Neural Network Tools and Principles, part 1

Discriminative models, e.g., image classification, object detection, body tracking

Intelligent Computational Media, Spring 2020
Prof. Perttu Hämäläinen
Aalto University

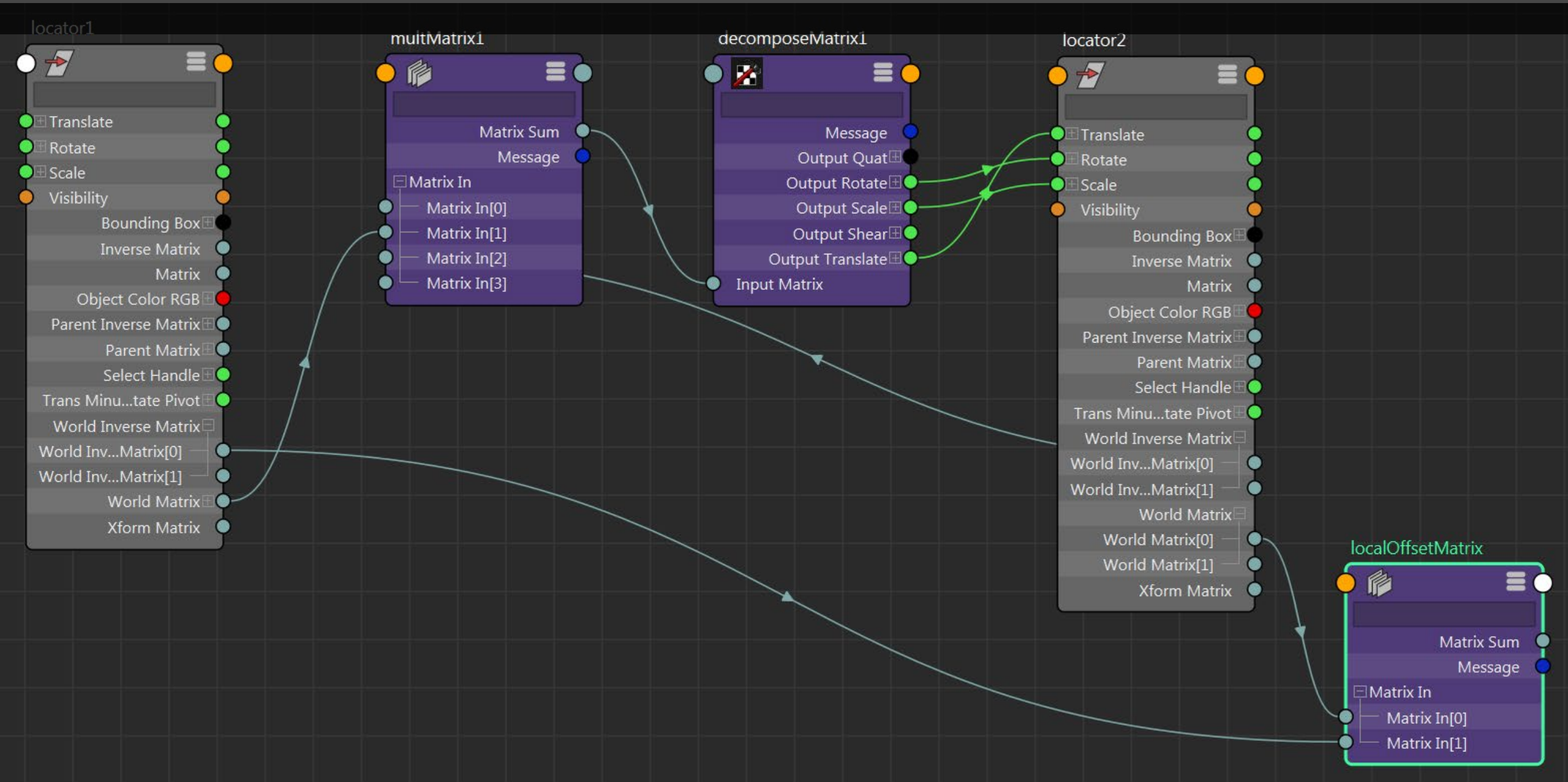
Contents

- **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

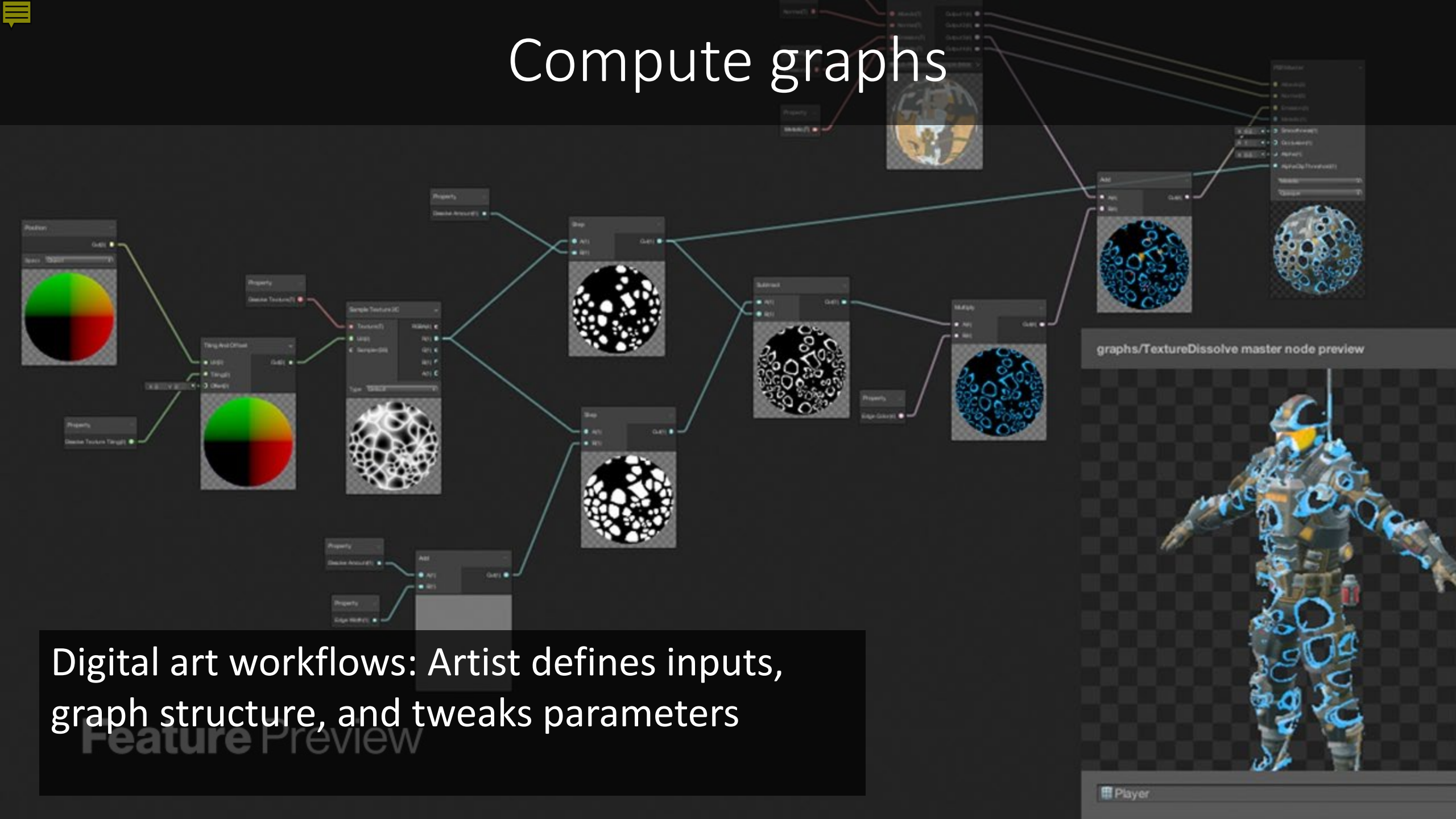
3Blue1Brown YouTube channel: Visual intuitions to math and neural networks

