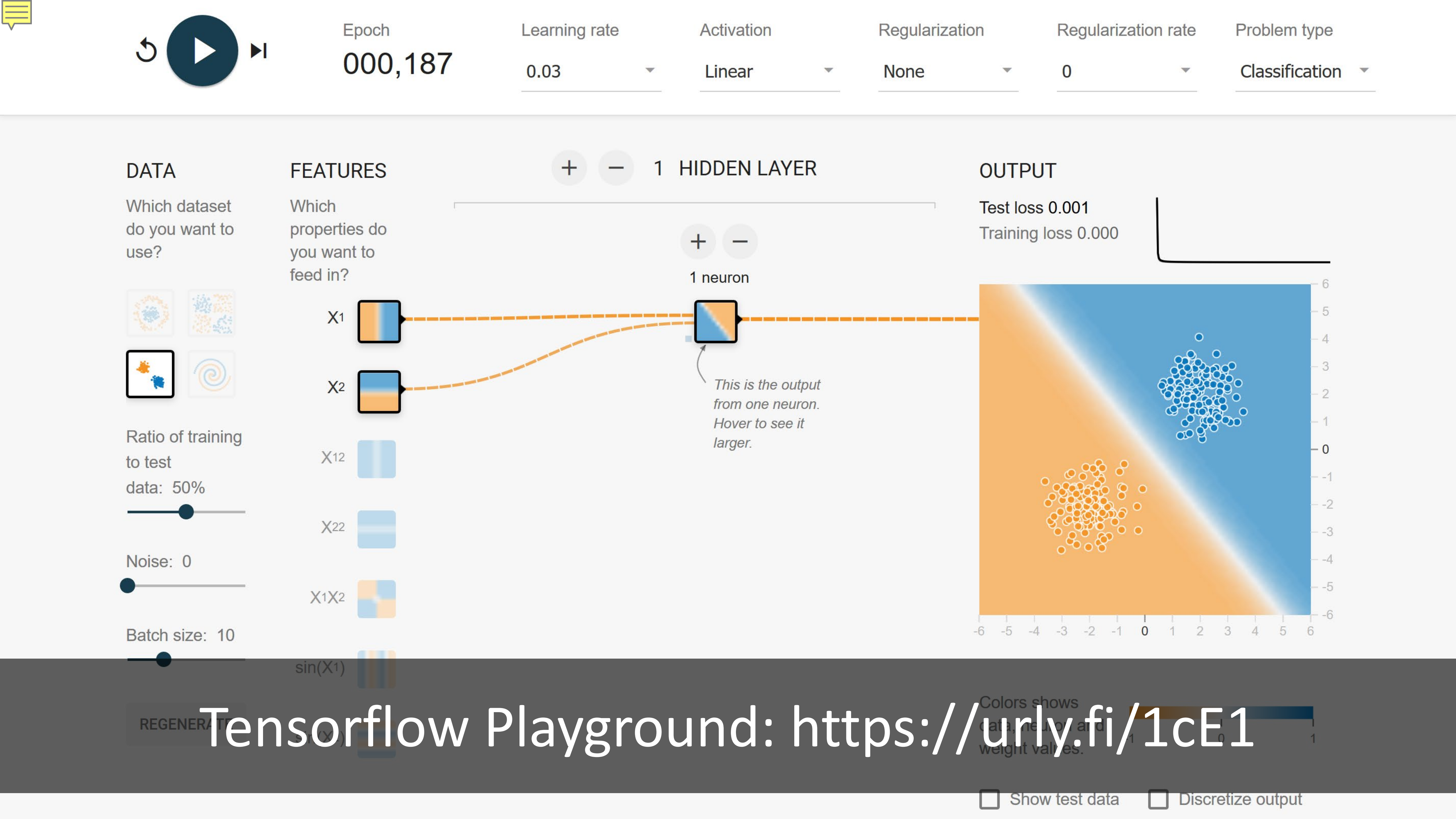
These results:

- A pretrained (readymade) StyleGAN2-ada face generator from NVIDIA's Github
- Custom Python code: Scrape images of rocks, foliage etc. using Google Image search API. Package the images to a .zip file expected by StyleGAN (as explained on the Github page)
- Training the network just the right amount of time, using Python command line tools provided by NVIDIA

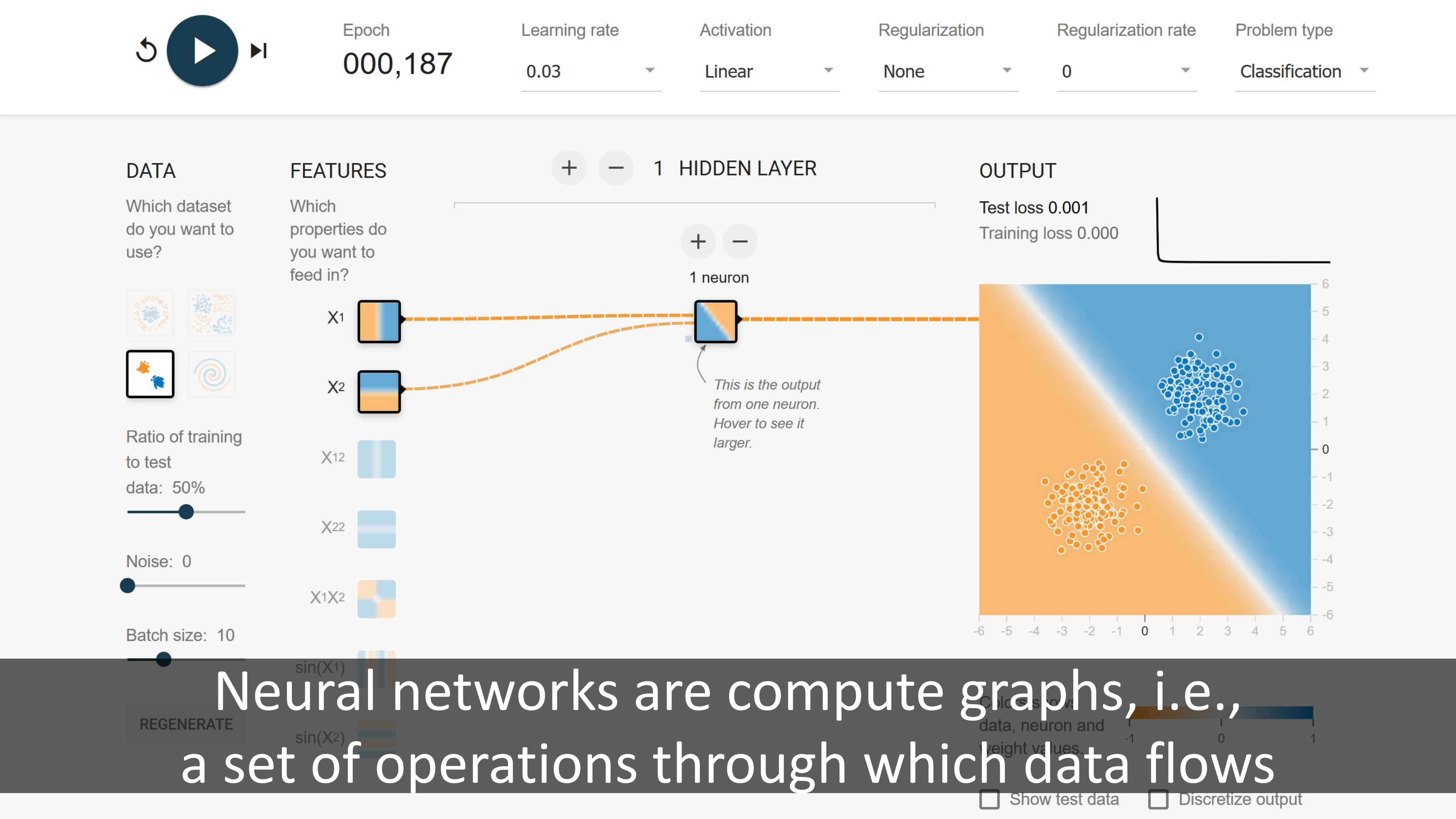
Working & thinking with compute graphs

Typical skills needed:

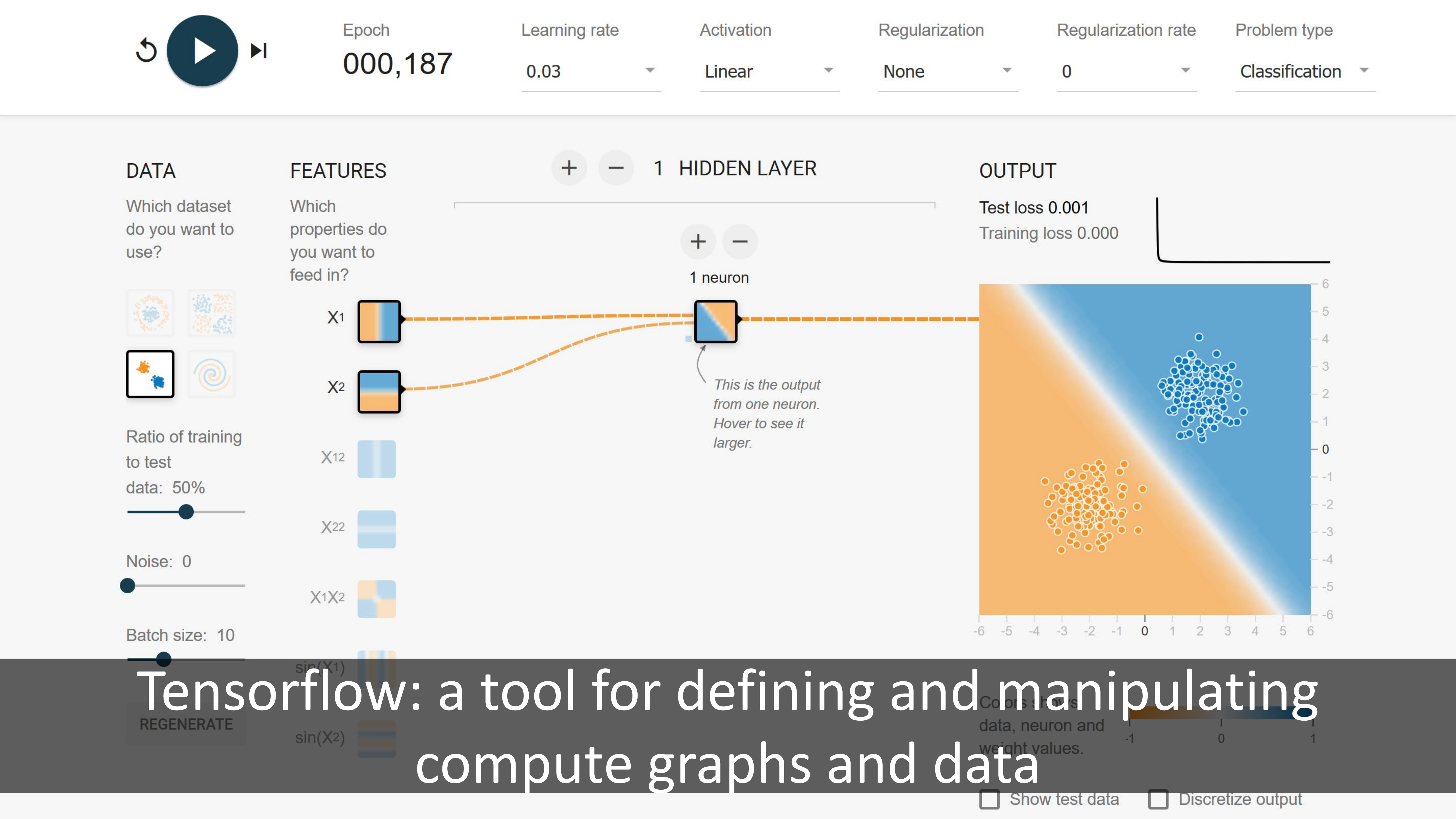
- Python network and file I/O
- Working with multidimensional number arrays using Python's Numpy package (basically all ML & AI builds on Numpy)
- Understanding what kinds of neural networks there are, what they can do, and what kind of data goes in and out



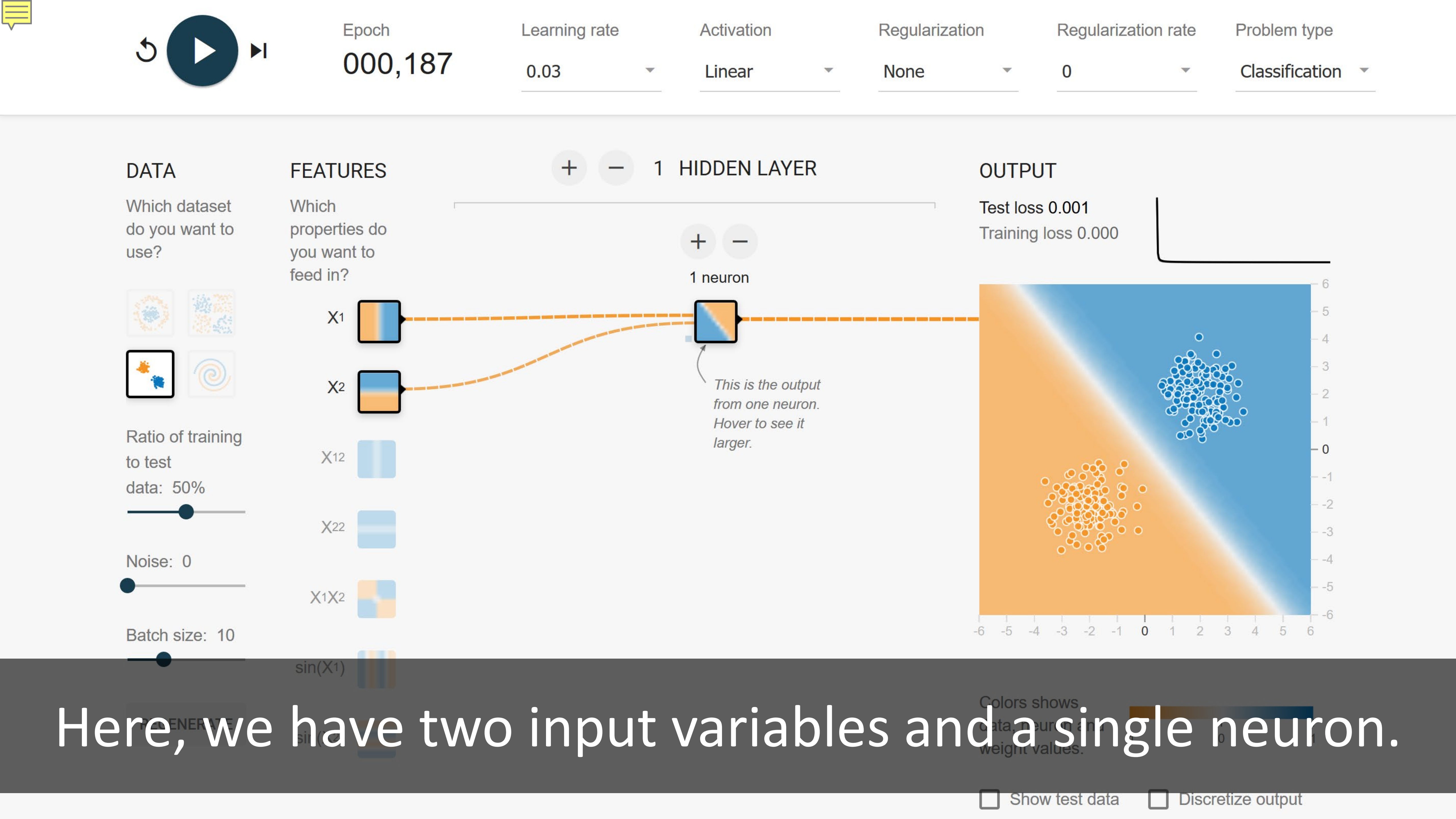
Tensorflow Playground: https://urly.fi/1cE1



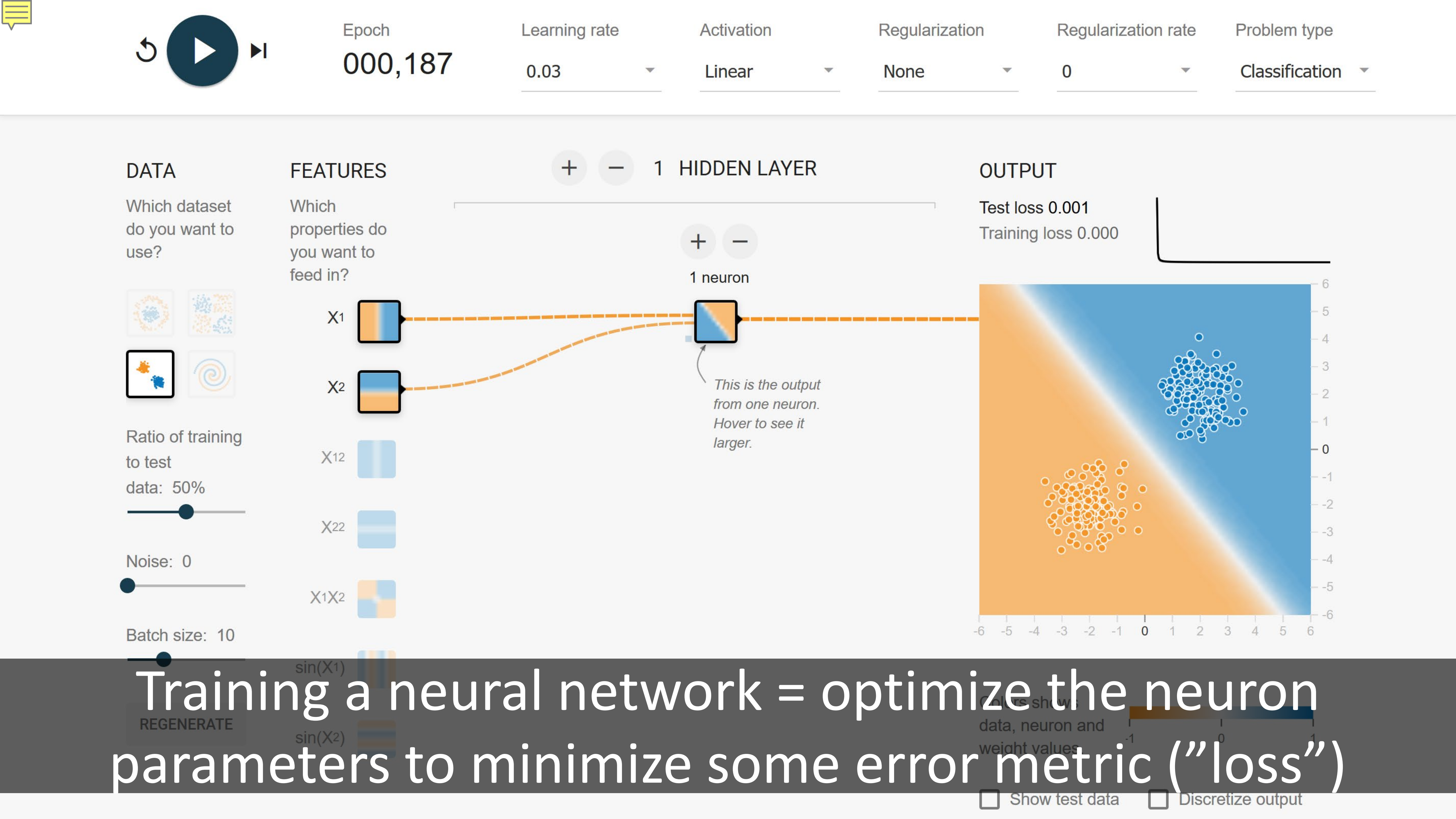
Neural networks are compute graphs, i.e.,
a set of operations through which data flows



Tensorflow: a tool for defining and manipulating compute graphs and data



Here, we have two input variables and a single neuron.

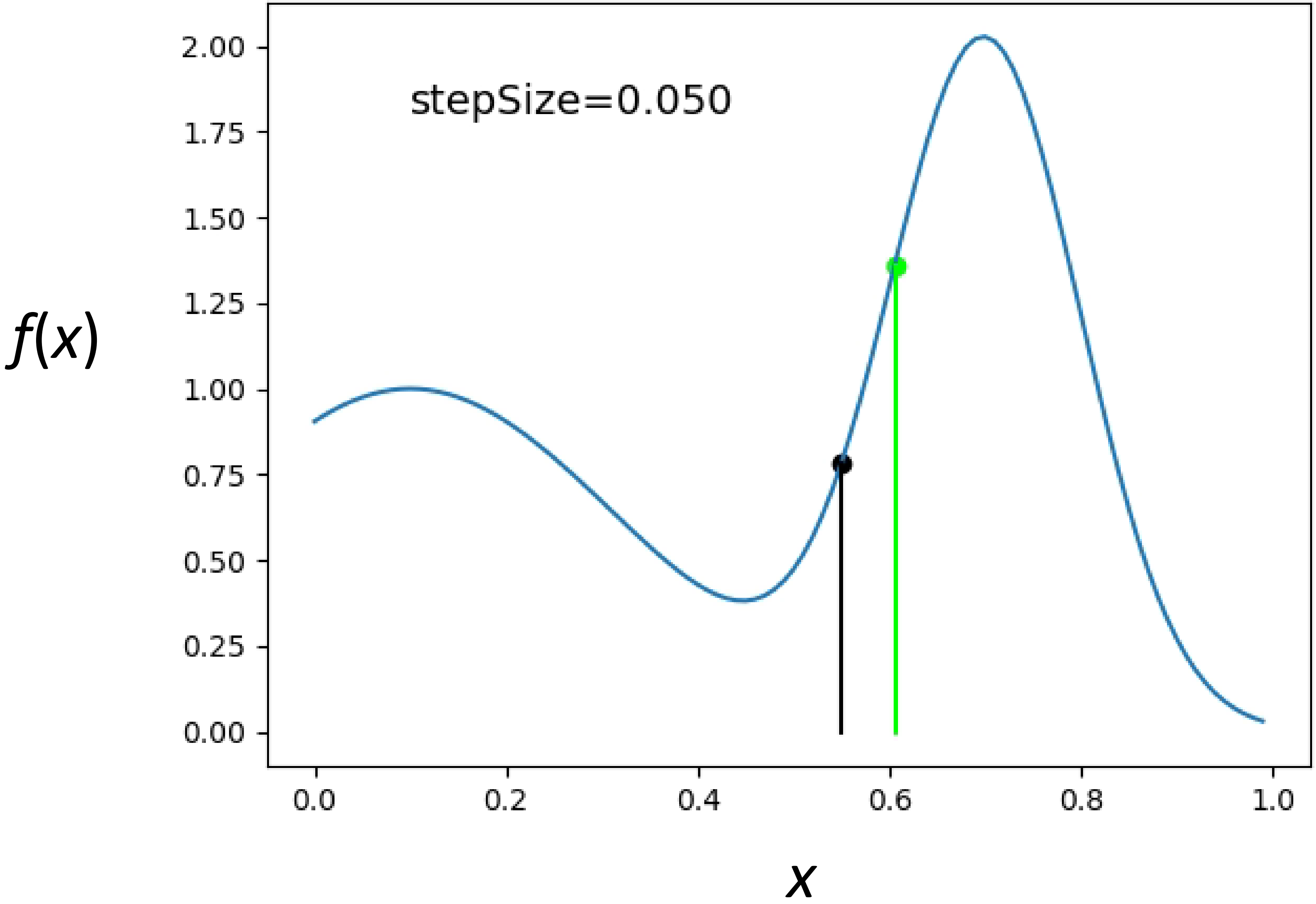


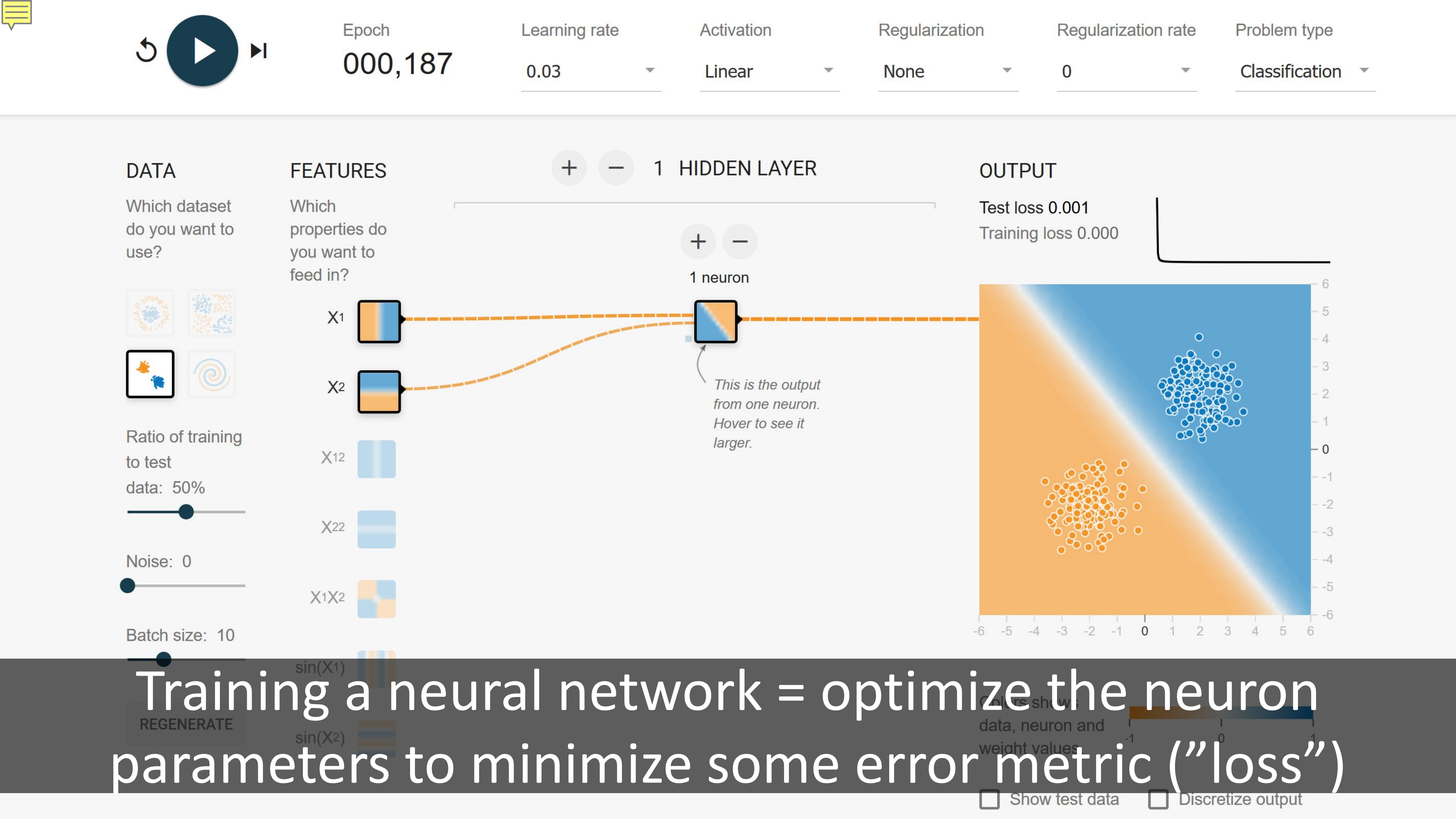
Training a neural network = optimize the neuron parameters to minimize some error metric ("loss")