- 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



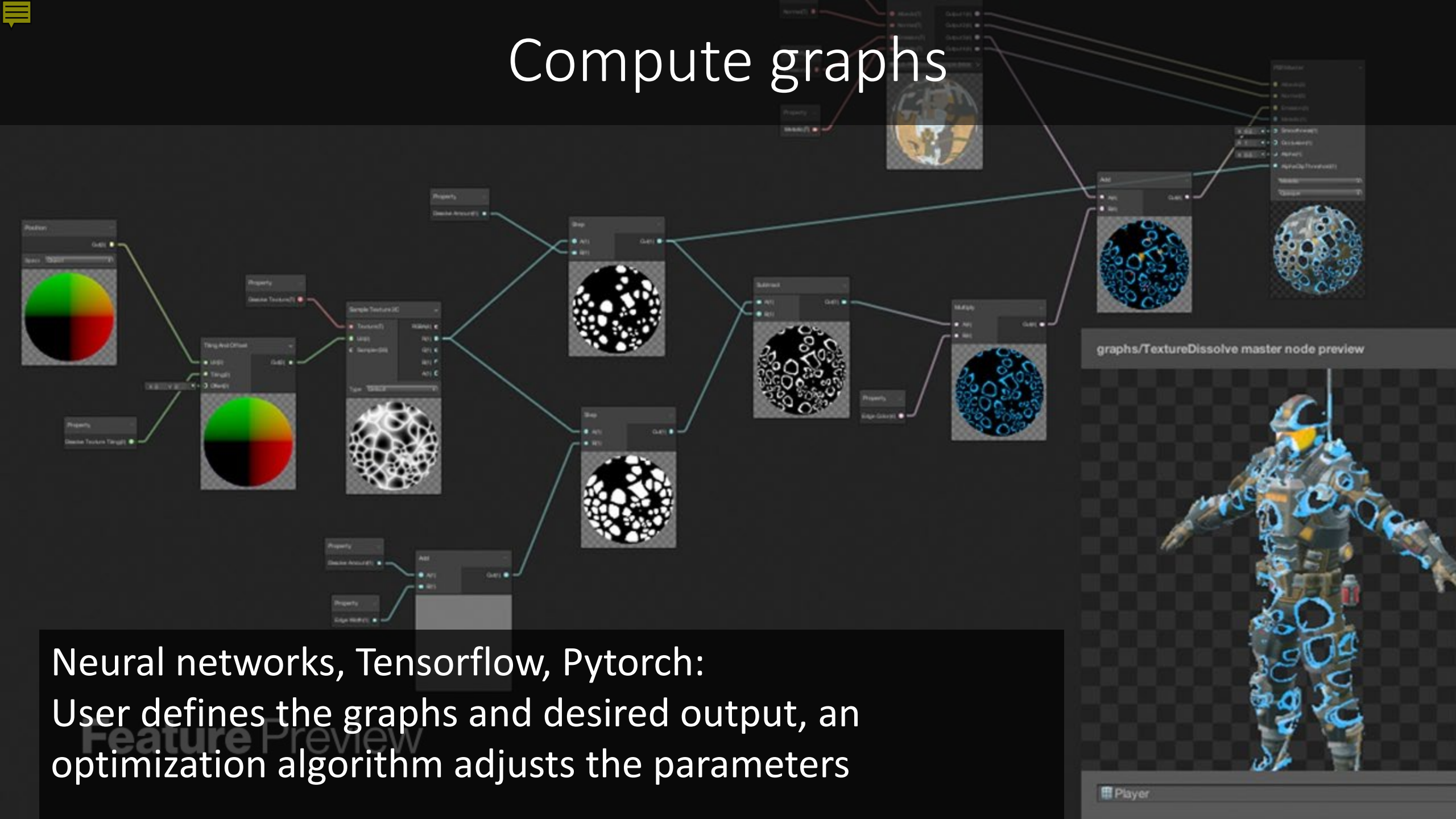
Compute graphs



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

Feature Preview

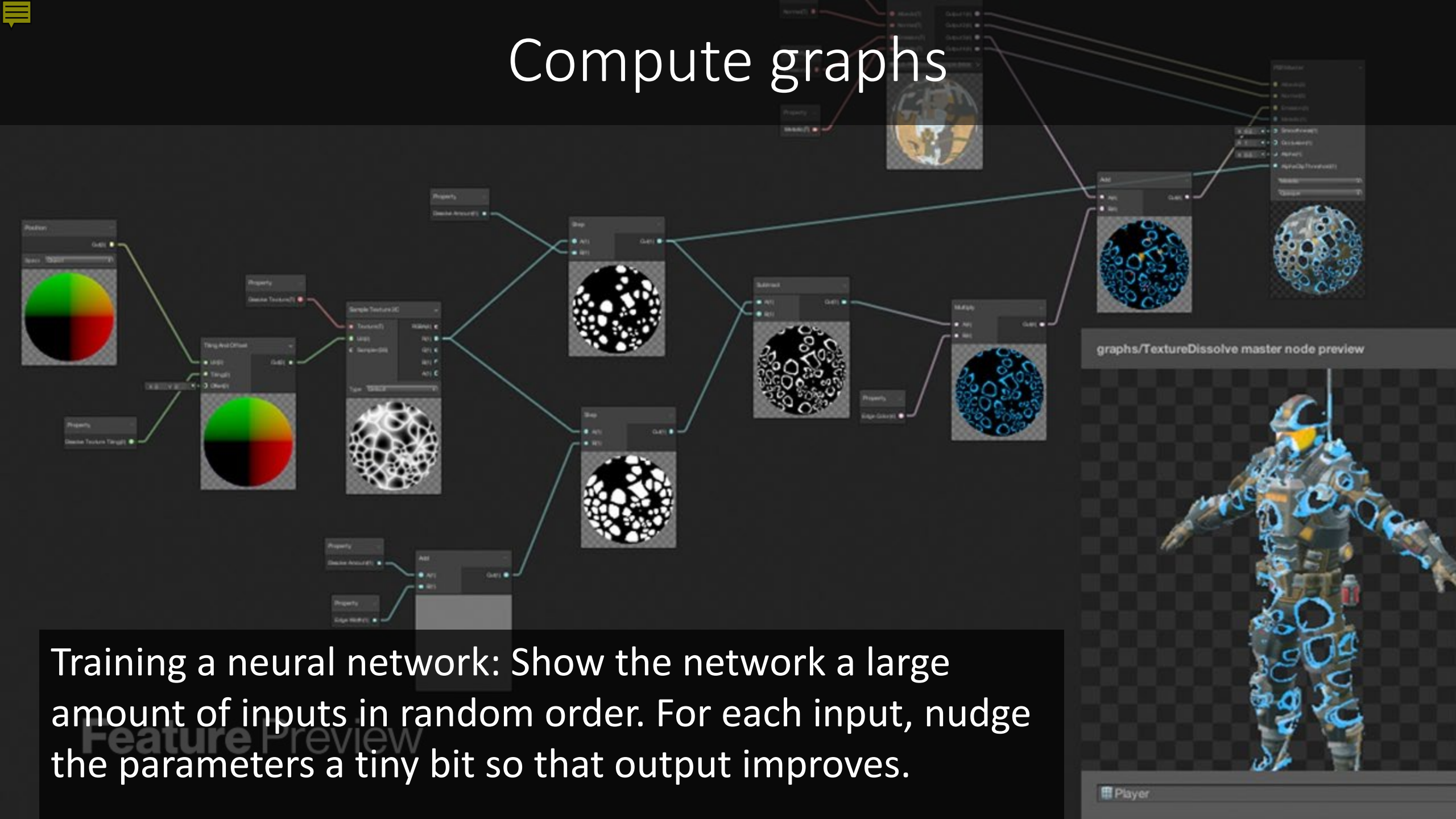
Compute graphs



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

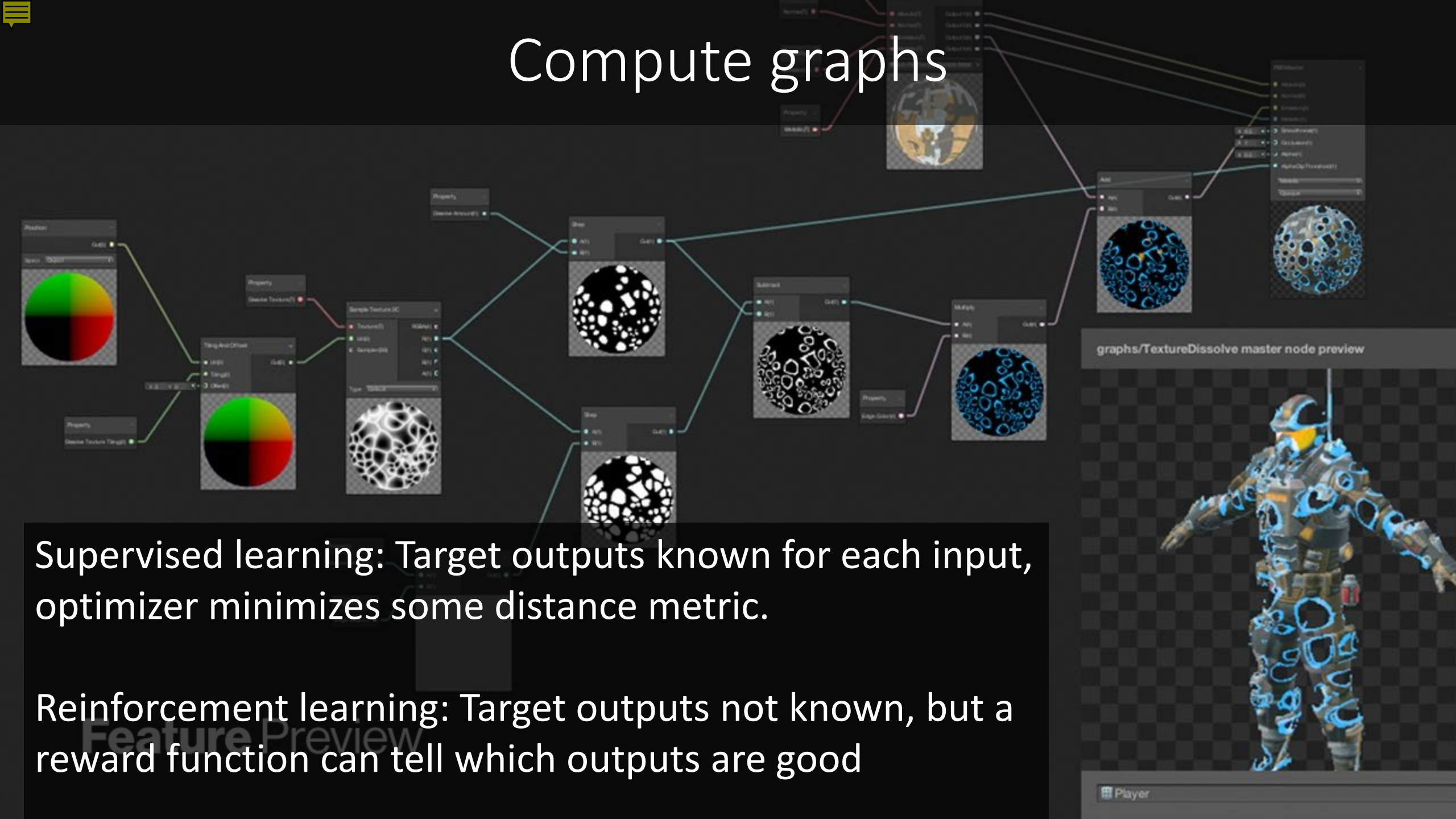
Feature Preview

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

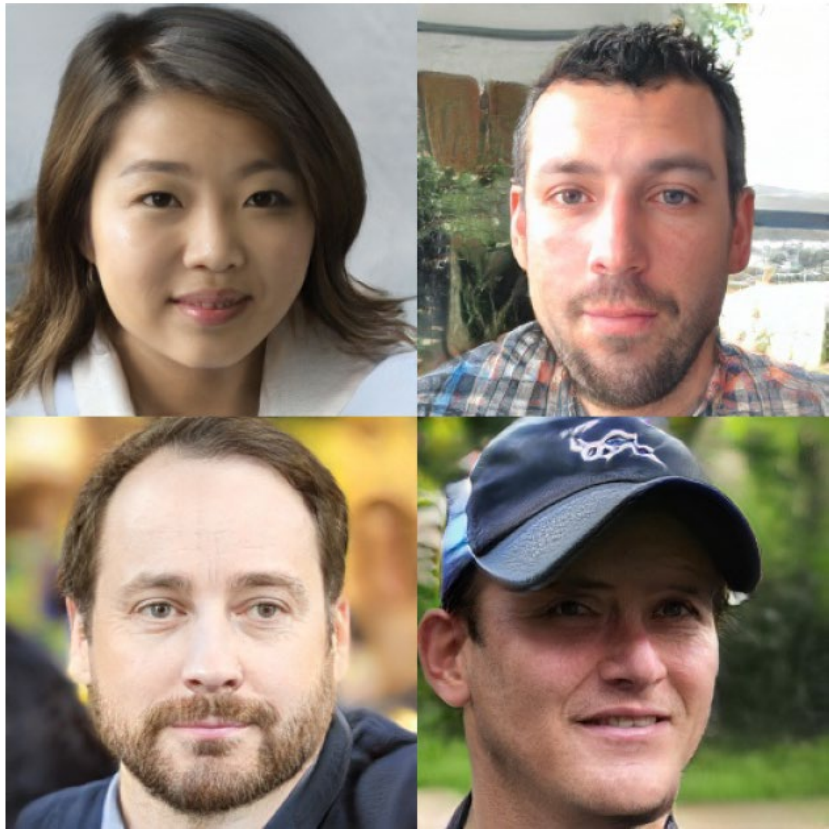


Supervised learning: Target outputs known for each input, optimizer minimizes some distance metric.

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

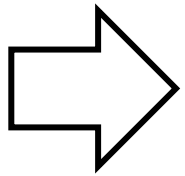
The same network “broken” with unrelated images of rocks, trees, foliage

Working & thinking with compute graphs

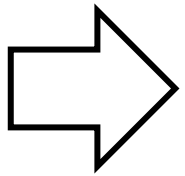
- A generator network takes in “latent space” vectors, i.e., vectors of hidden variables that explain the results (face gender, age, rotation…)
- Training the network = based on training images, infer an approximation of the latent variables and the generative process

Latent space vector

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



Generator network

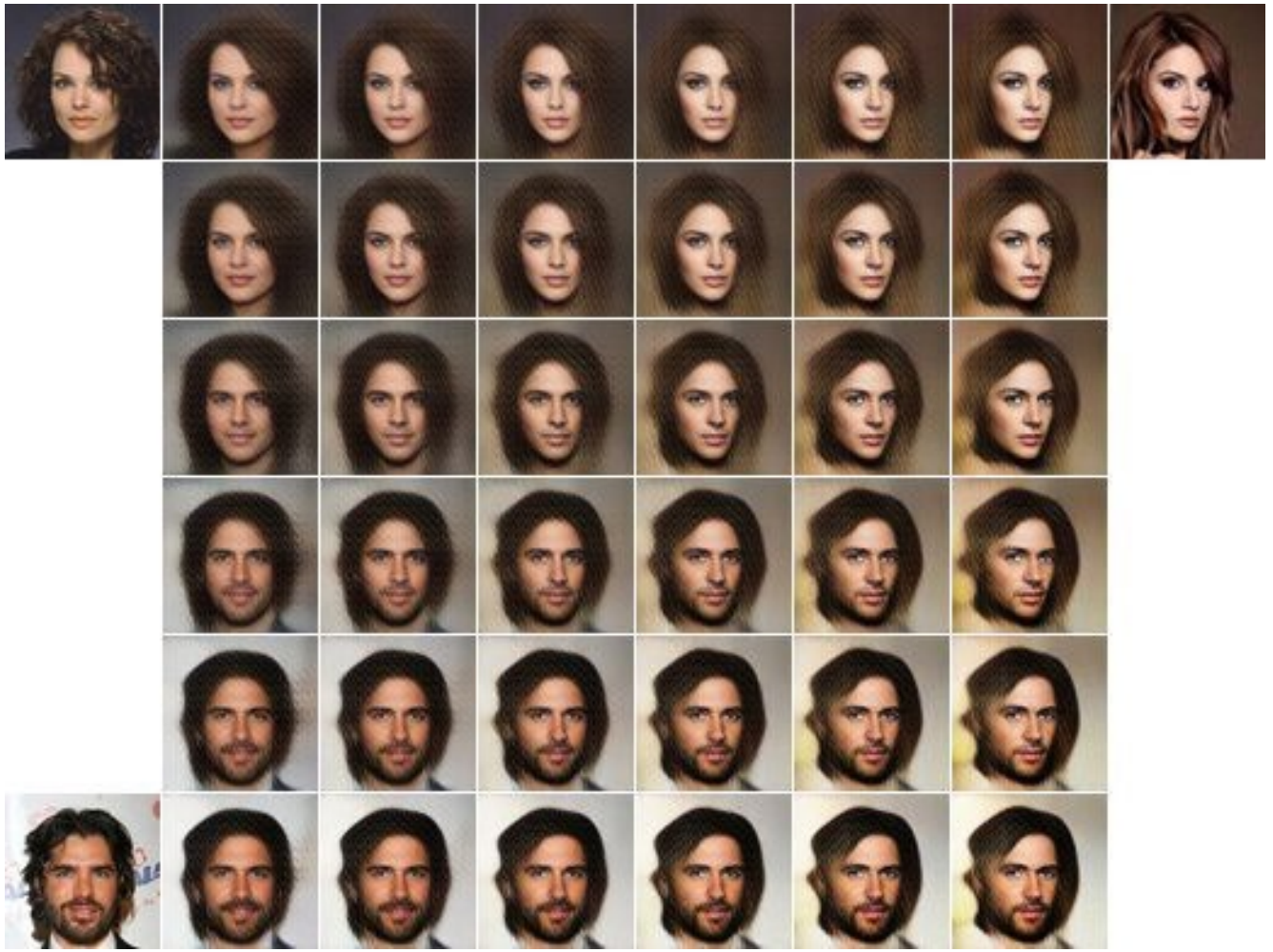


Output image: this is just an array of numbers (pixel RGB values)

$$[[r,g,b,\ldots]$$
$$[r,g,b,\ldots]$$
$$\vdots$$
$$\vdots$$
$$]$$

=

A 2D latent space for generating faces



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

Everything is just numbers

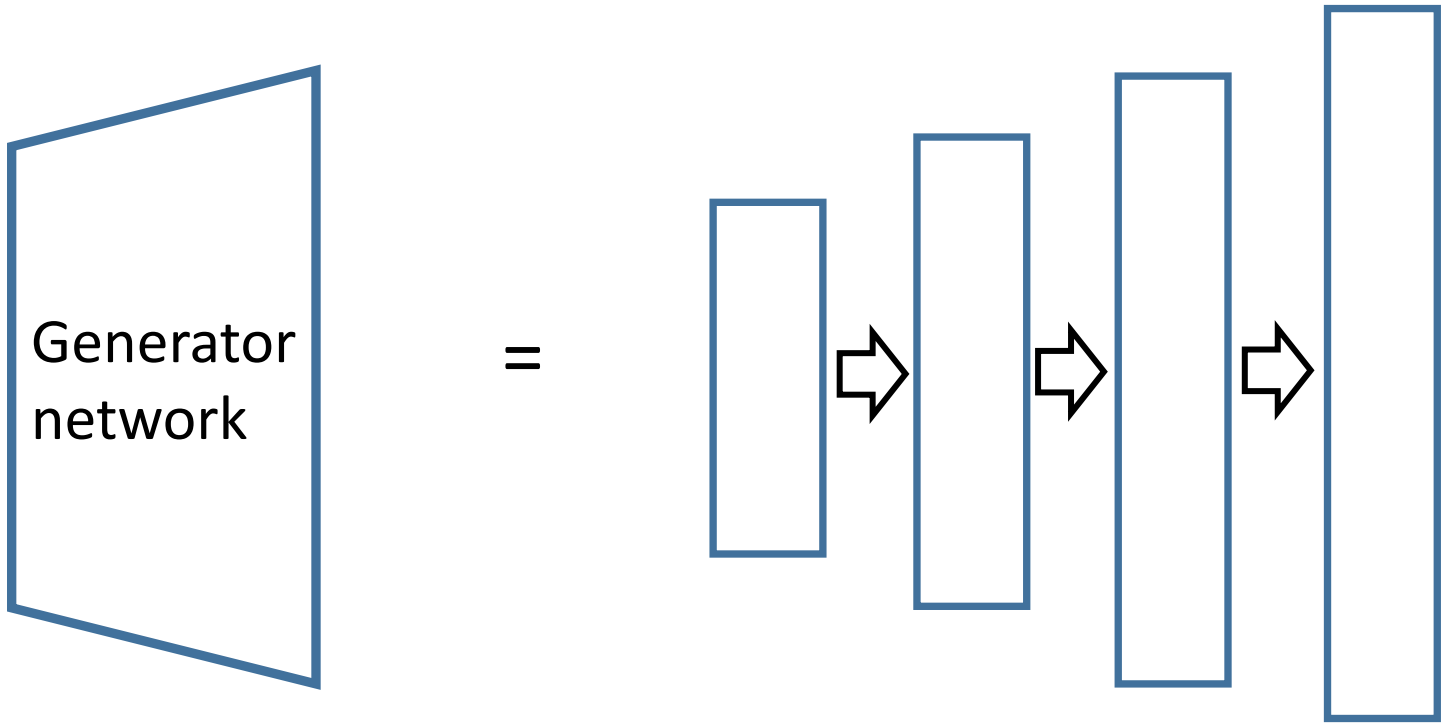
- 1-dimensional arrays: Text (each number corresponding to a character), 1-channel audio
- 2-dimensional arrays: grayscale images, stereo audio
- 3-dimensional arrays: color images (array shape [width,height,channels]), multiple stereo audio files
- 4-dimensional arrays: multiple color images, [index,width,height,channels]

…

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

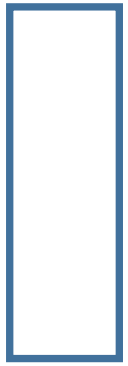
- Usually, a neural network can be broken down into layers
- What data moves between layers and how is defined by the network designer
- This is like an electronics device consisting of a few integrated circuits like a microcontroller and a Bluetooth chip.