All AI is (mathematical) optimization

- Adjust some **x** to minimize or maximize some *f*(**x**)
- Usually, one denotes vectors with boldface and scalars with italic, i.e., $\mathbf{x}=[x_1,x_2,...]$
- Neural network training: **x** denotes network parameters, *f*(**x**) is the loss or error function
- Pathfinding: **x** denotes path steps, *f*(**x**) is path length
- Gameplay AI: **x** denotes actions, *f*(**x**) is the utility or cost function
- Animation: **x** denotes muscle activations or other movement synthesis parameters, *f*(**x**) measures goal attainment, e.g., based on player input.
- Neural networks: optimal **x** has to be found through numerical iteration. (No closed-form solution that could be derived algebraically)

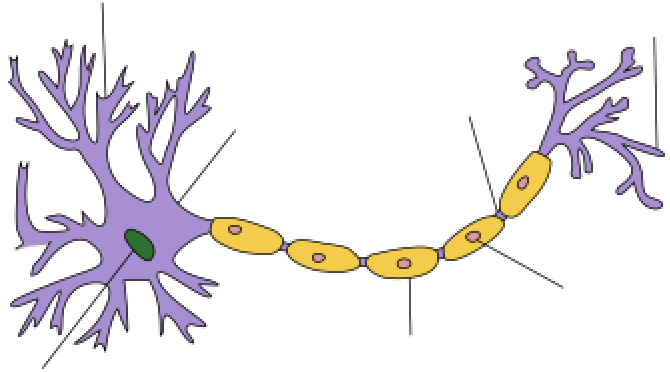
Optimization = exploring the $f(\mathbf{x})$ landscape

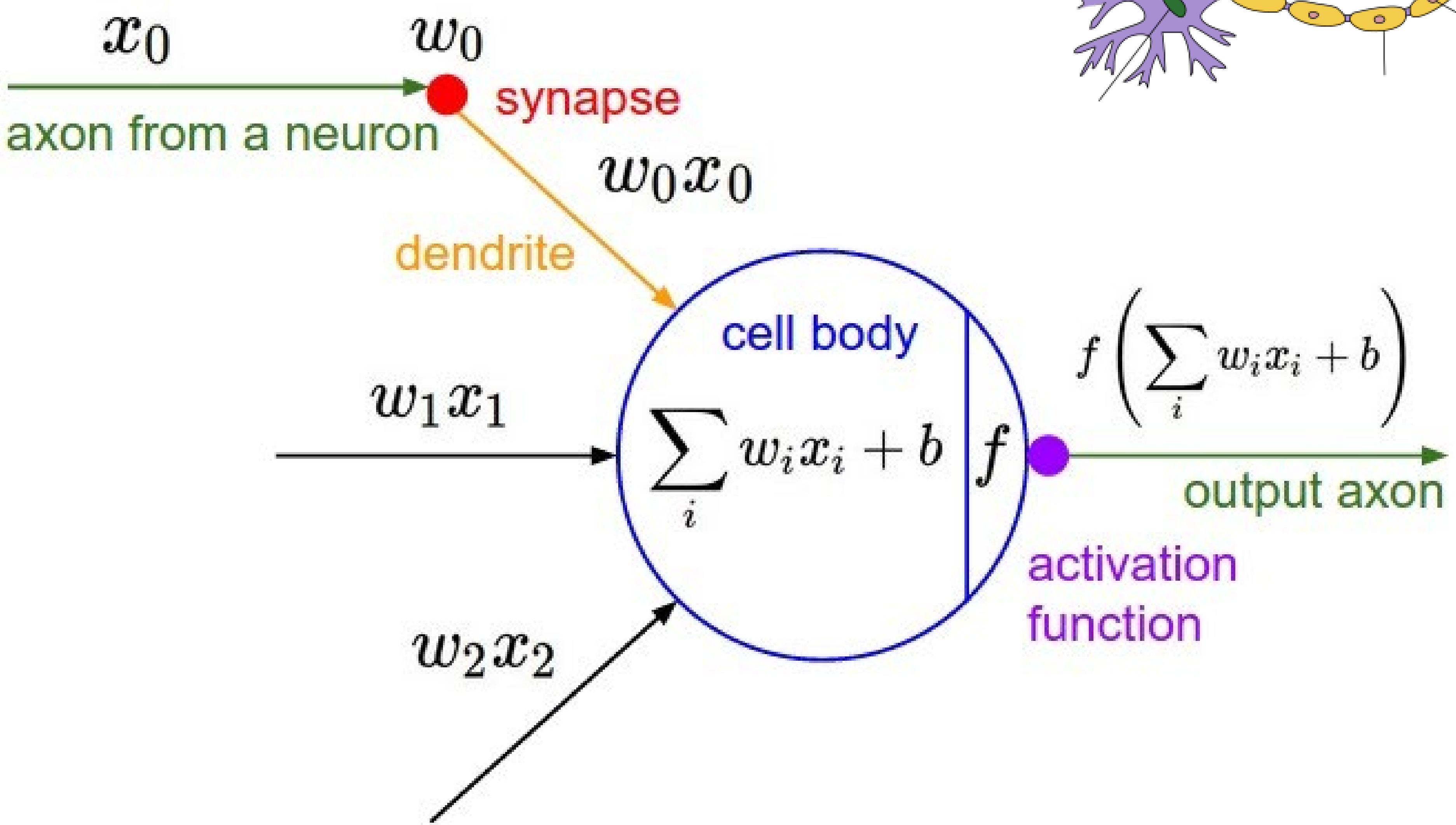




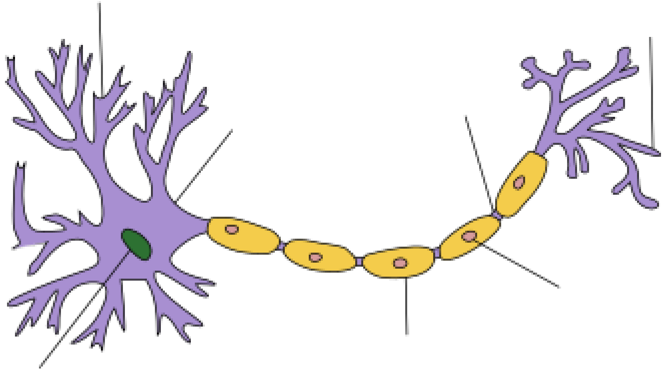
Training a neural network = optimize the neuron parameters to minimize some error metric (”loss”)

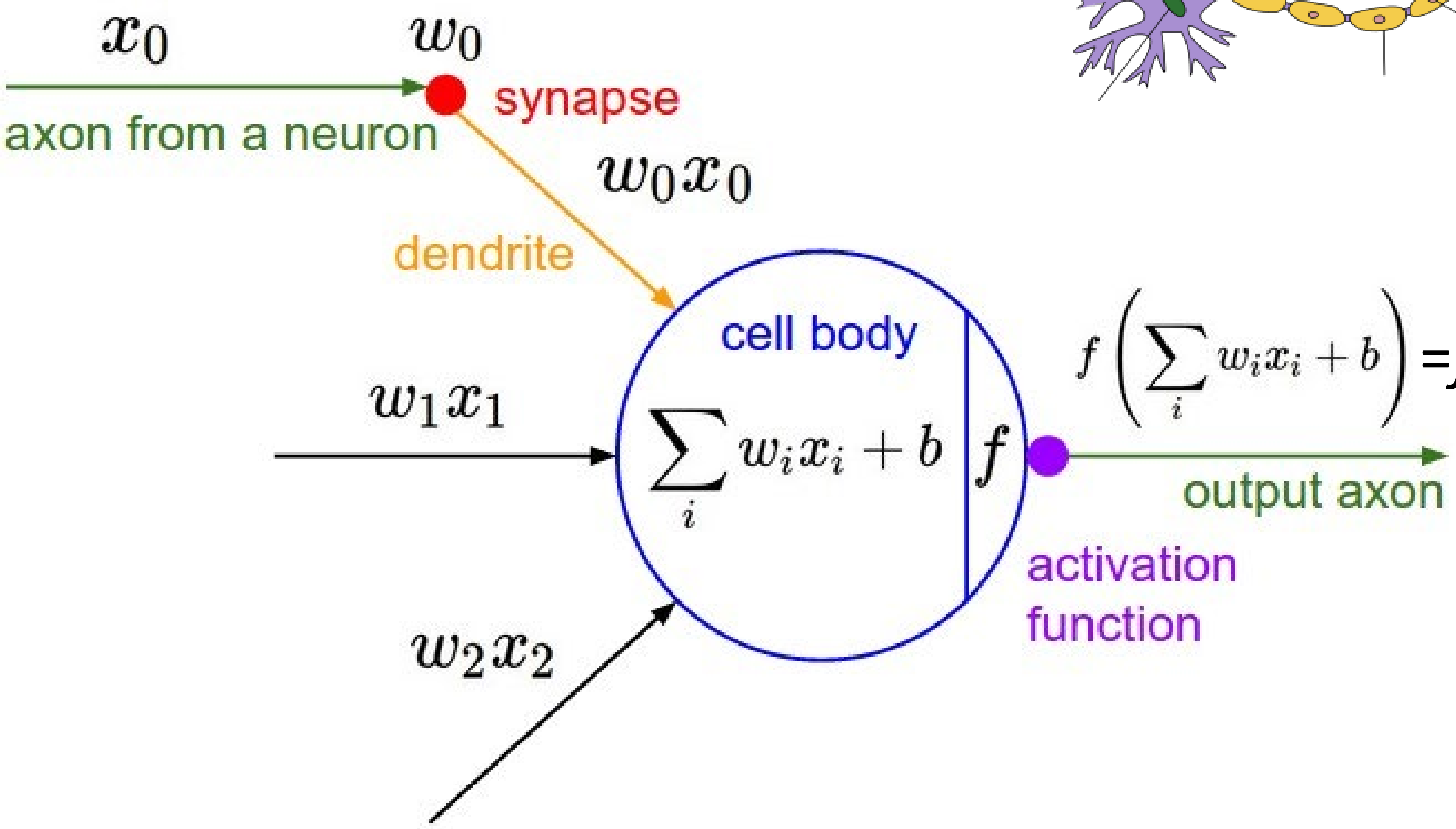
An artificial neuron





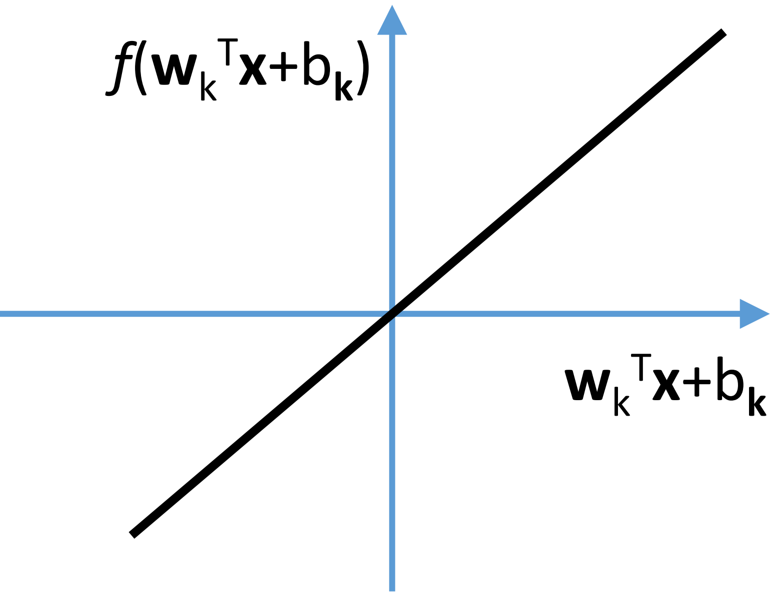
An artificial neuron



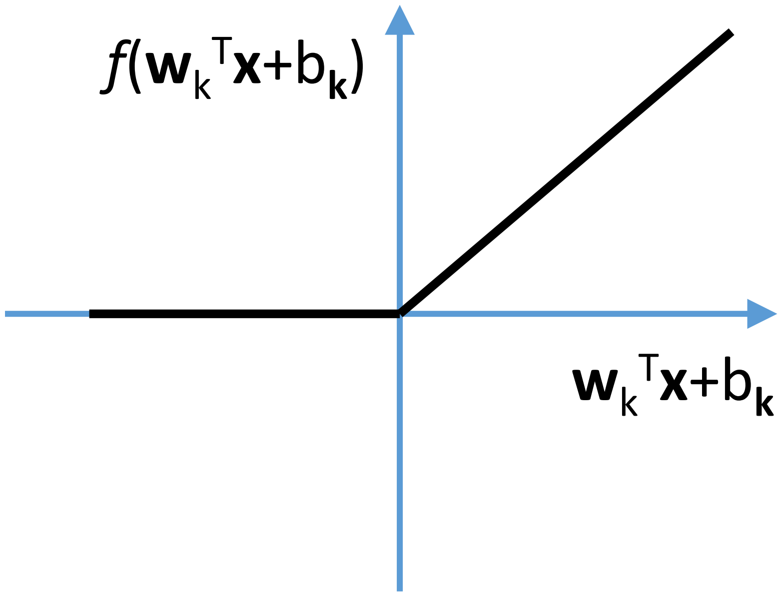


$= f(\mathbf{w}^{T}\mathbf{x}+\mathrm{b})$

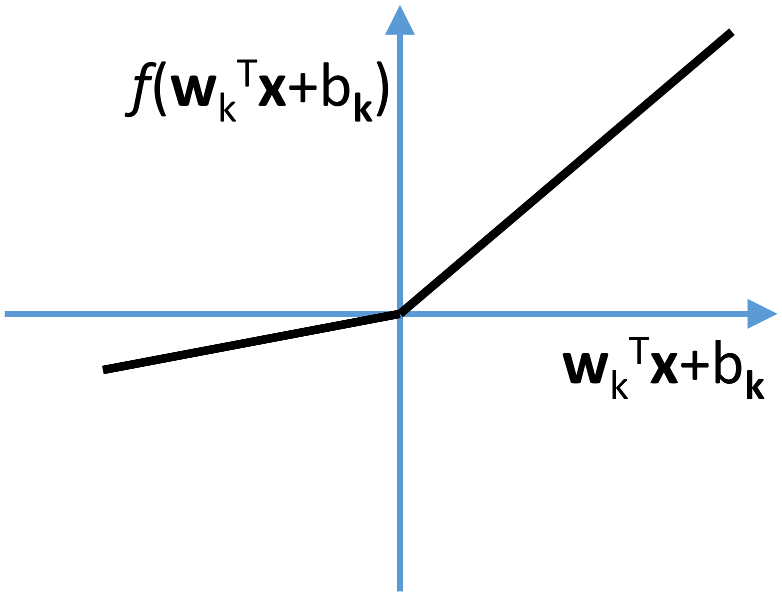
Linear activation



ReLU activation

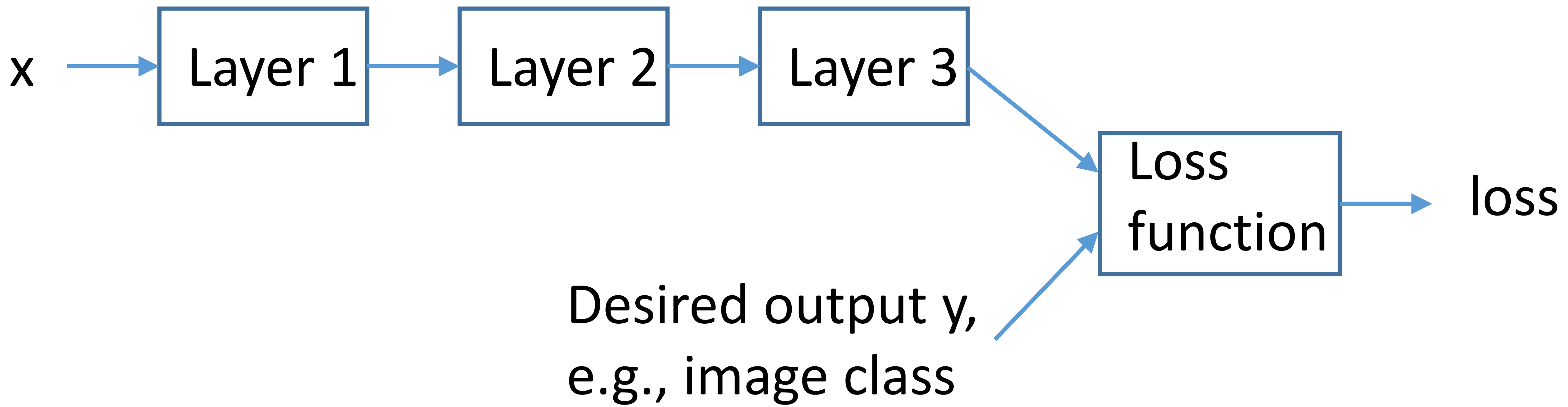


Leaky ReLU



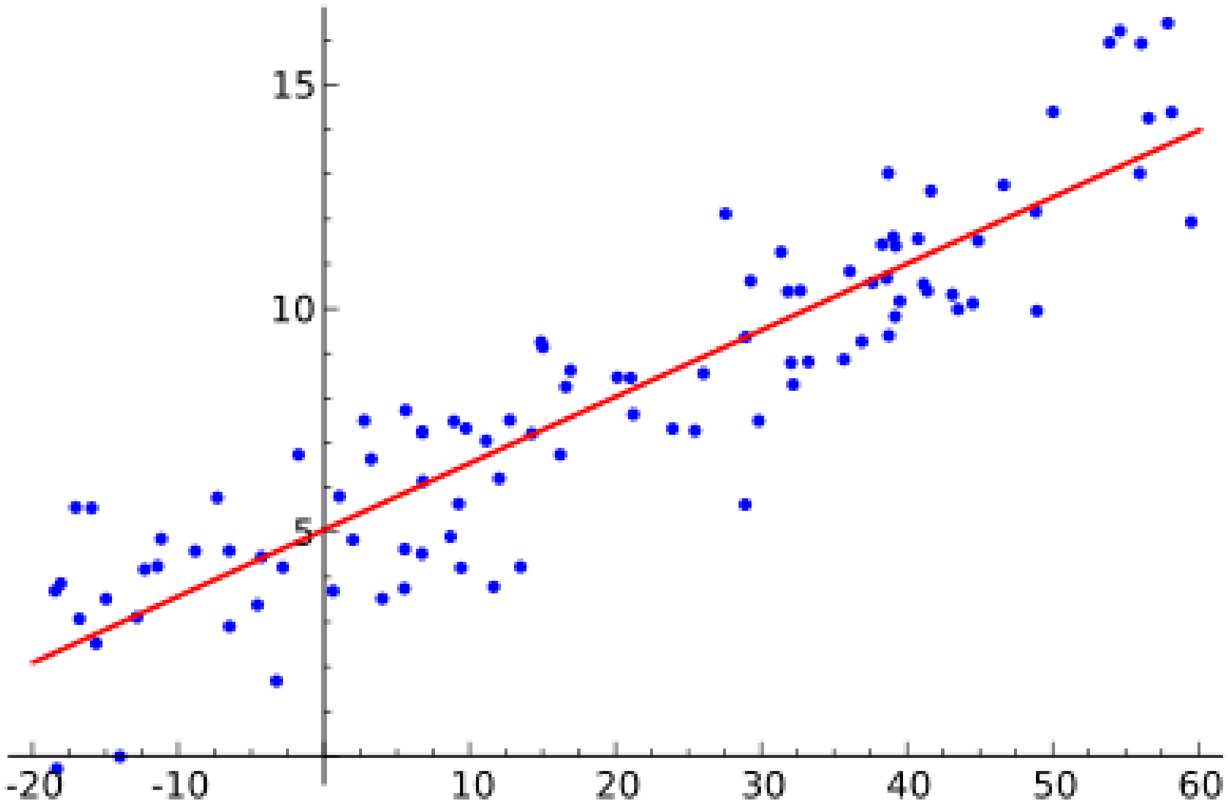
Training = optimization, minimizing loss

1. Feed a random *minibatch* of input data x through the network
2. Compute the loss function for each pair
3. Change the weights such that loss decreases, on average
4. Rinse & repeat!



Common loss functions

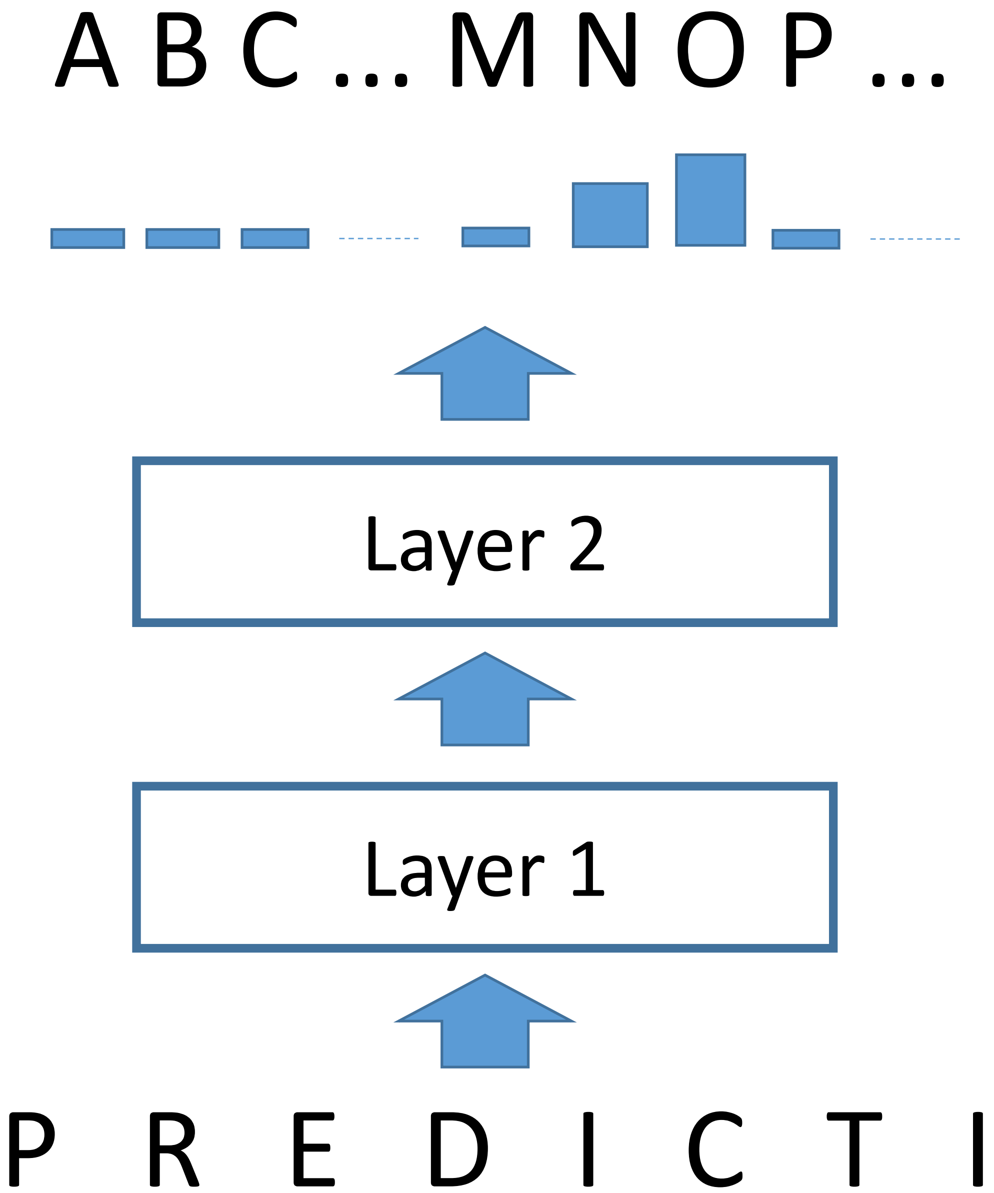
- Sum of squared errors, i.e., differences between network output variables and desired output variables



Common loss functions

- Softmax cross-entropy, in classification where one output neuron for each class

```
def cross_entropy(X,y):
    """
    X is the output from fully connected layer (num_examples x num_classes)
    y is labels (num_examples x 1)
    """
    m = y.shape[0]
    p = softmax(X)
    log_likelihood = -np.log(p[range(m),y])
    loss = np.sum(log_likelihood) / m
    return loss
```