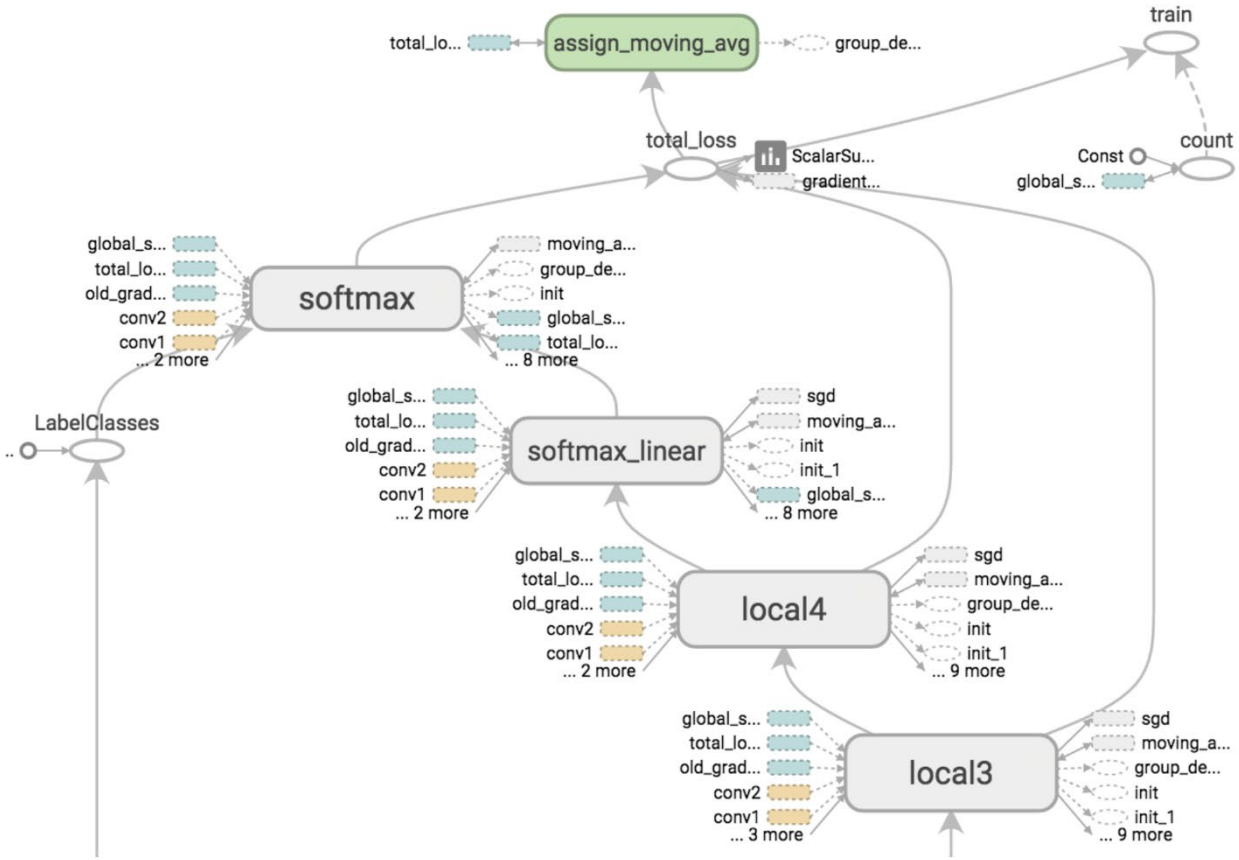
Working & thinking with compute graphs

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

Network layer

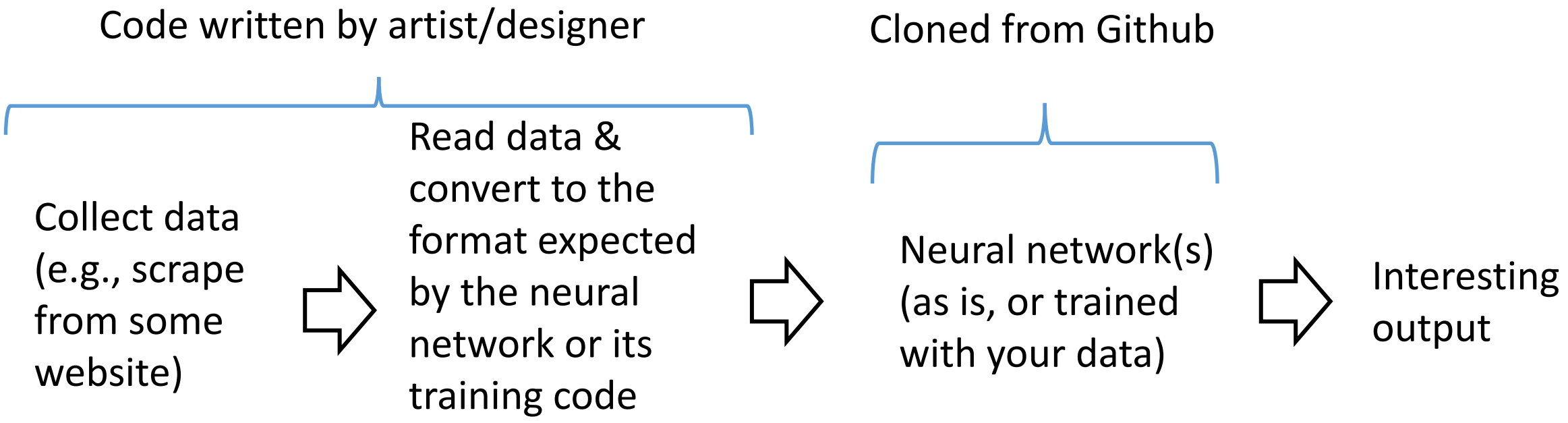


=



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?



Working & thinking with compute graphs

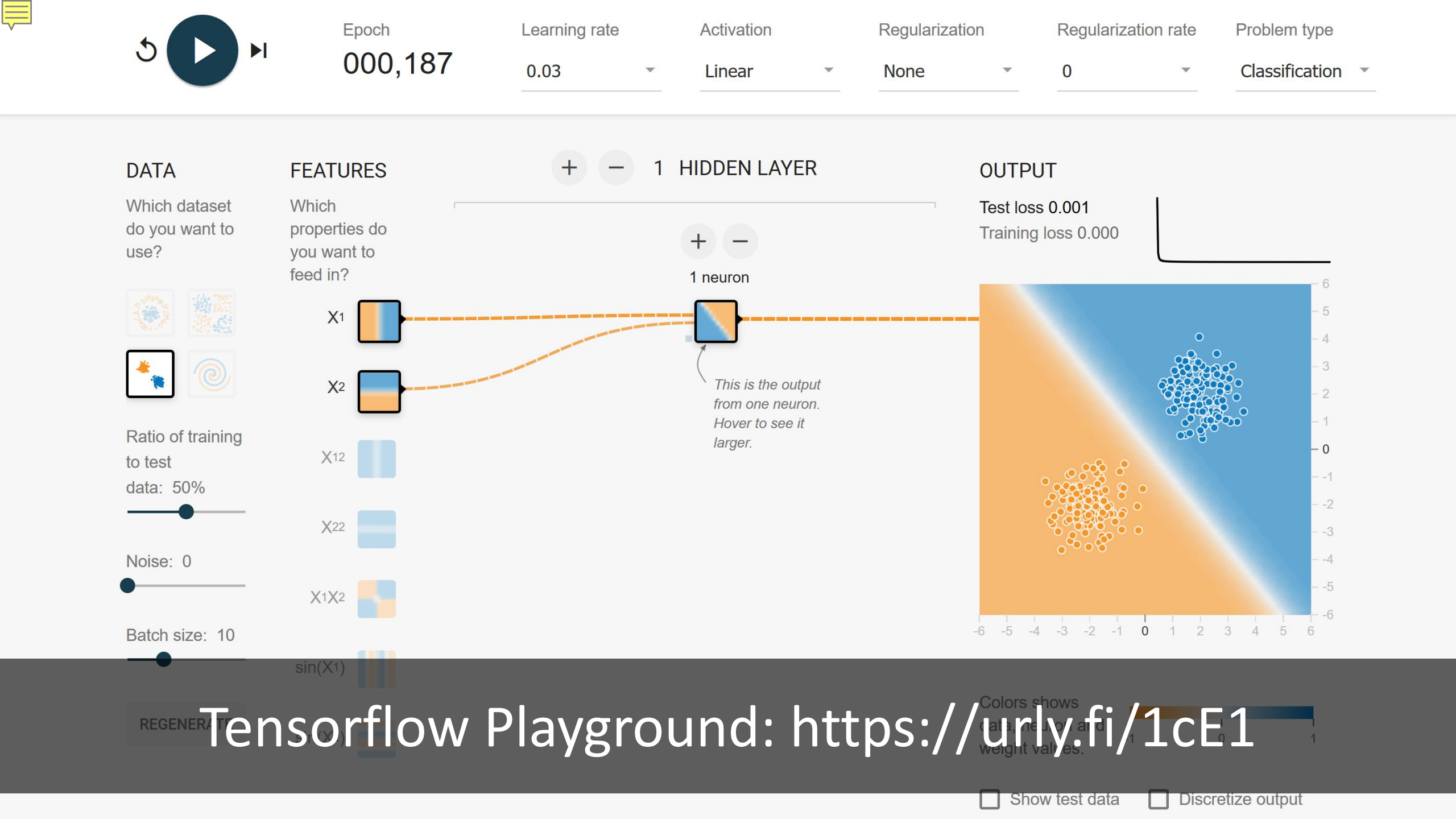
These results:

- 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

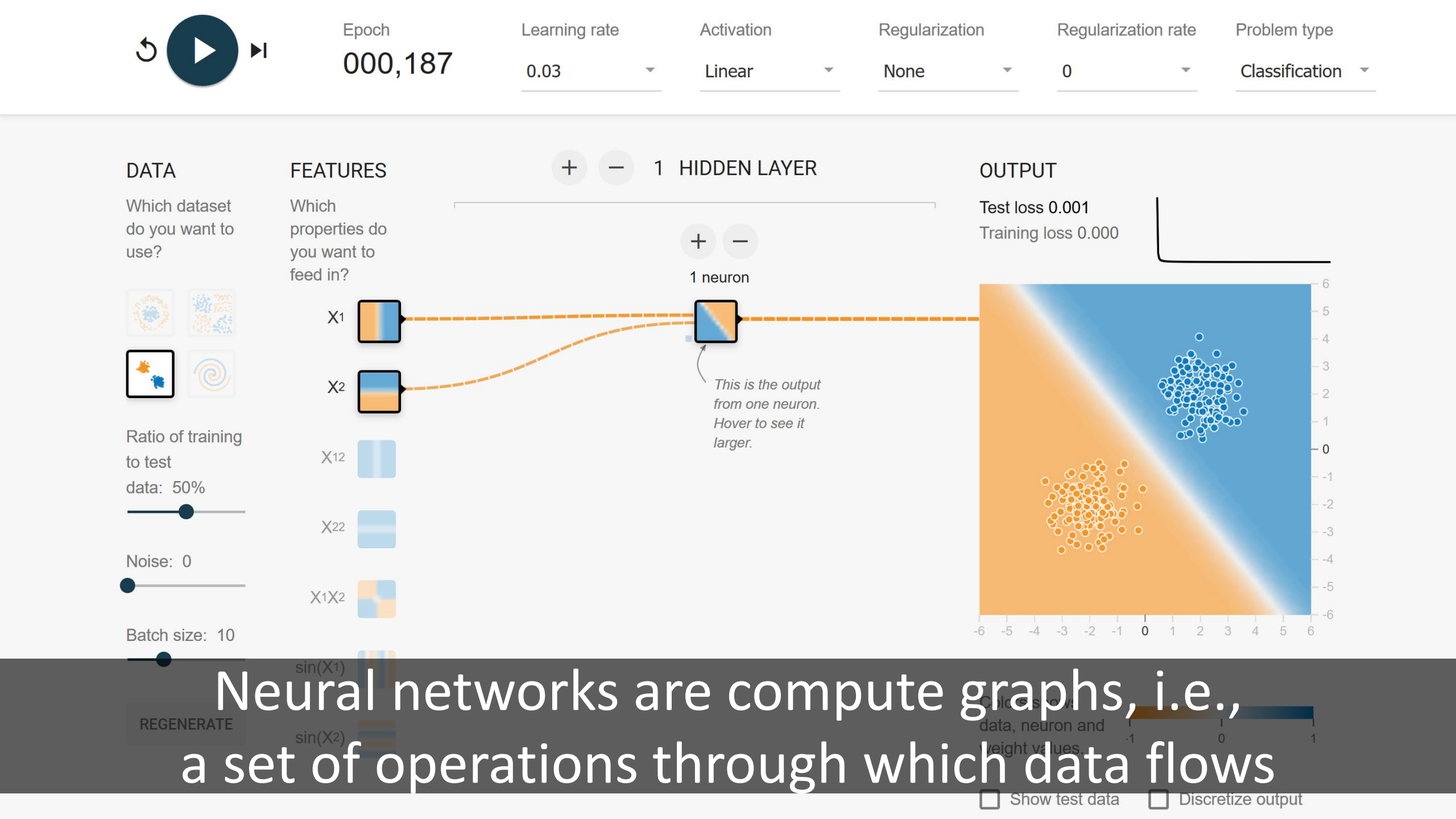
Working & thinking with compute graphs

Typical skills needed:

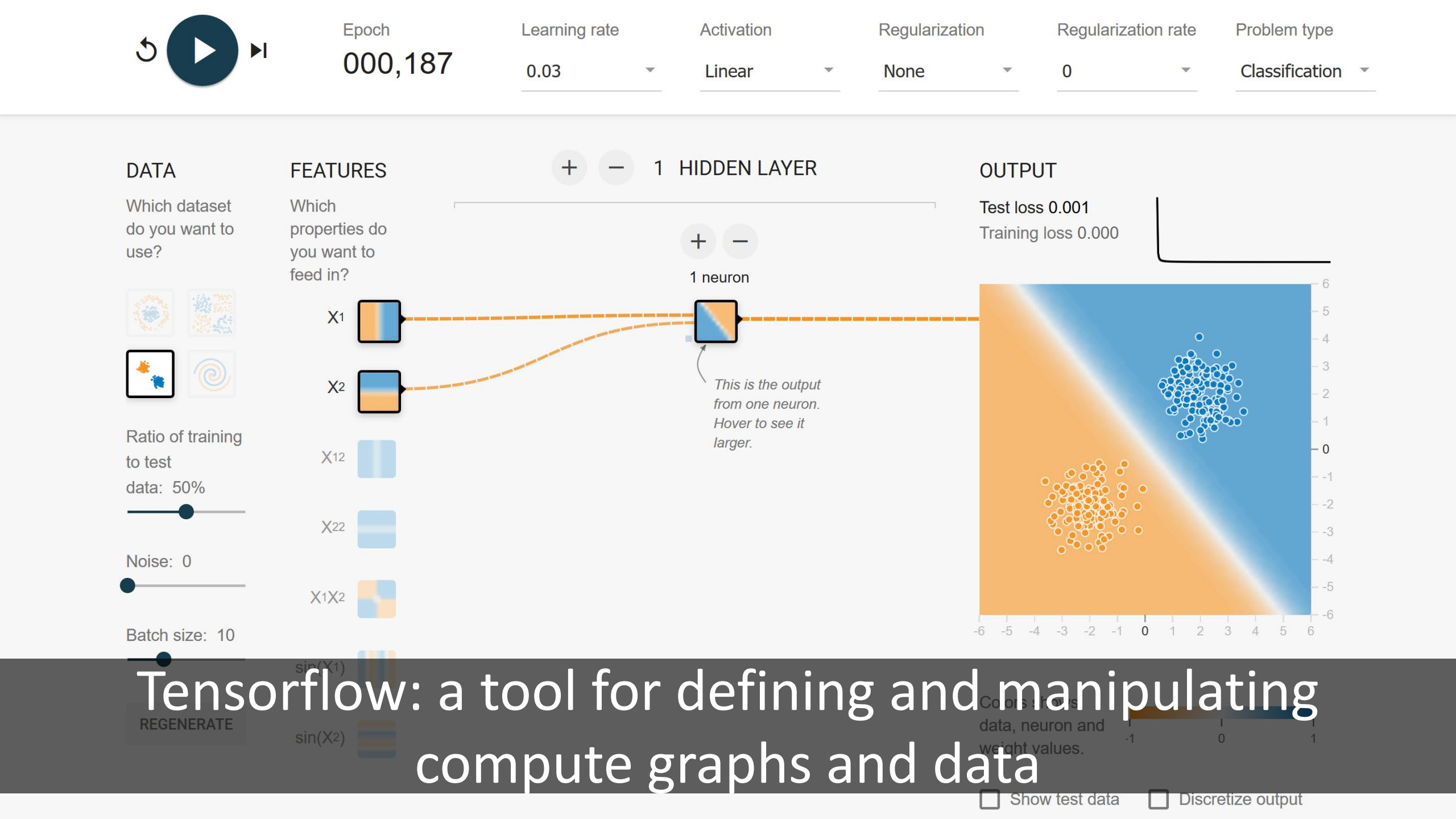
- Python network and file I/O
- Working with multidimensional number arrays using Python's Numpy package (basically all ML & AI builds on Numpy)
- Understanding what kinds of neural networks there are, what they can do, and what kind of data goes in and out



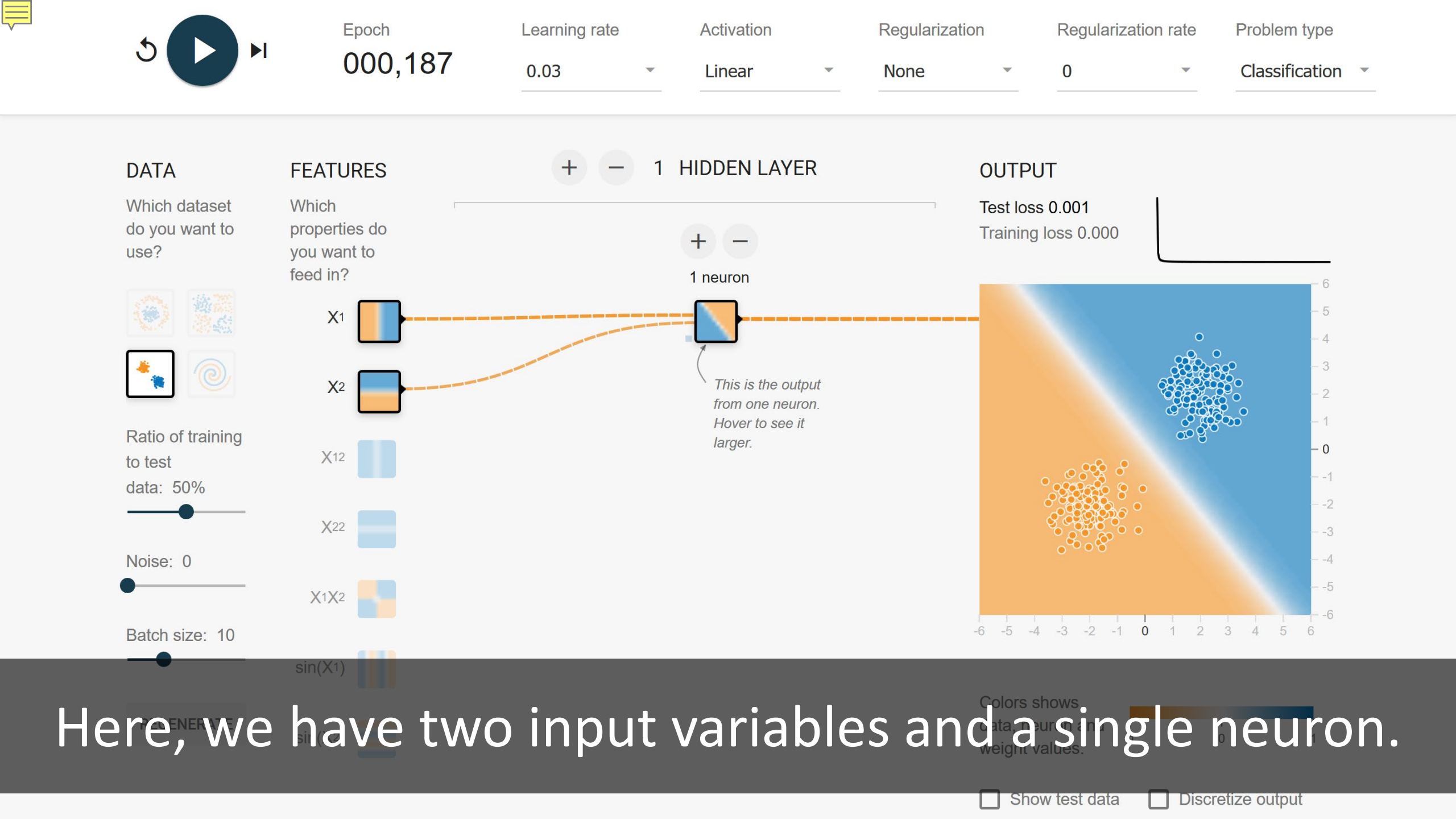
Tensorflow Playground: https://urly.fi/1cE1

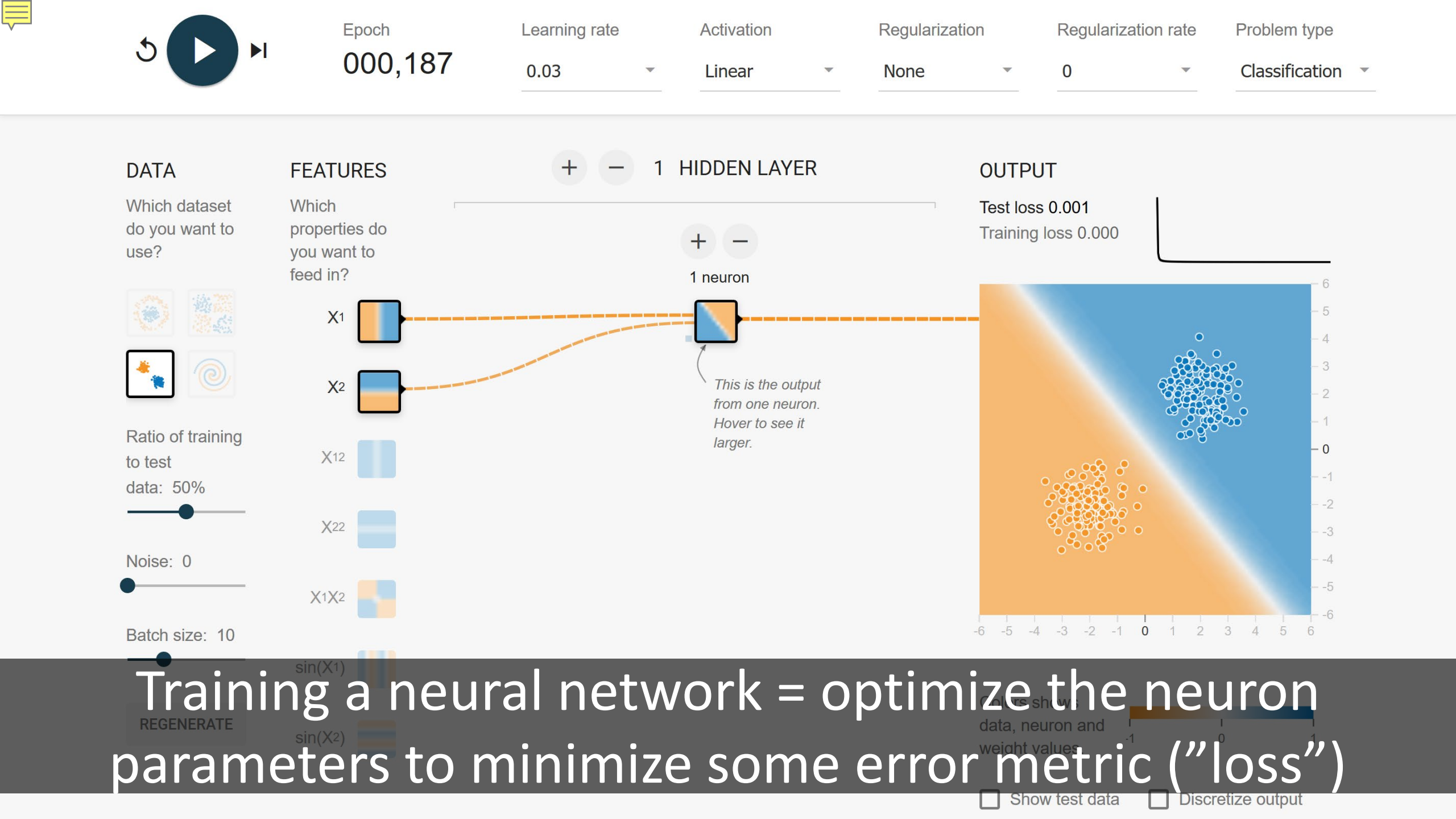


Neural networks are compute graphs, i.e.,
a set of operations through which data flows



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

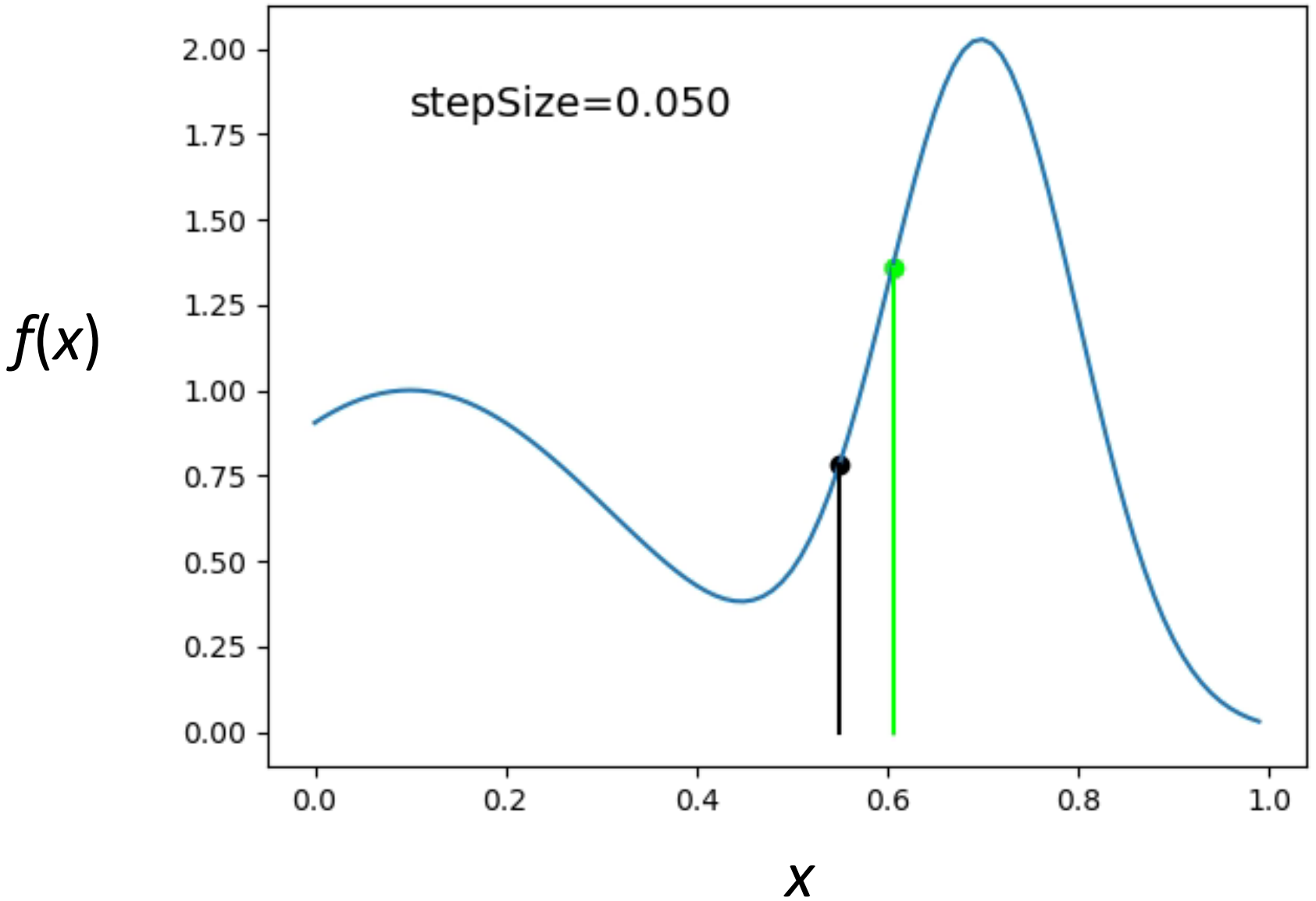

Epoch
000,187
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
X1
X2
X12
X22
X1X2
sin(X1)
1 HIDDEN LAYER
1 neuron
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.001
Training loss 0.000
Colors shows data, neuron and weight values.
Show test data
Discretize output
Here, we have two input variables and a single neuron.