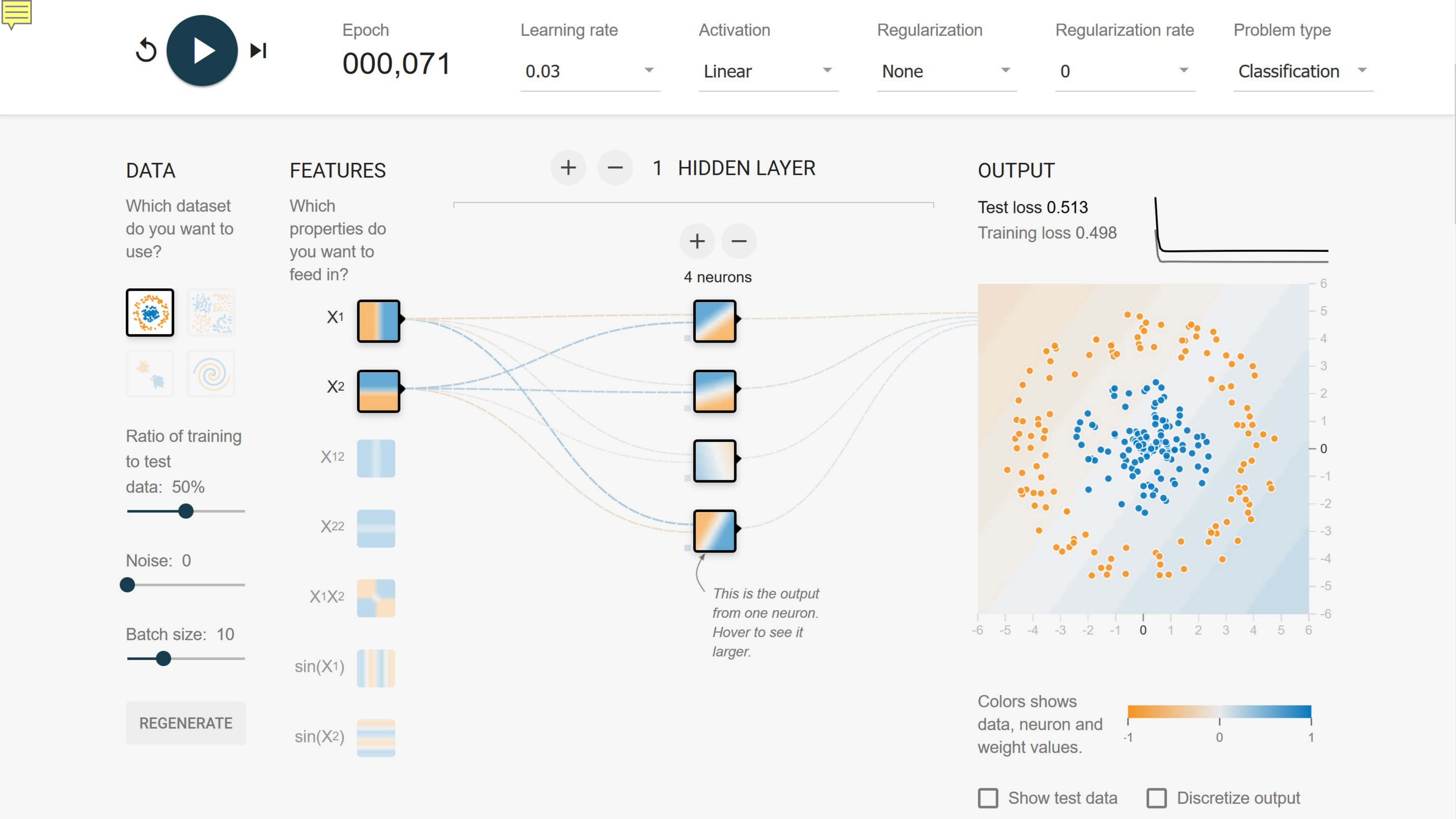
A B C … M N O P …
Layer 2
Layer 1
P R E D I C T I

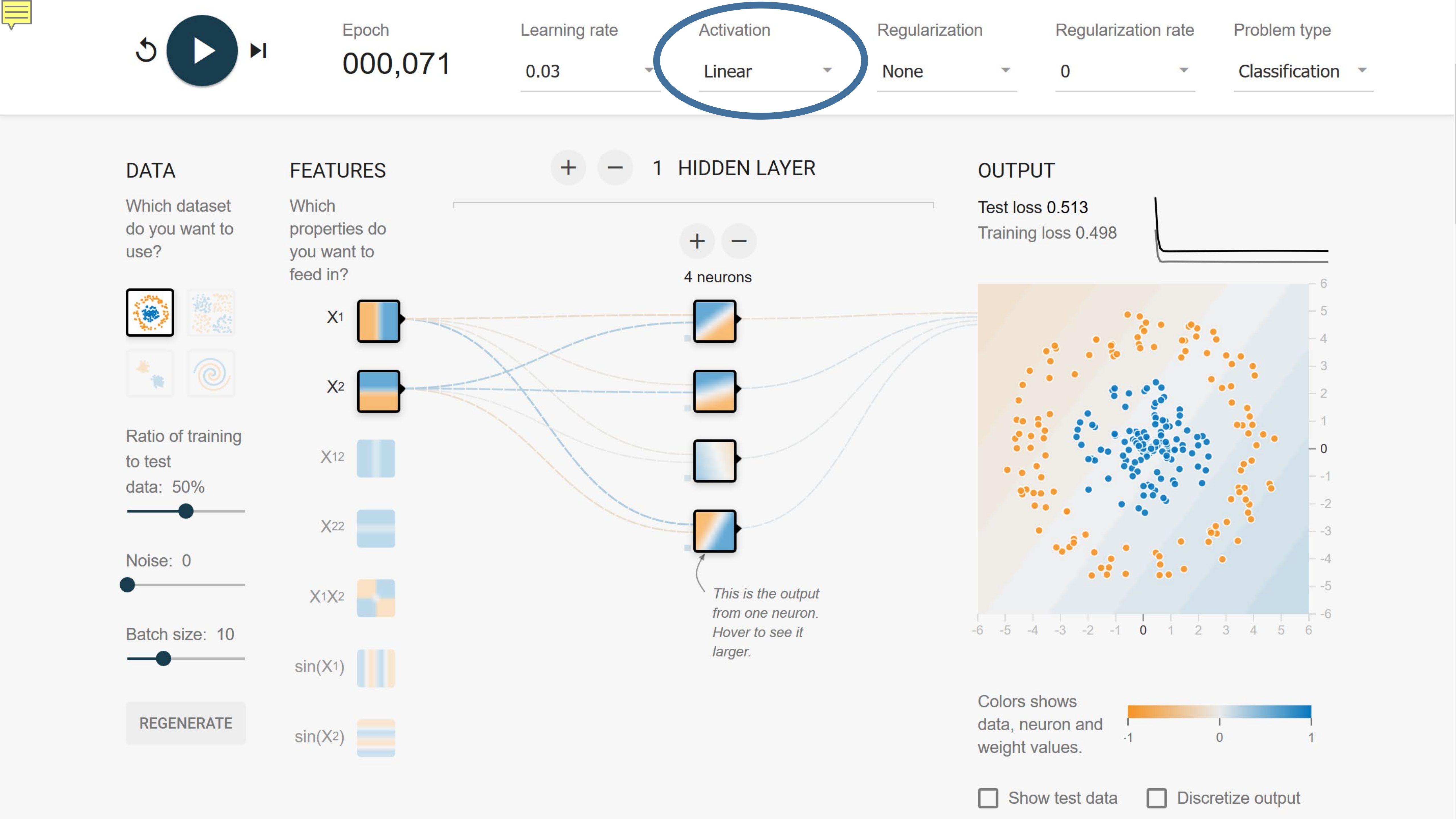
Contents

- Preliminaries: compute graphs, artificial neurons, activation functions, loss functions
- **Understanding nonlinear activations**
- Image generator architectures

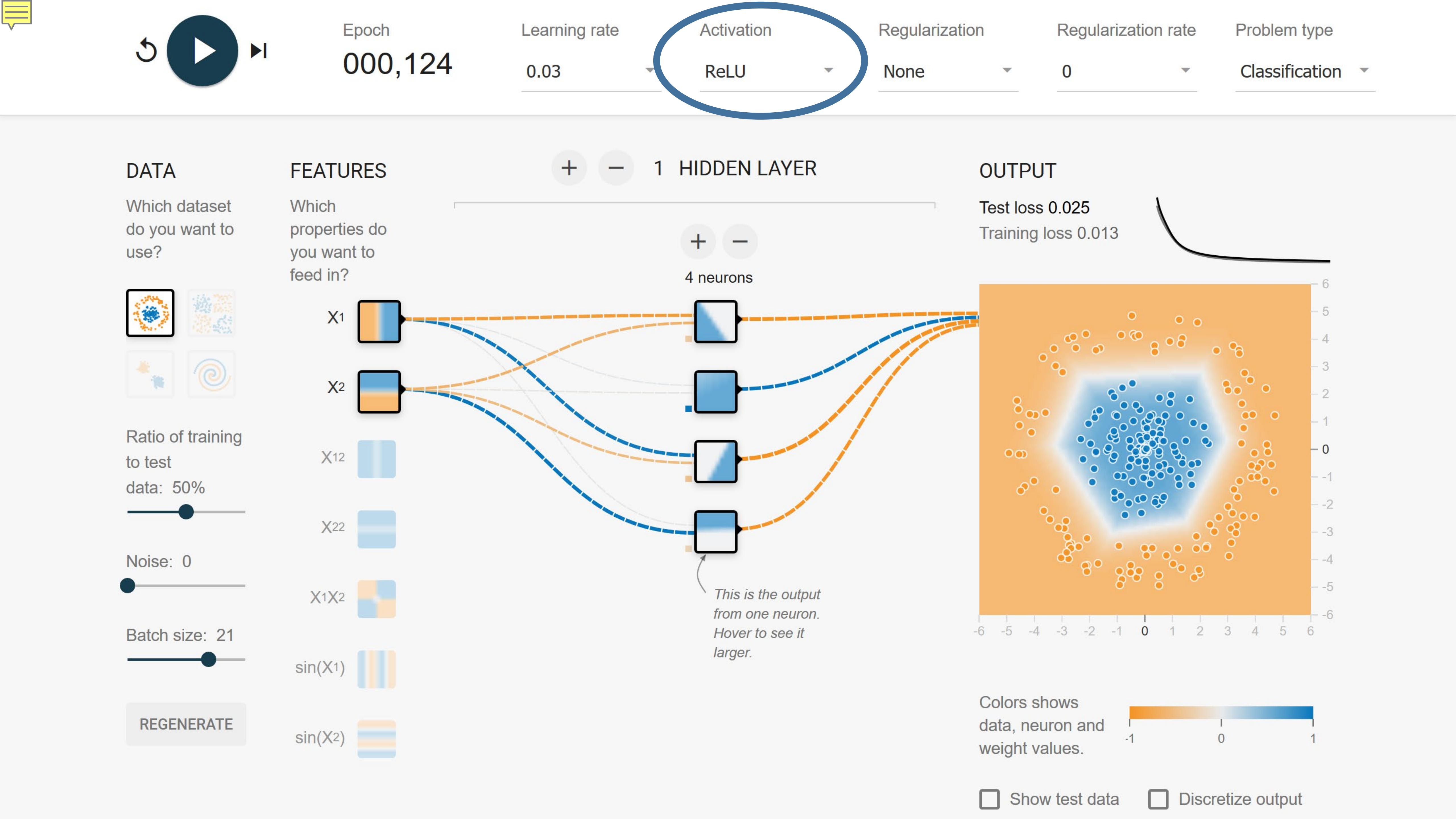
Contents

- Preliminaries: compute graphs, artificial neurons, activation functions, loss functions
- **Understanding nonlinear activations**
- Understanding skip-connections
- Convolutional neural networks
- Encoder-decoder architectures
- Applications, software packages
- Transfer learning


Epoch
000,071
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)
1 HIDDEN LAYER
4 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.513
Training loss 0.498
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

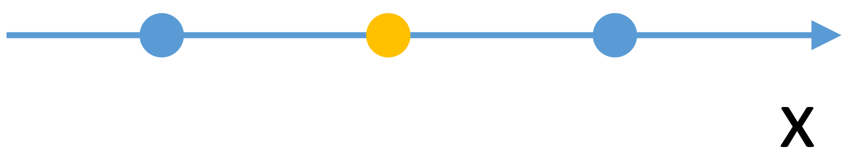


Epoch
000,071
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
4 neurons
OUTPUT
Test loss 0.513
Training loss 0.498
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)
This is the output from one neuron. Hover to see it larger.
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

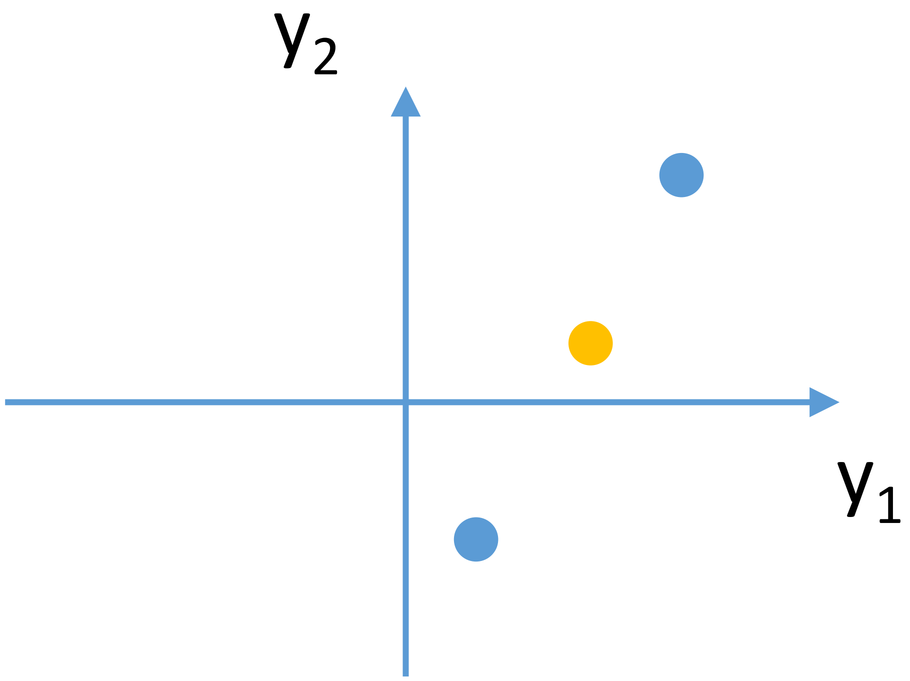

Epoch
000,124
Learning rate
0.03
Activation
ReLU
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 21
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)
1 HIDDEN LAYER
4 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.025
Training loss 0.013
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

The effect of nonlinearity

1D input



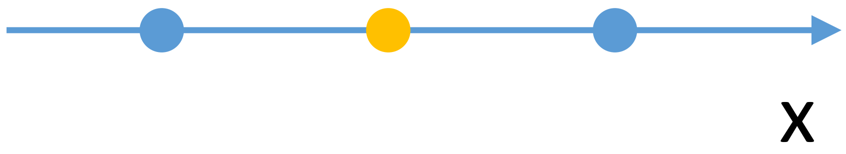
2D output of 2 linear neurons



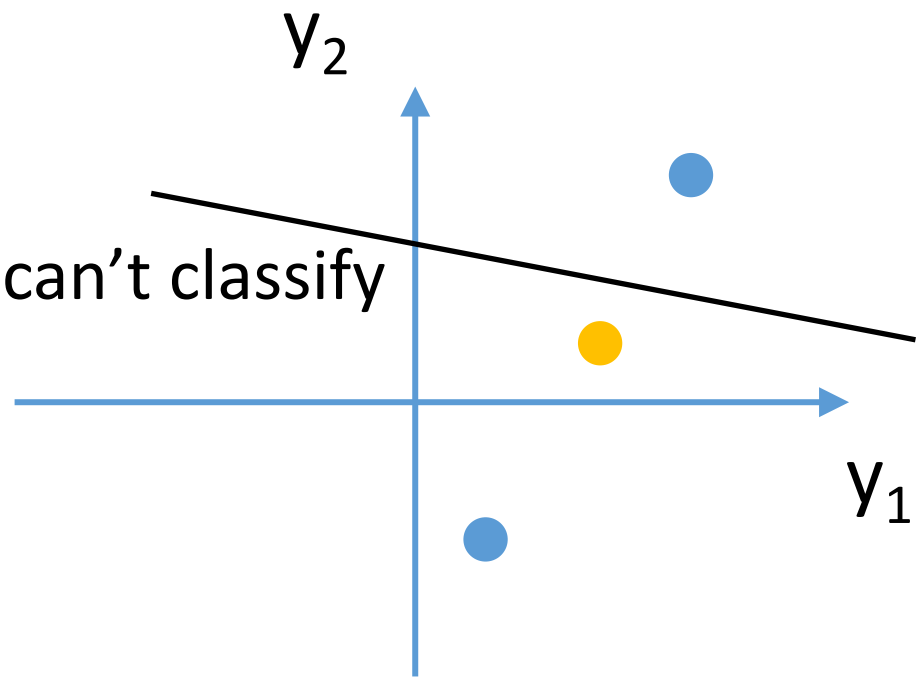
The effect of nonlinearity

1D input

2D output of 2 linear neurons

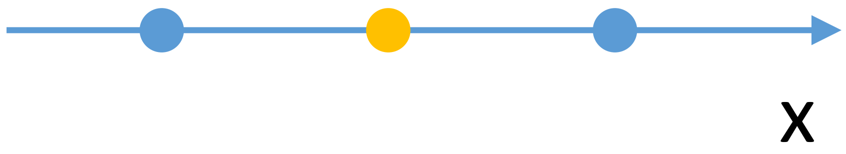


Output neuron can't classify

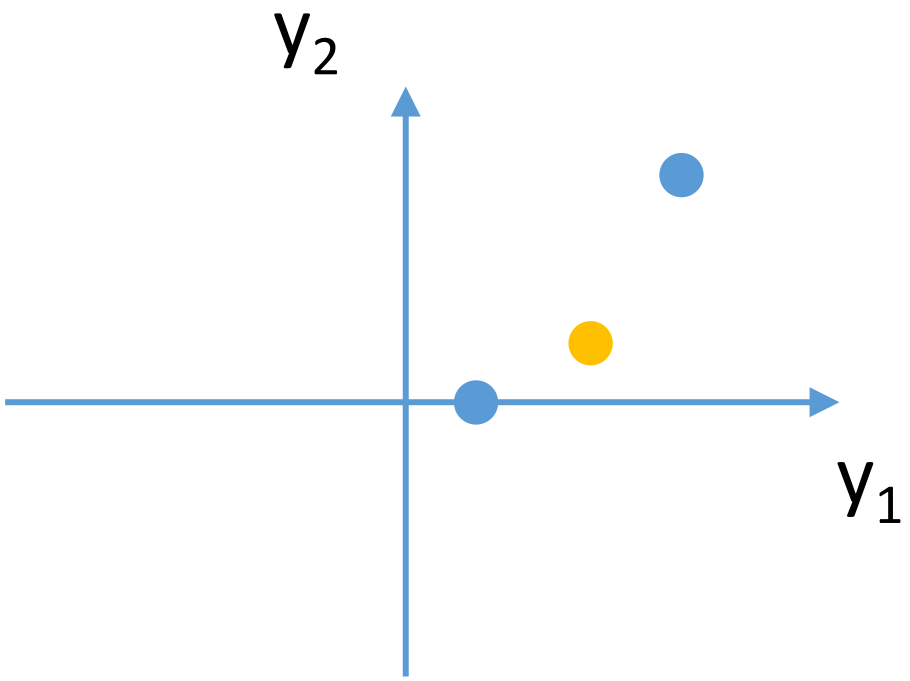


The effect of nonlinearity

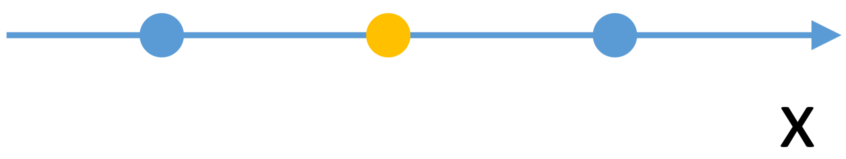
1D input

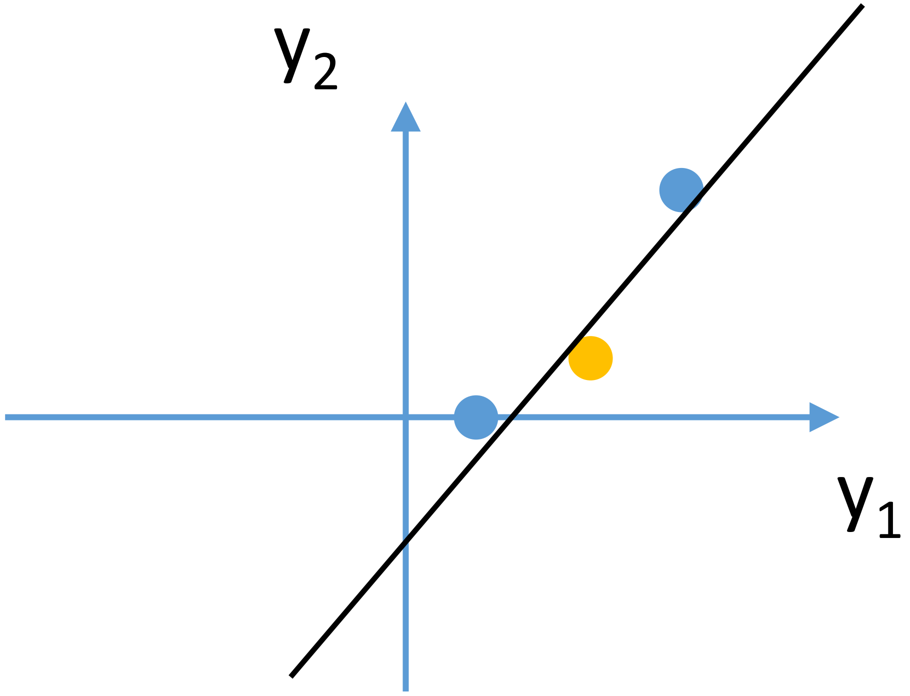


2D output of 2 RELU neurons



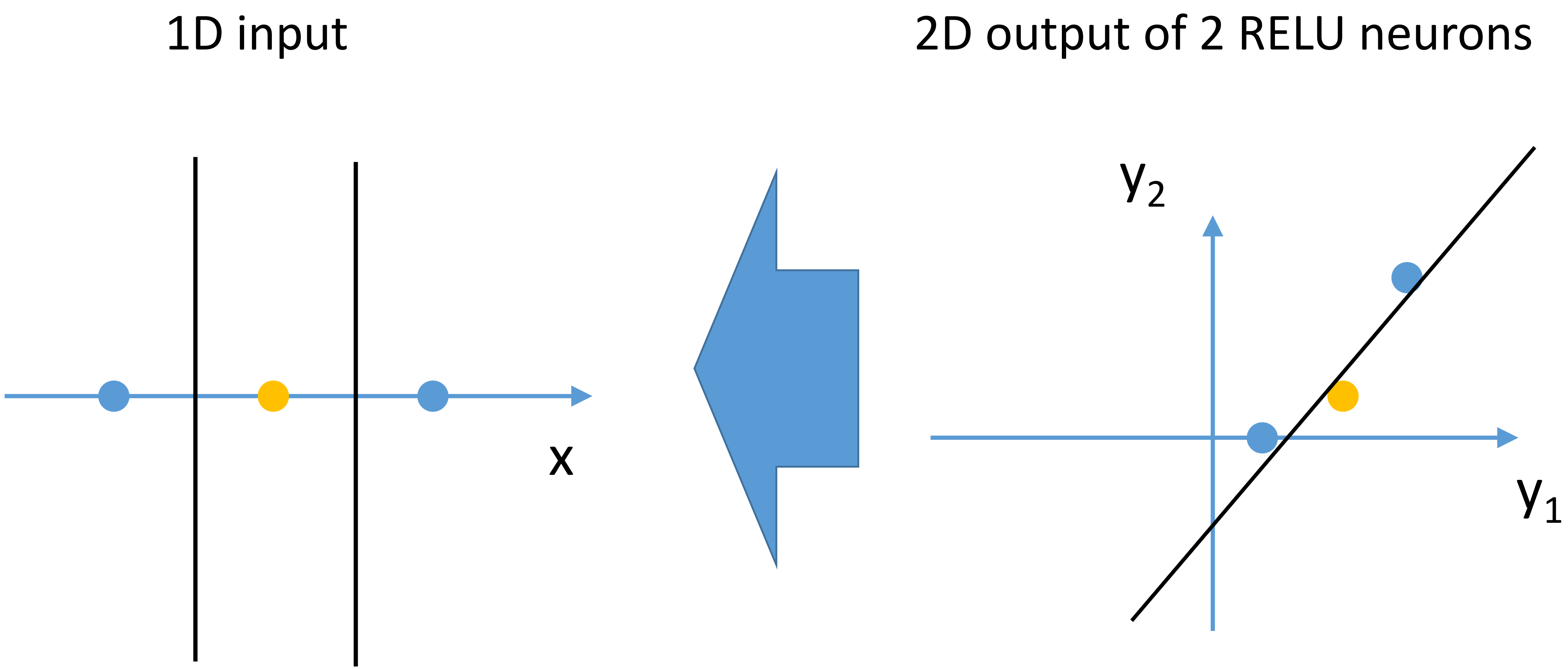
The effect of nonlinearity



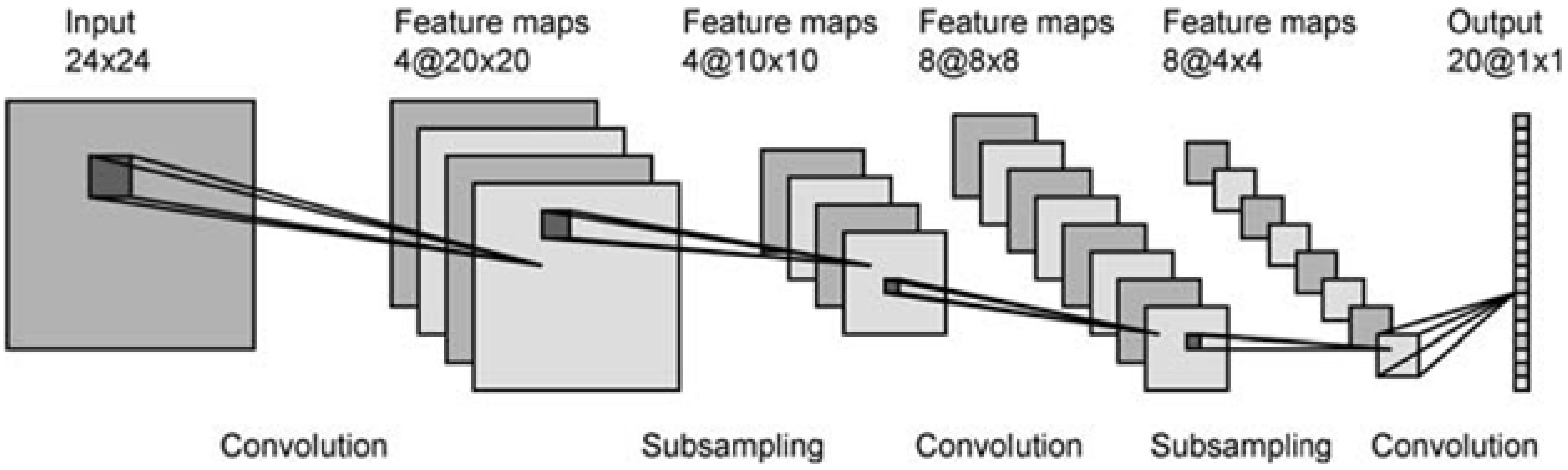


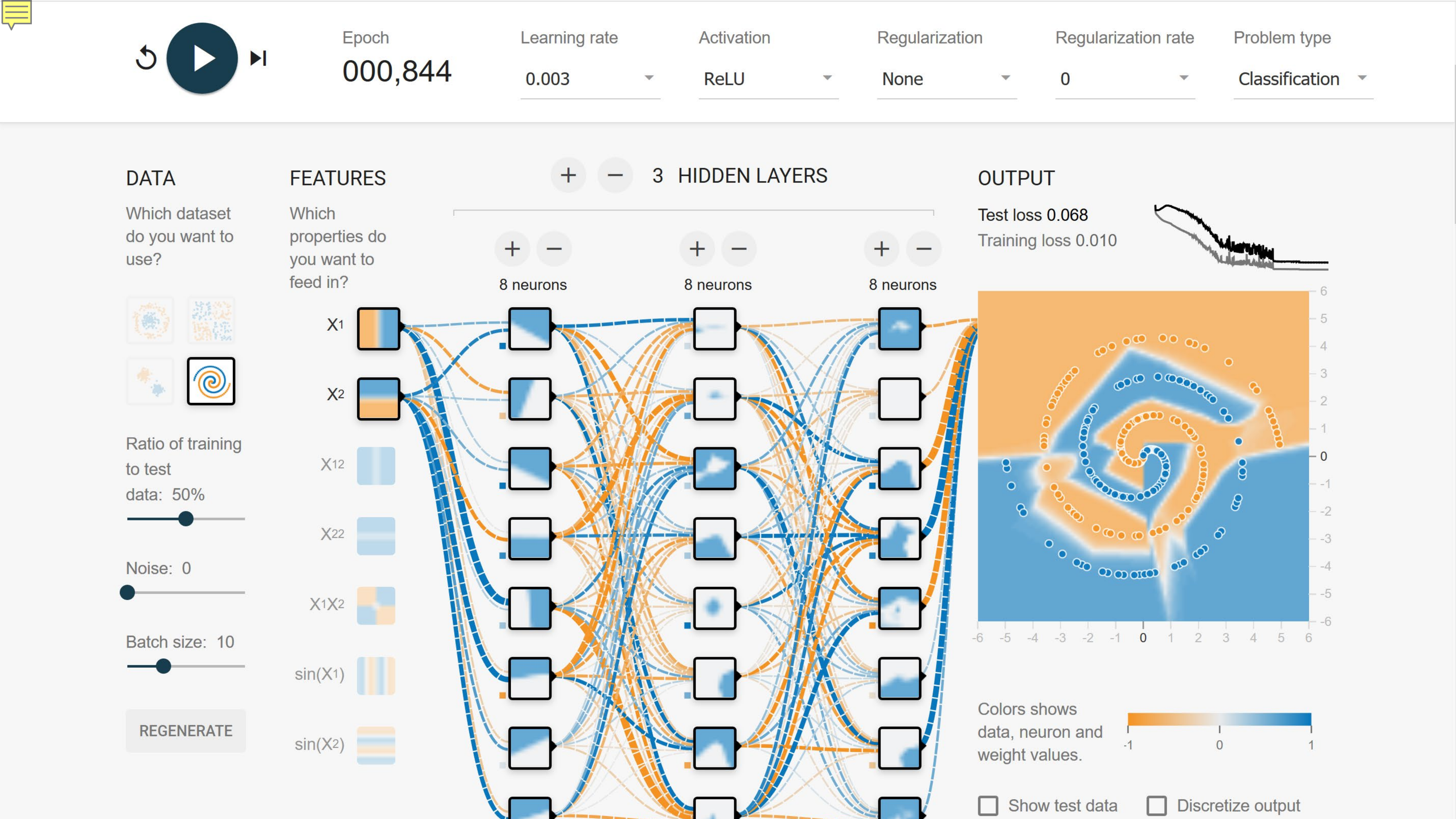
Linear split works now!

Split into two halves in a high-dimensional representation = multiple splits in the input space