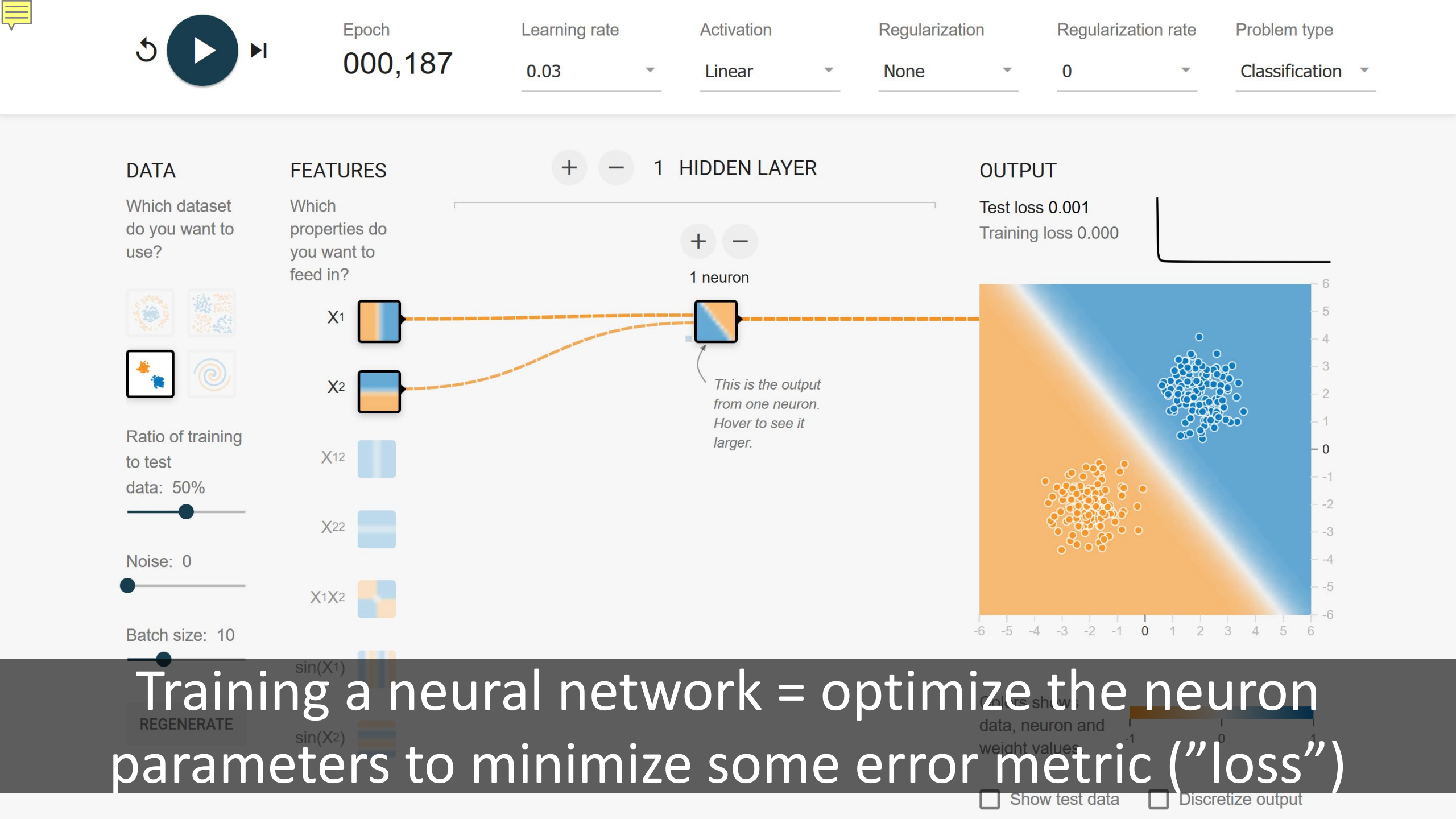

Epoch
000,187
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)
1 HIDDEN LAYER
1 neuron
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.001
Training loss 0.000
Colors shows data, neuron and weight values.
Show test data
Discretize output
Training a neural network = optimize the neuron parameters to minimize some error metric (”loss”)

All AI is (mathematical) optimization

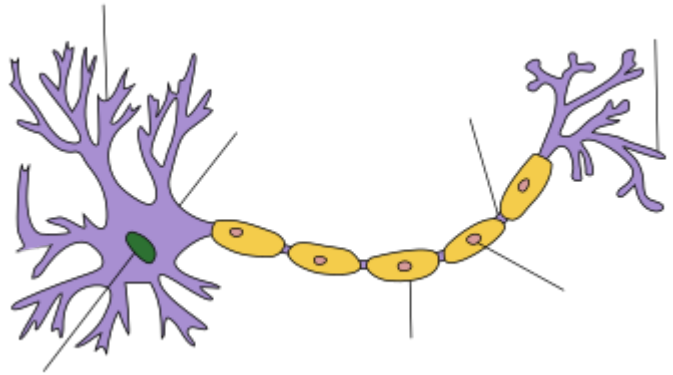
- Adjust some **x** to minimize or maximize some $f(\mathbf{x})$
- Usually, one denotes vectors with boldface and scalars with italic, i.e., $\mathbf{x}=[x_1,x_2,...]$
- Neural network training: **x** denotes network parameters, $f(\mathbf{x})$ is the loss or error function
- Pathfinding: **x** denotes path steps, $f(\mathbf{x})$ is path length
- Gameplay AI: **x** denotes actions, $f(\mathbf{x})$ is the utility or cost function
- Animation: **x** denotes muscle activations or other movement synthesis parameters, $f(\mathbf{x})$ 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)

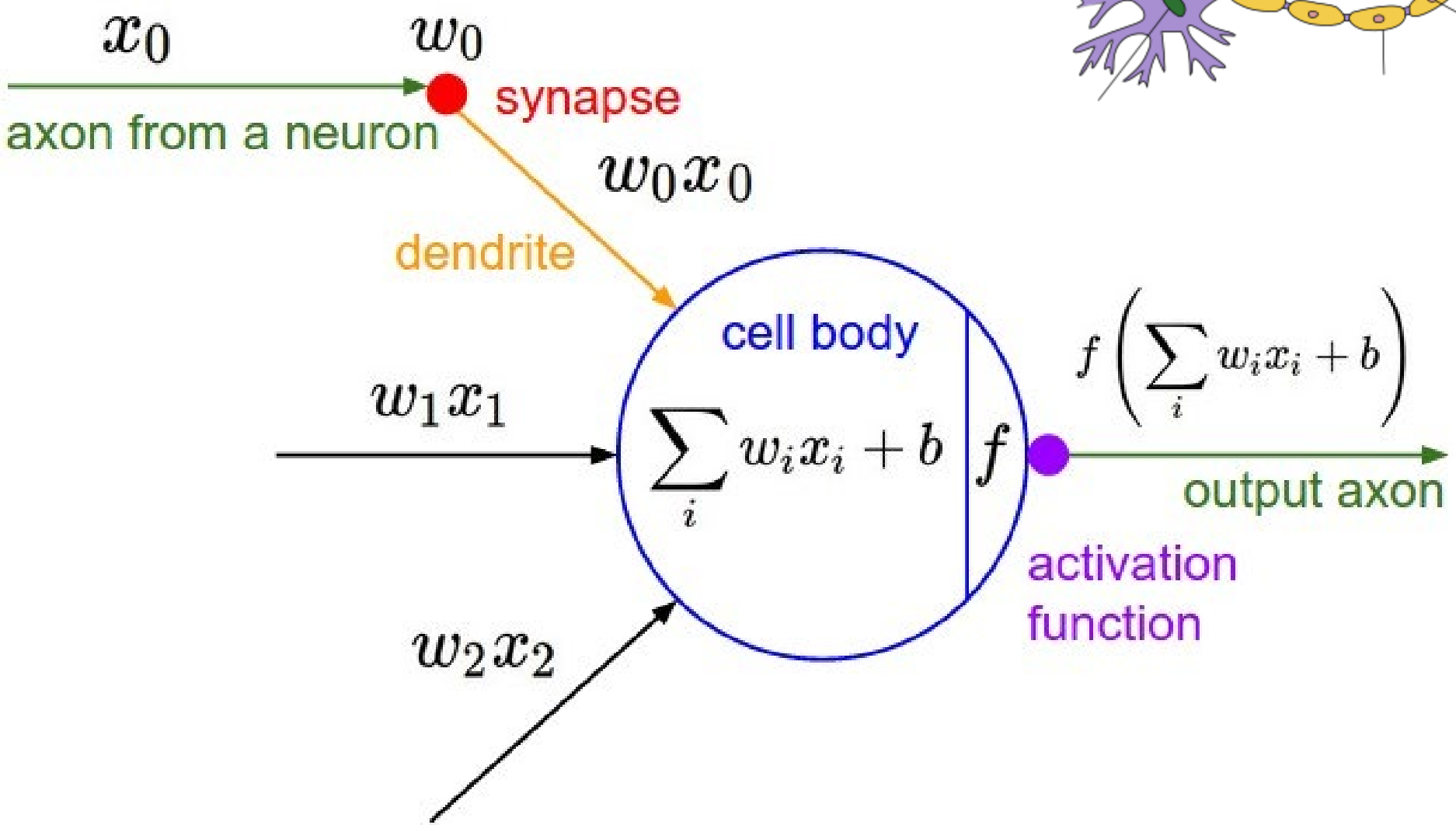
Optimization = exploring the $f(\mathbf{x})$ landscape



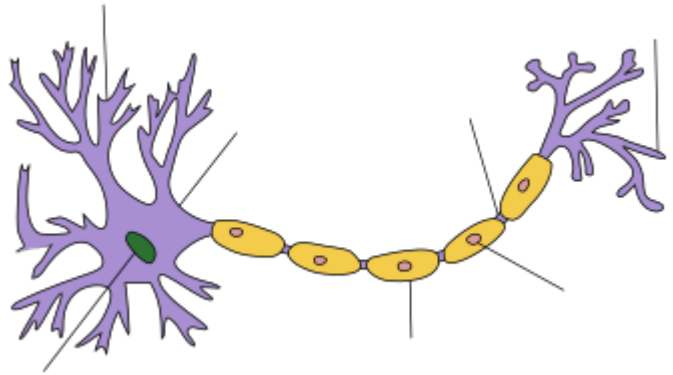

Epoch
000,187
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)
1 HIDDEN LAYER
1 neuron
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.001
Training loss 0.000
Colors shows data, neuron and weight values.
Show test data
Discretize output
Training a neural network = optimize the neuron parameters to minimize some error metric (”loss”)