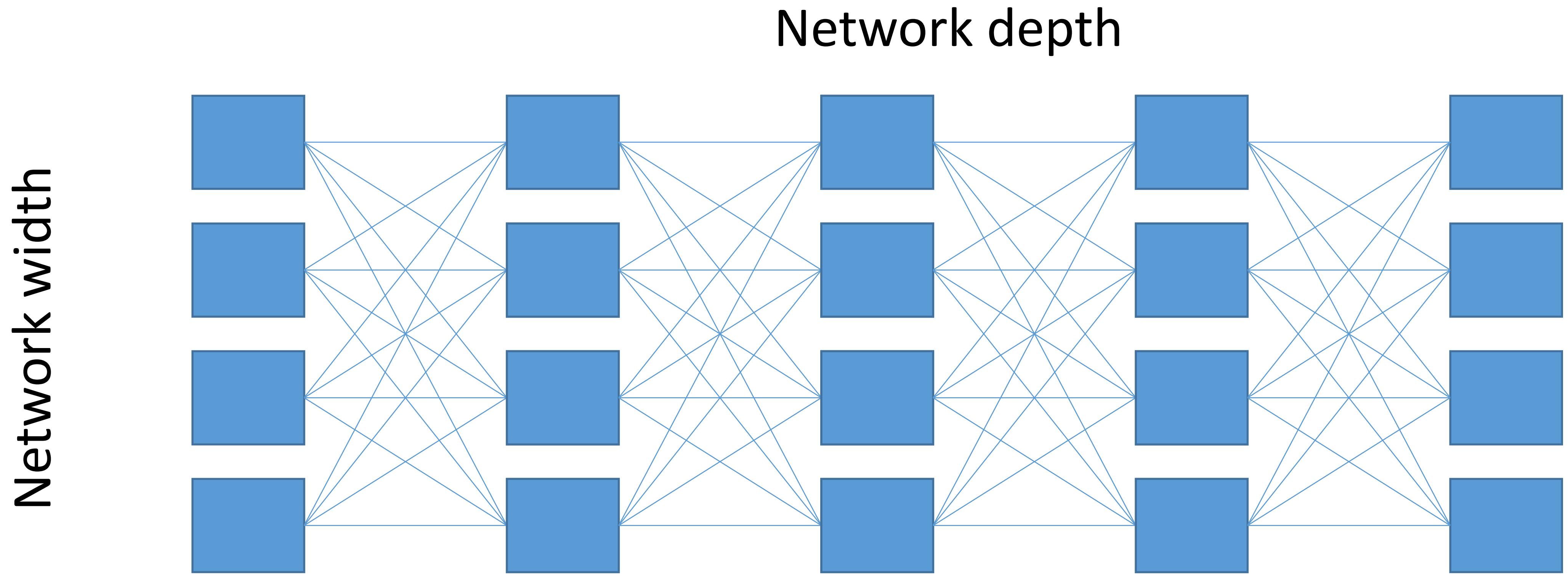


Later layers usually have more neurons => next layer has higher-dimensional inputs and can do more complex splits

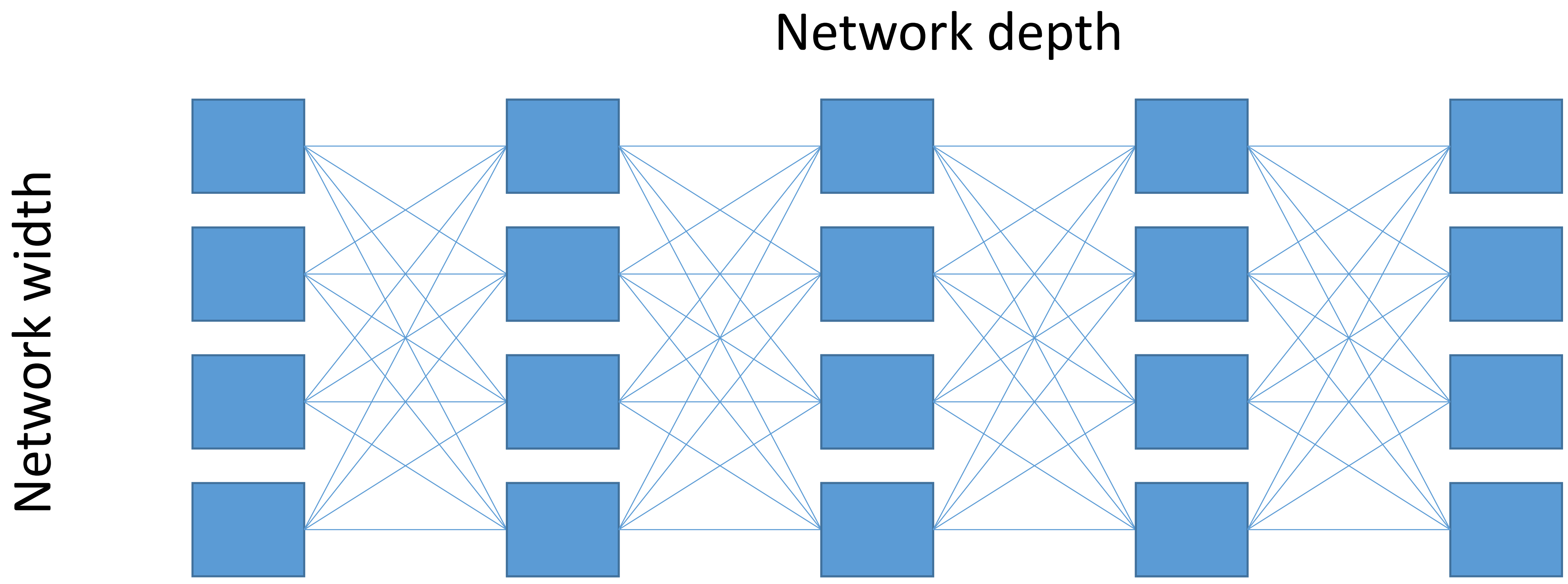



Epoch
000,844
Learning rate
0.003
Activation
ReLU
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
FEATURES
Which properties do you want to feed in?
3 HIDDEN LAYERS
8 neurons
8 neurons
8 neurons
OUTPUT
Test loss 0.068
Training loss 0.010
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Intuition on the power of depth



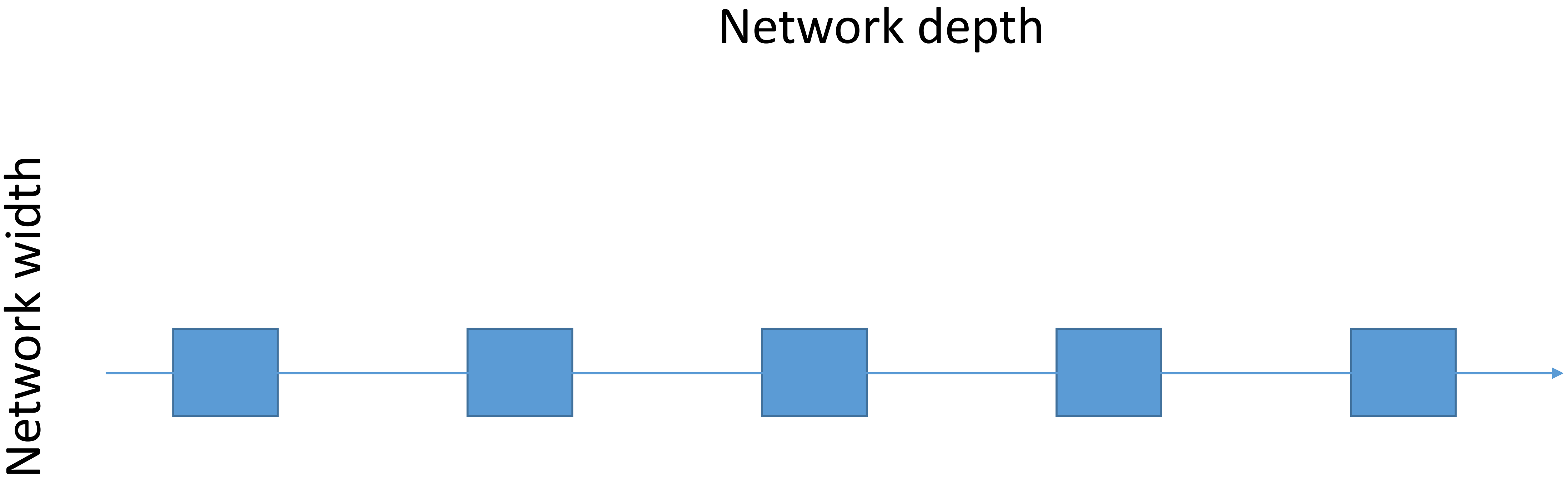
Intuition on the power of depth



Neurons turning on and off ”route” the data through the network along different paths.