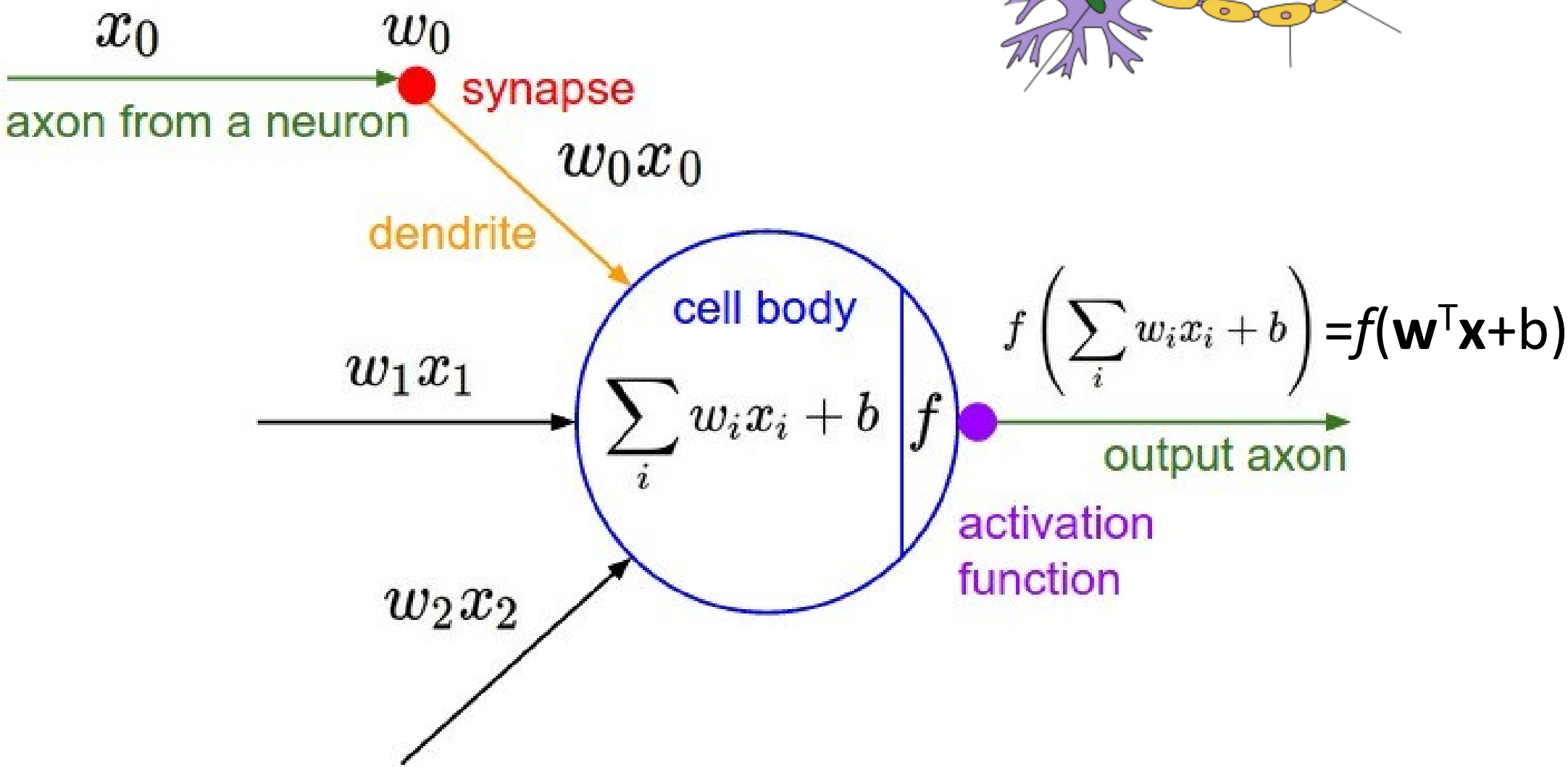
An artificial neuron



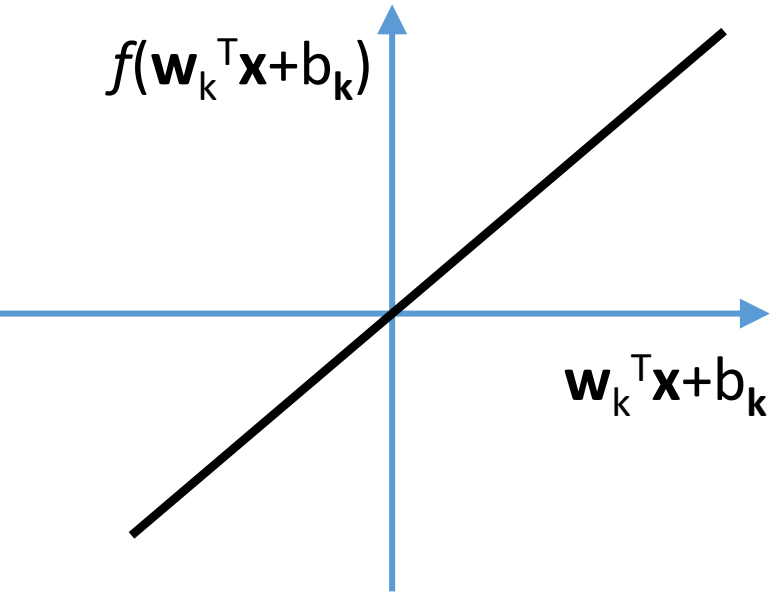


An artificial neuron

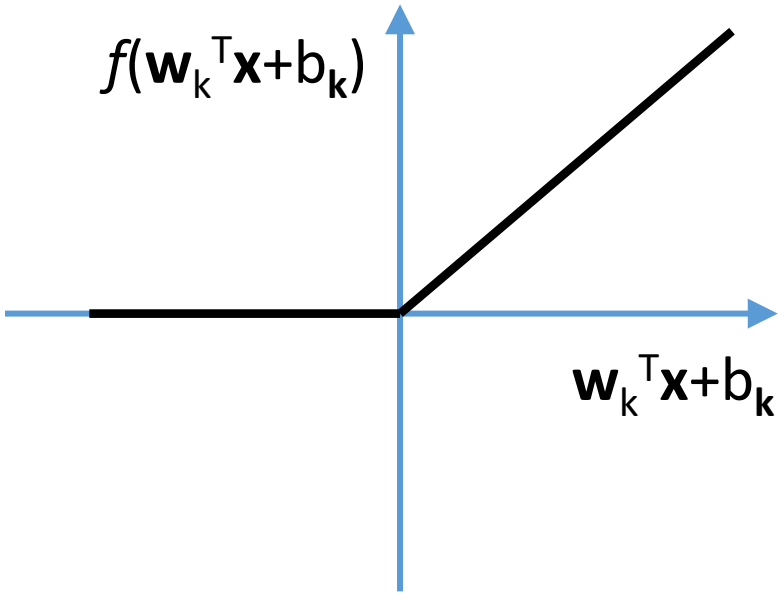




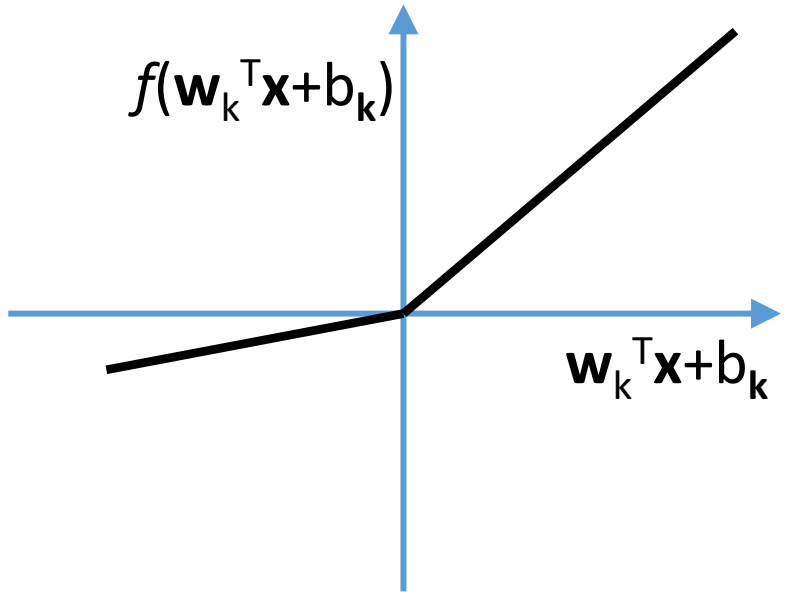
Linear activation



ReLU activation

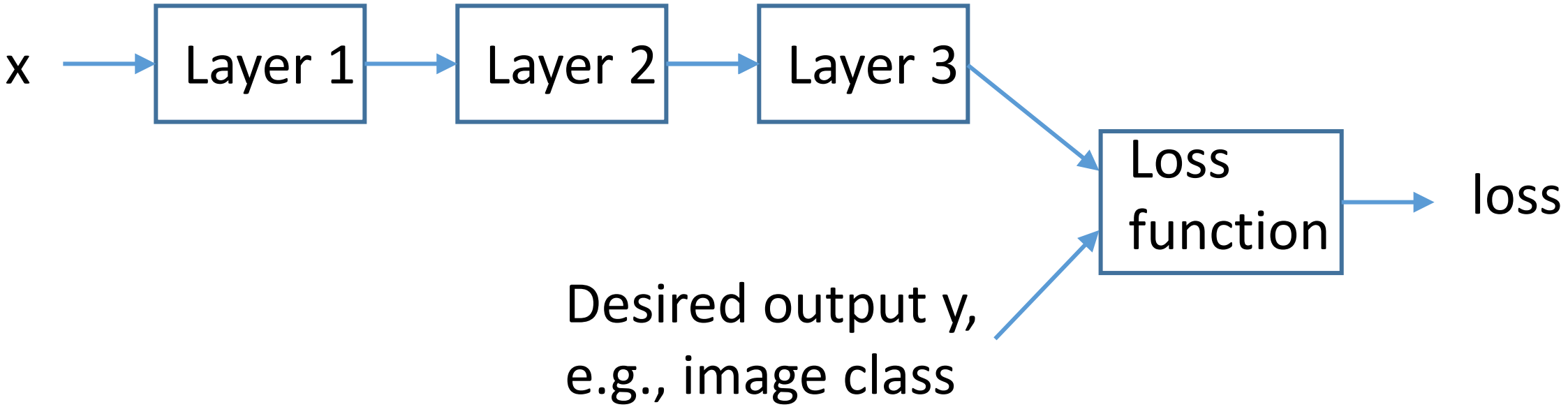


Leaky ReLU



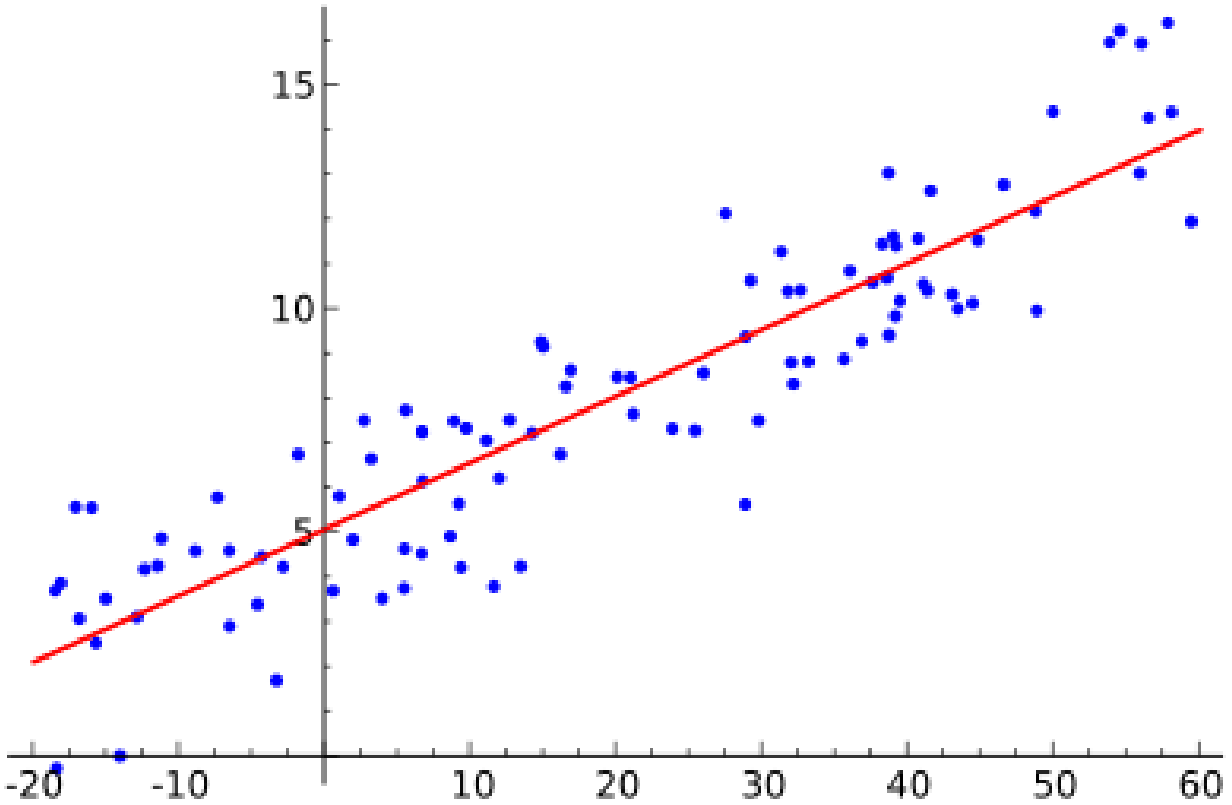
Training = optimization, minimizing loss

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

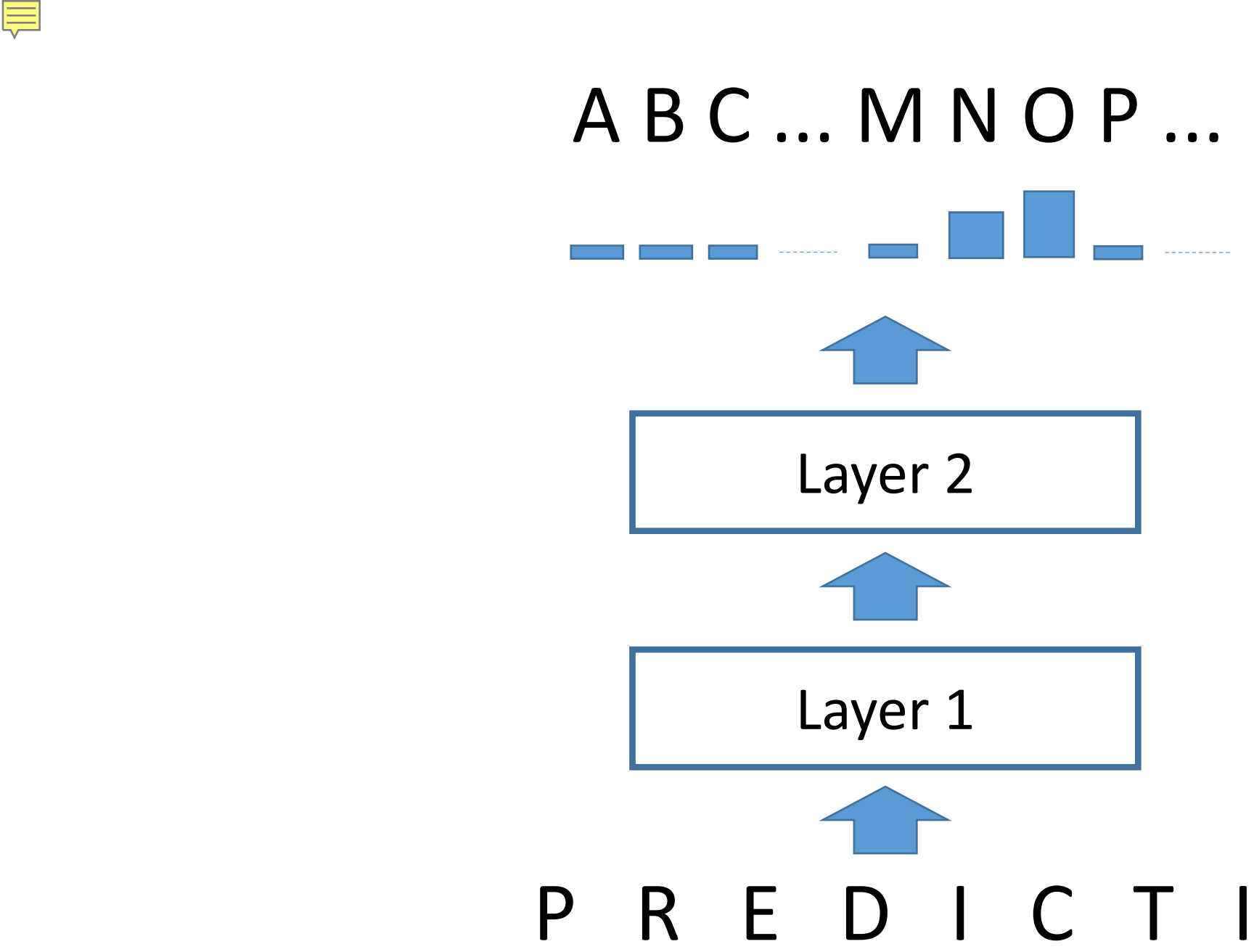
- Sum of squared errors, i.e., differences between network output variables and desired output variables



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**
- 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?

- X_1
- X_2
- X_1^2
- X_2^2
- X_1X_2
- $\sin(X_1)$
- $\sin(X_2)$

1 HIDDEN LAYER

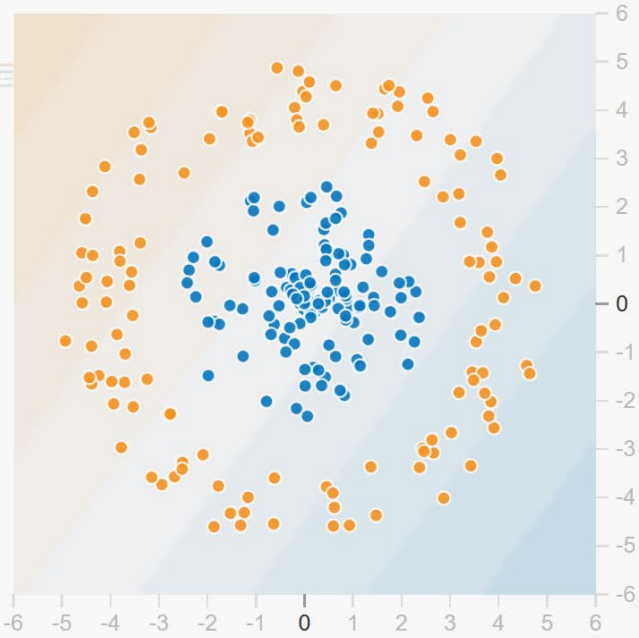
4 neurons

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?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

- X_1
- X_2
- X_1^2
- X_2^2
- X_1X_2
- $\sin(X_1)$
- $\sin(X_2)$

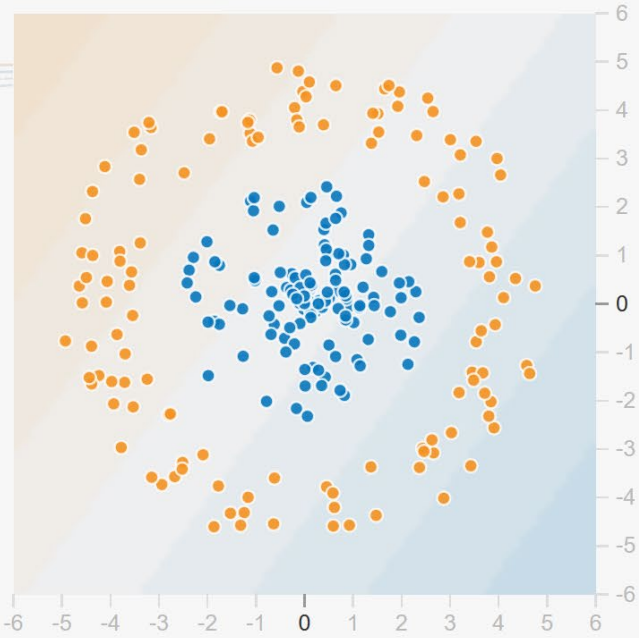
1 HIDDEN LAYER

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
000,124

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?

- X_1
- X_2
- X_1^2
- X_2^2
- X_1X_2
- $\sin(X_1)$
- $\sin(X_2)$

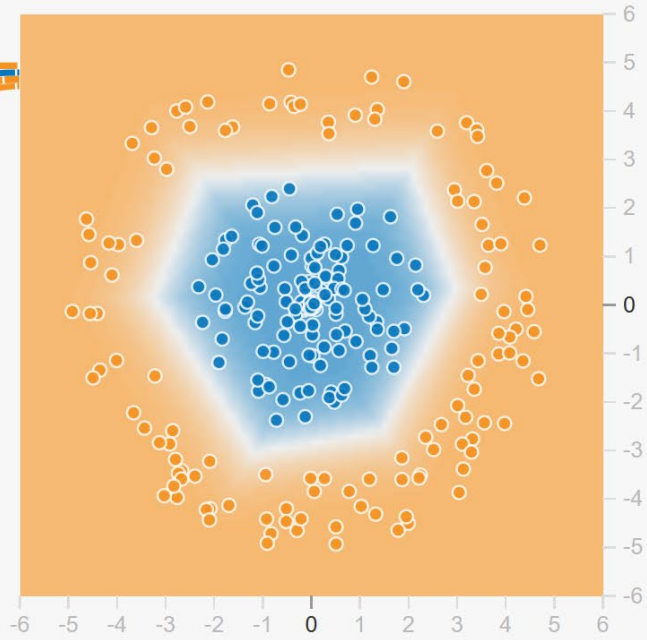
1 HIDDEN LAYER

4 neurons

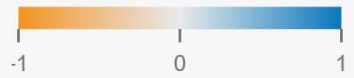
This is the output from one neuron. Hover to see it larger.