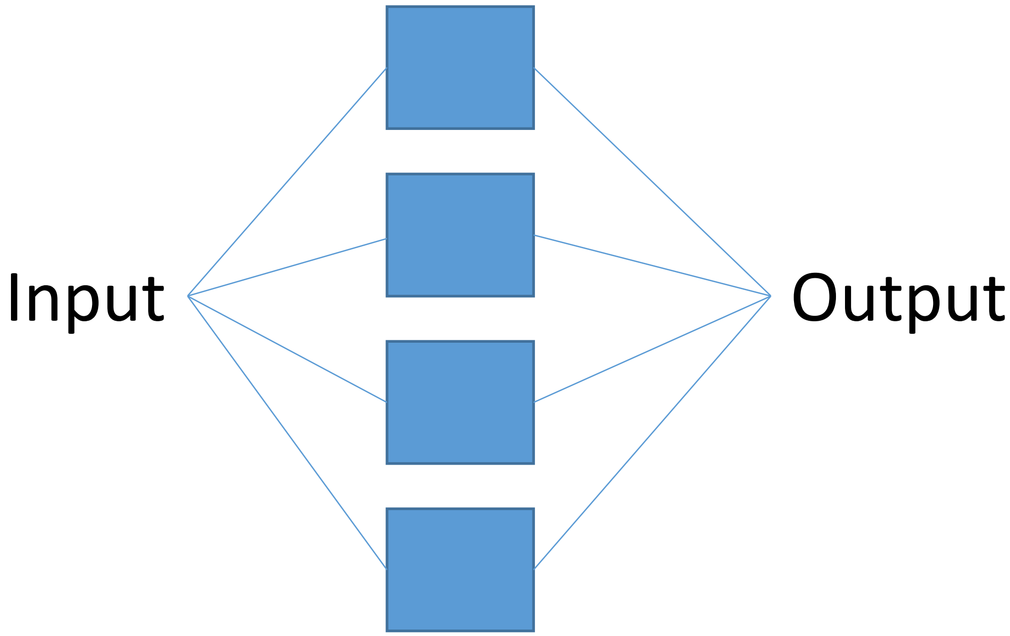
How many paths can the data take through the network?

Intuition on the power of depth



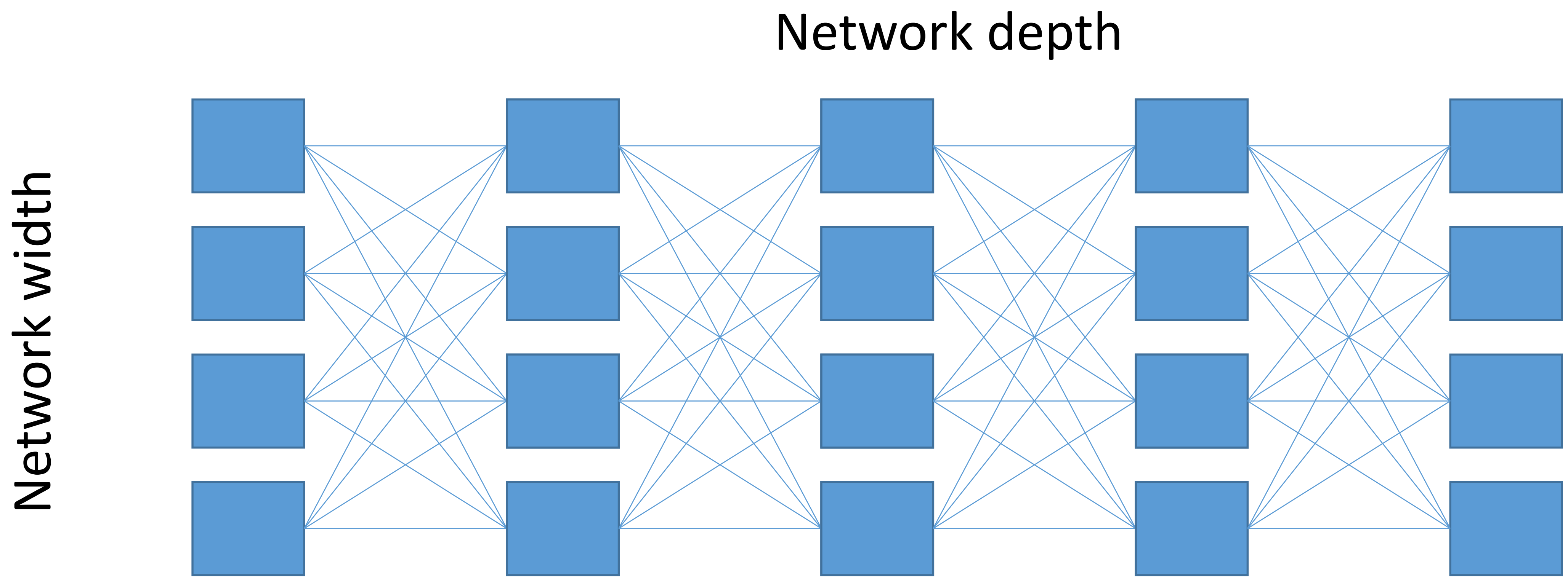
Narrow but deep network: only one path

Intuition on the power of depth



Wide but shallow network: Only as many paths as there are neurons

Intuition on the power of depth

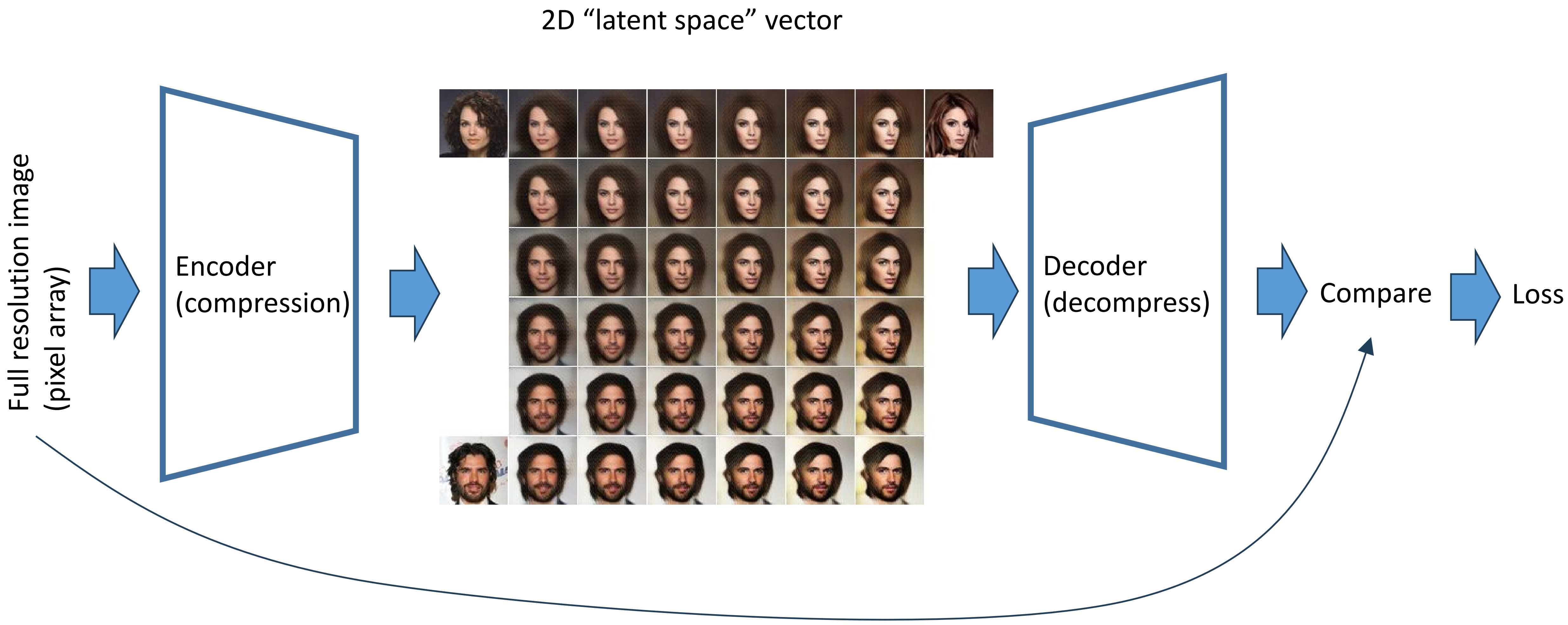


Wide and deep network: $\text{width}^{\text{depth}}$ paths. This is the power of deep learning.

Rules of thumb

- Various network blocks transform the input to a new representation space
- Transforming to a sufficiently high-dimensional space allows linear regression or classification of practically any data
  - A single high-dimensional linear split can carve out complex nonlinear regions of the input space
  - Generalizing to new data points may be poor => regularization techniques needed
- Transforming to a lower-dimensional space = compression, forces learning the most important underlying structure of the data

Training an autoencoder with a 2D "bottleneck"



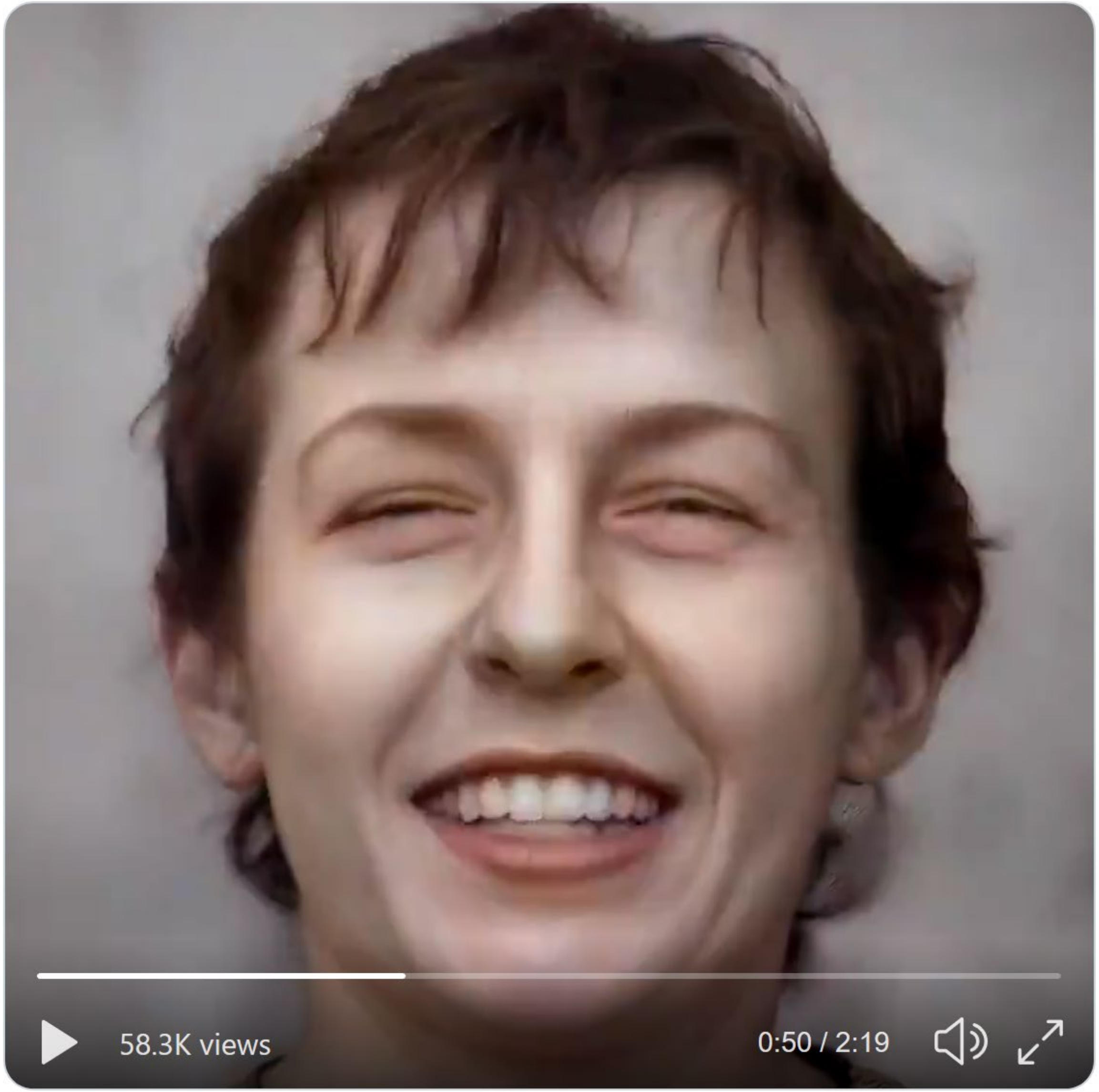
https://medium.com/@juliendespois/latent-space-visualization-deep-learning-bits-2-bd09a46920df

Music visualization using StyleGAN 2 latent space

https://twitter.com/quasimondo/status/1244562140217905153?s=20

Mario Klingemann @quasimondo · Mar 30
Current progress on mapping music to facial expression vectors. #StyleGAN2 #realtime

Song: "Triggernometry" by Kraftamt, 2014



Types of image generators

Model type	Generation speed	Image quality	Image diversity	Interpolable latent space	Examples
VAE (Variational Autoencoder)	fast	low	high	Yes	https://arxiv.org/abs/1312.6114
GAN (Generative Adversarial Networks)	fast	high	low	Yes	StyleGAN 1-3, BigGAN, https://github.com/NVlabs/stylegan3, https://www.tensorflow.org/hub/tutorials/biggan_generation_with_tf_hub
Flow-based	fast	medium	high	Yes	https://openai.com/research/glow
Transformer (autoregressive generation patch-by-patch)	slow	high	high	No	https://github.com/google-research/parti
Diffusion	very slow	very high	high	No (only jumpy)	DALL-E, Midjourney, Stable Diffusion