OUTPUT

Test loss 0.025
Training loss 0.013



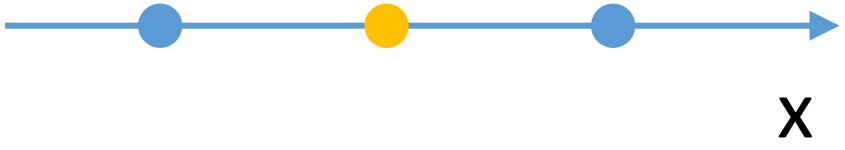
Colors shows data, neuron and weight values.



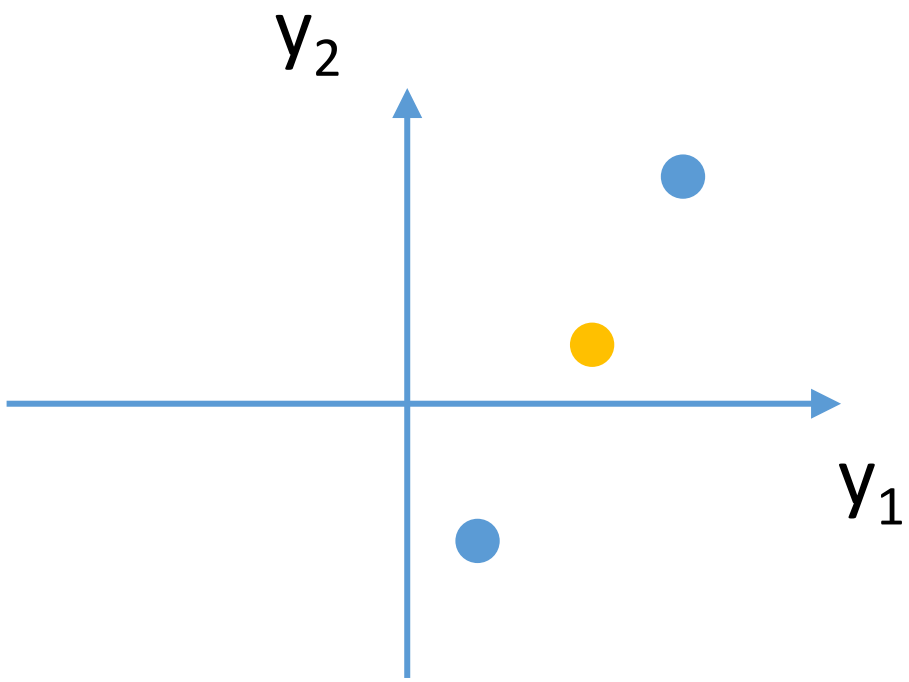
Show test data Discretize output

The effect of nonlinearity

1D input

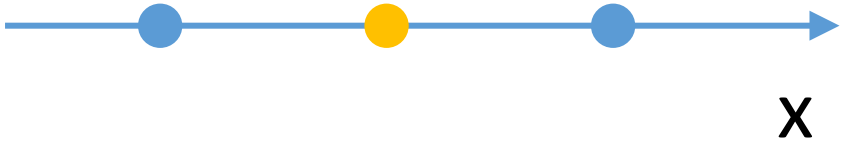


2D output of 2 linear neurons



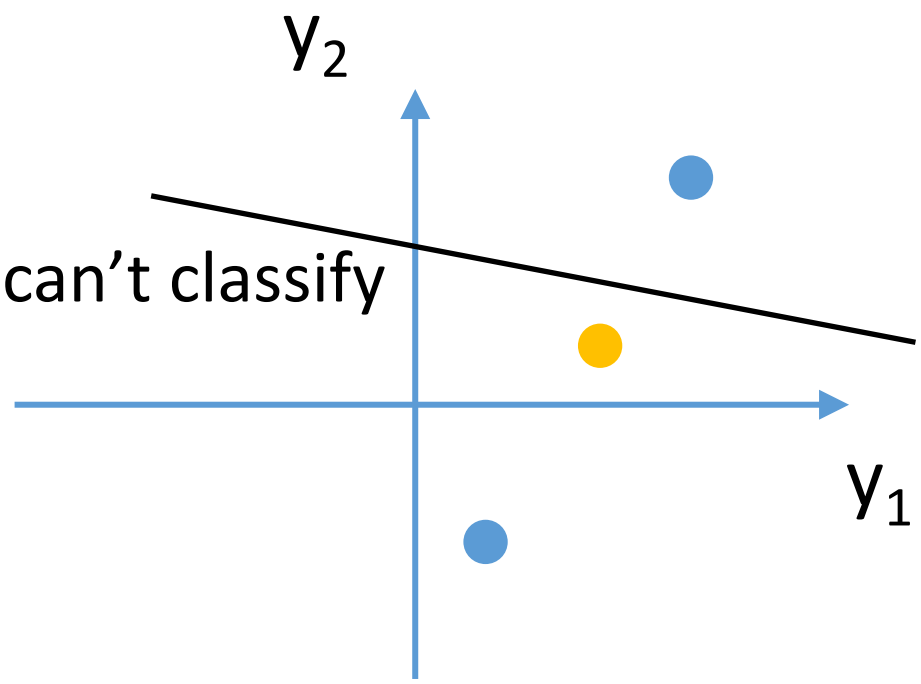
The effect of nonlinearity

1D input



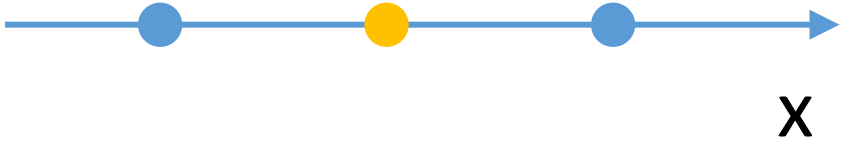
Output neuron can’t classify

2D output of 2 linear neurons

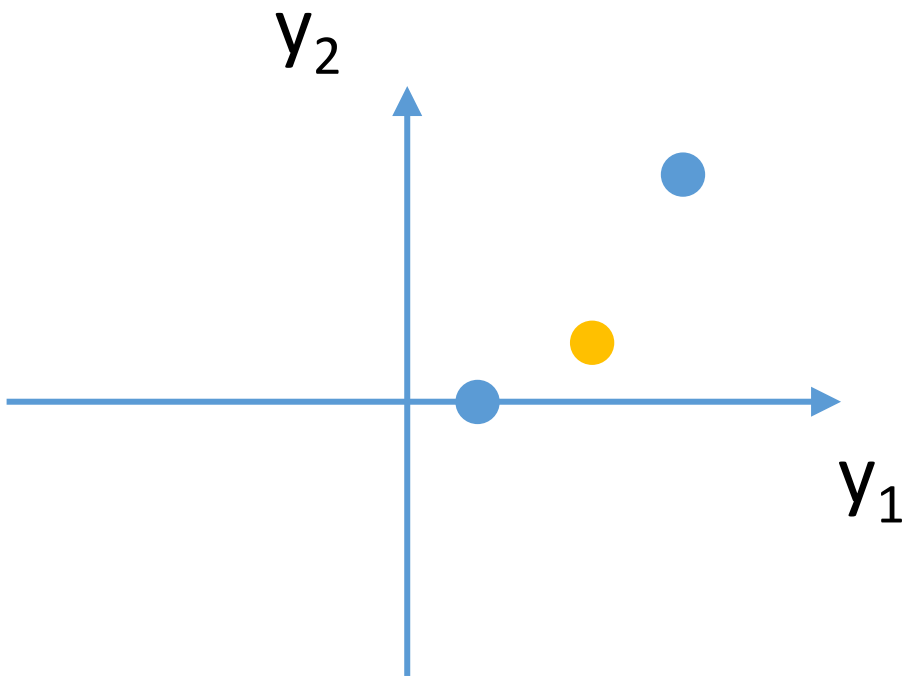


The effect of nonlinearity

1D input

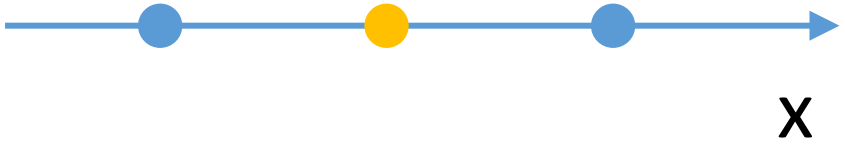


2D output of 2 RELU neurons

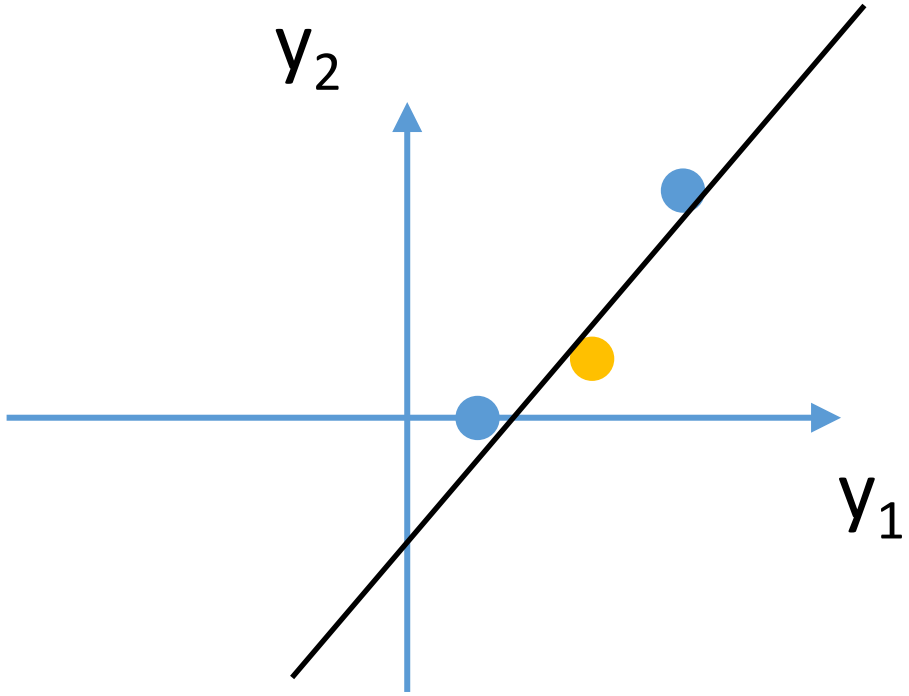


The effect of nonlinearity

1D input



2D output of 2 RELU neurons

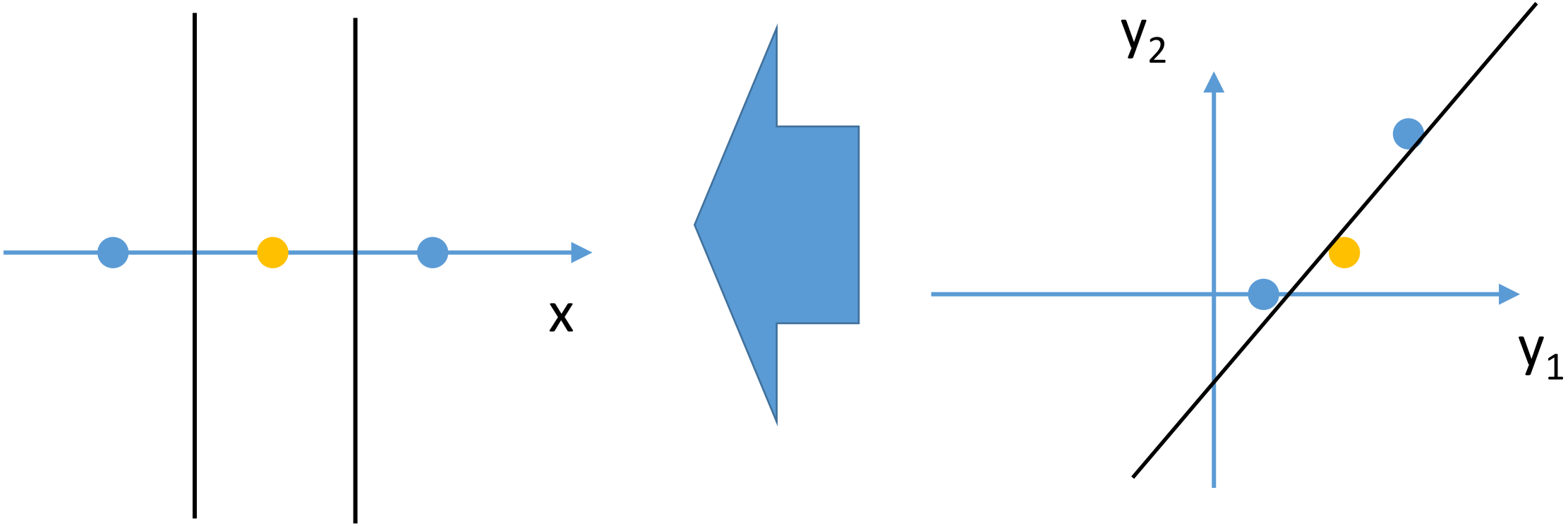


Linear split works now!

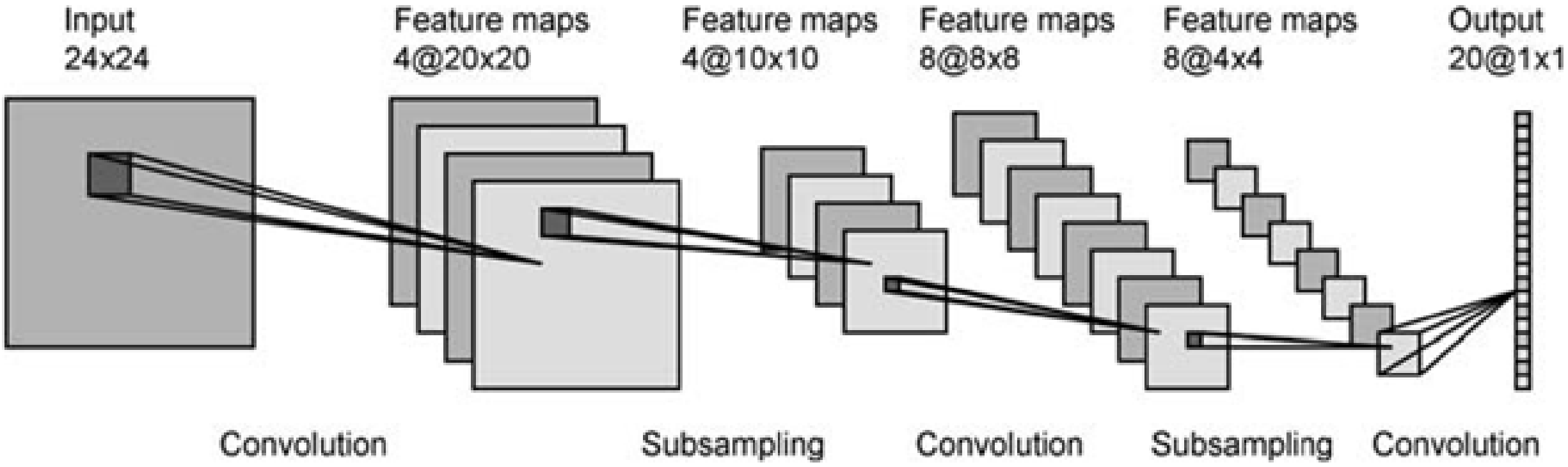
Split into two halves in a high-dimensional representation = multiple splits in the input space

1D input

2D output of 2 RELU neurons



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?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

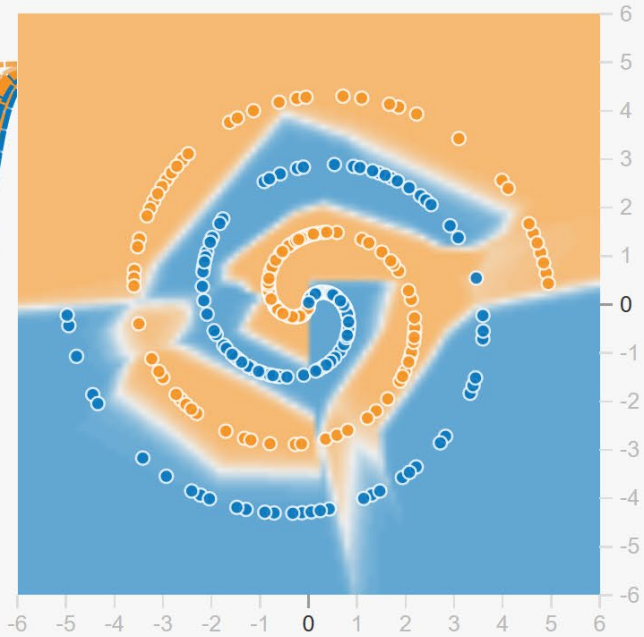
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)

3 HIDDEN LAYERS

8 neurons | 8 neurons | 8 neurons

OUTPUT

Test loss 0.068
Training loss 0.010

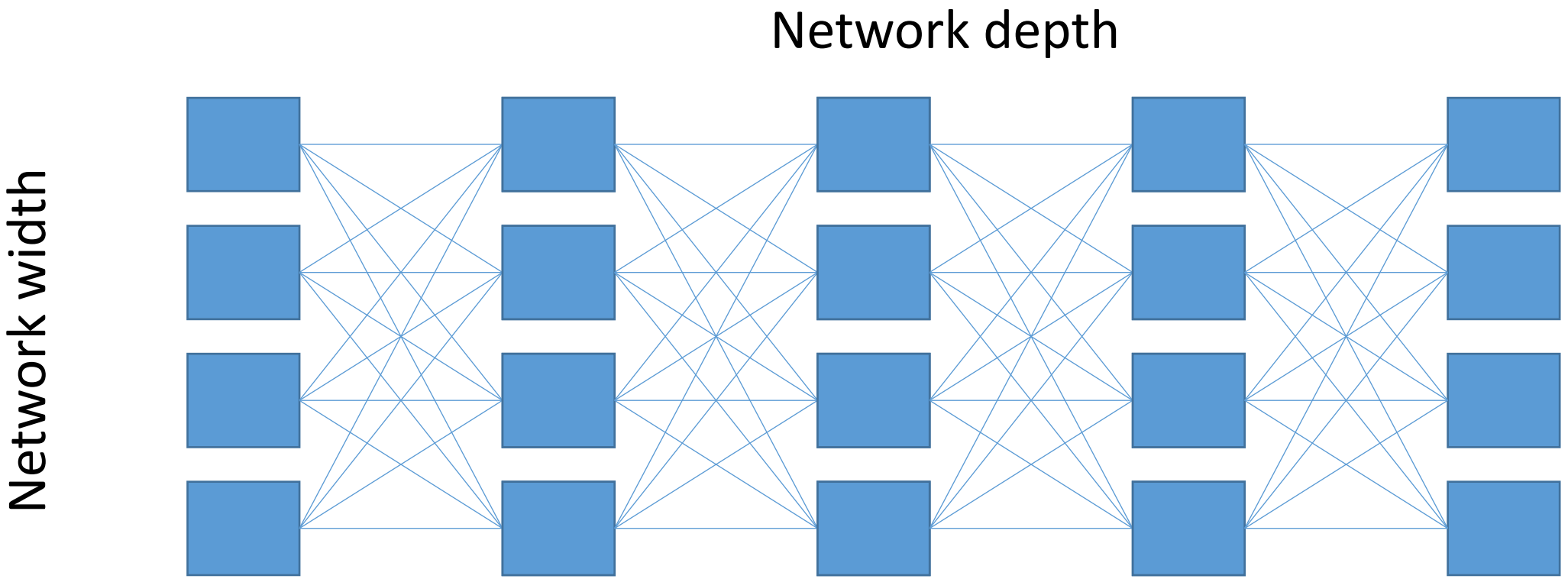


Colors shows data, neuron and weight values.

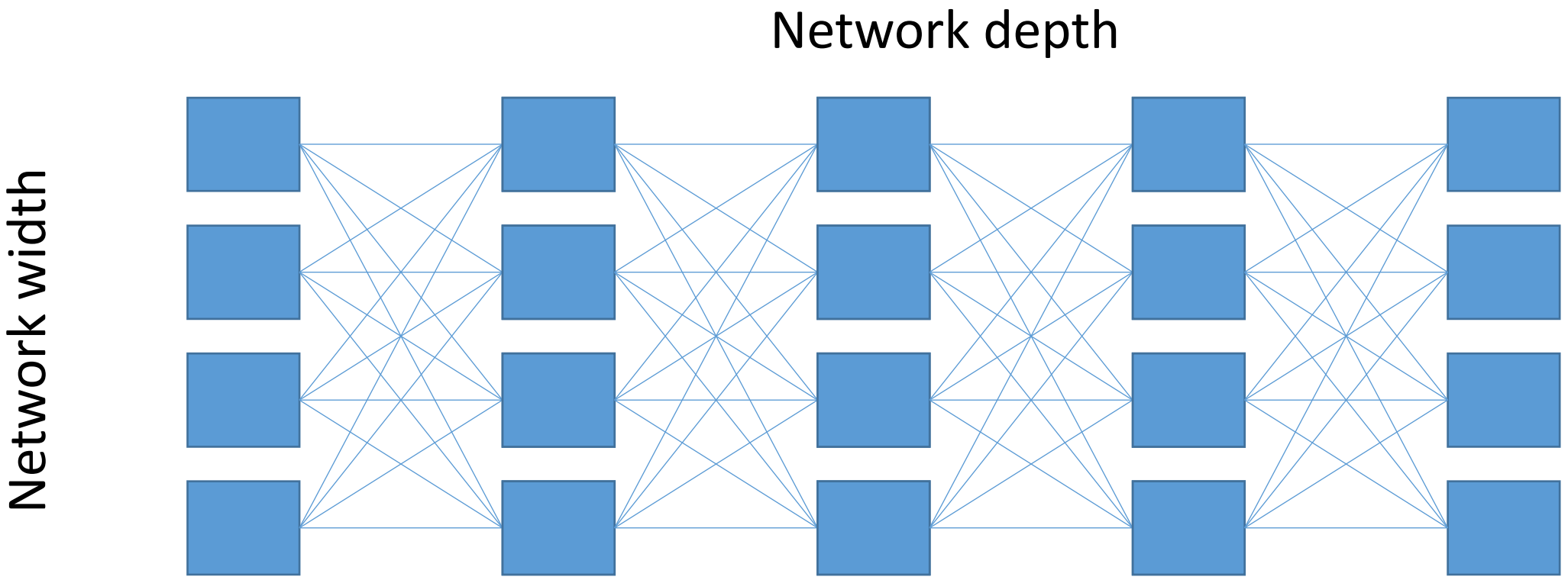
-1 0 1

Show test data Discretize output

Intuition on the power of depth



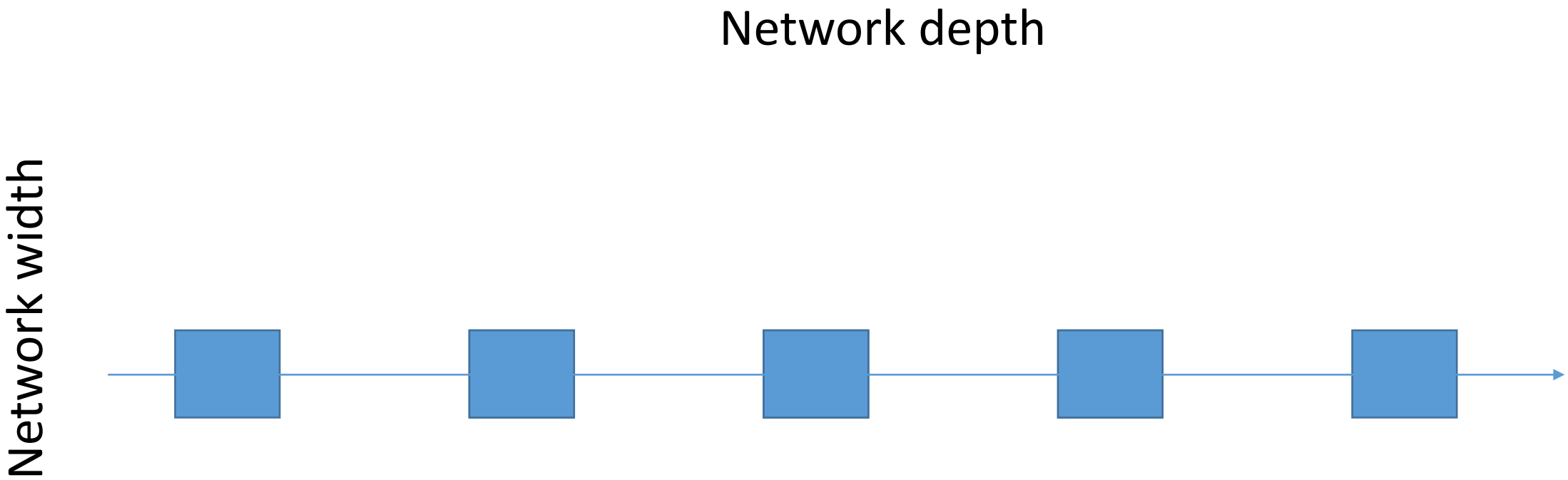
Intuition on the power of depth



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

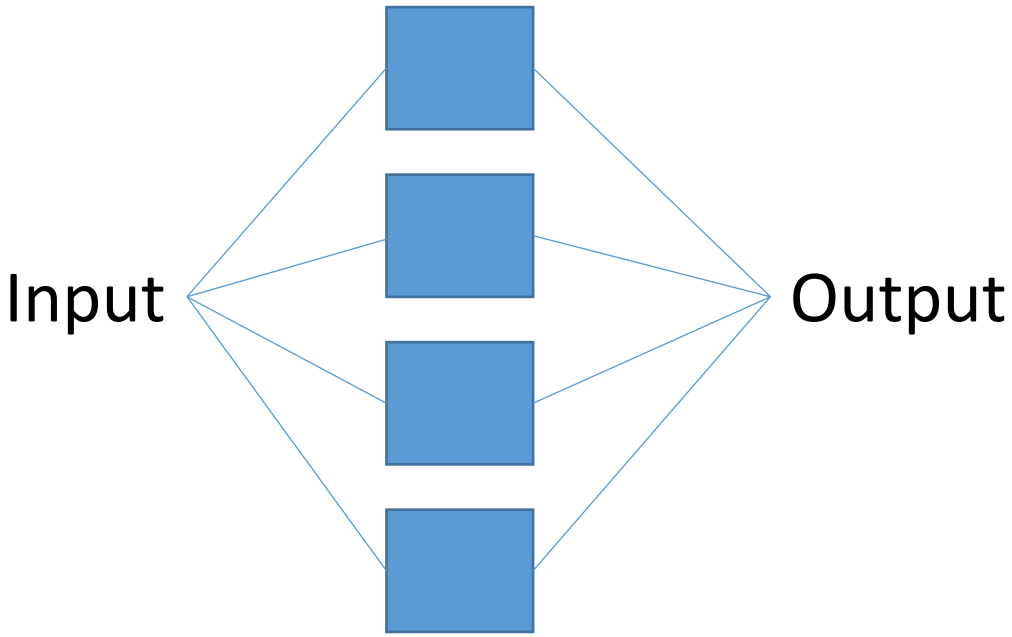
How many paths can the data take through the network?

Intuition on the power of depth



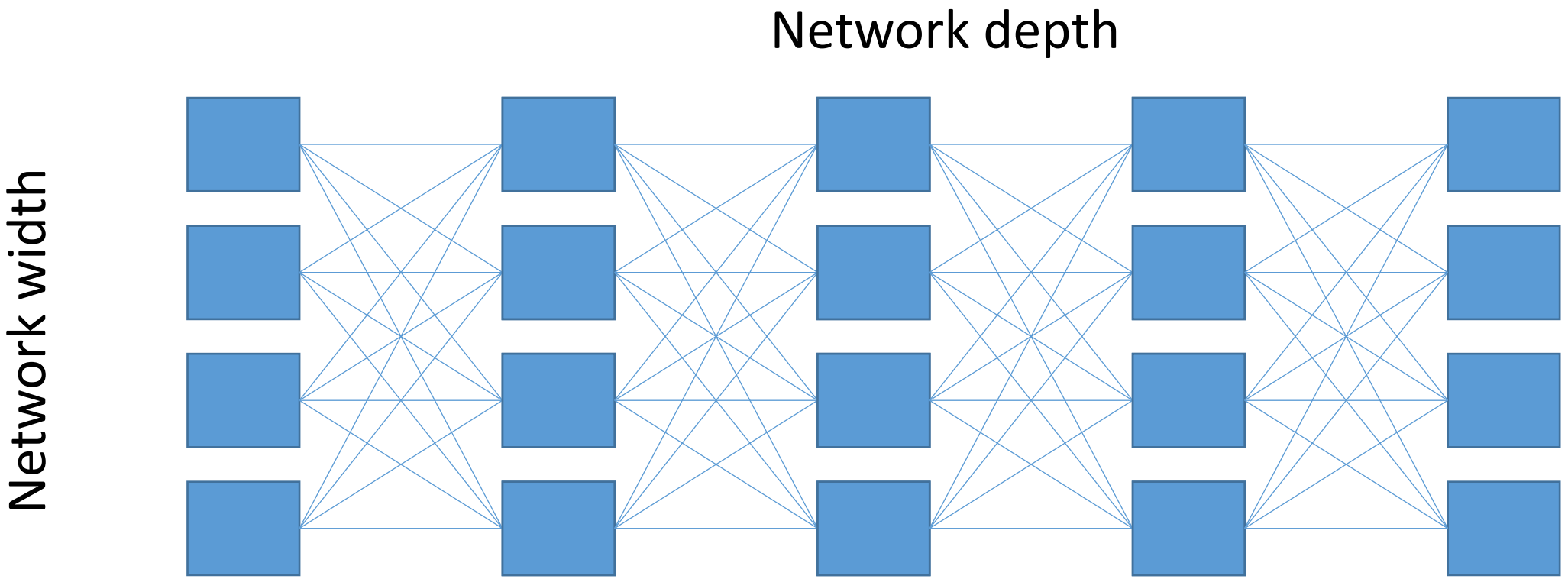
Narrow but deep network: only one path

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.

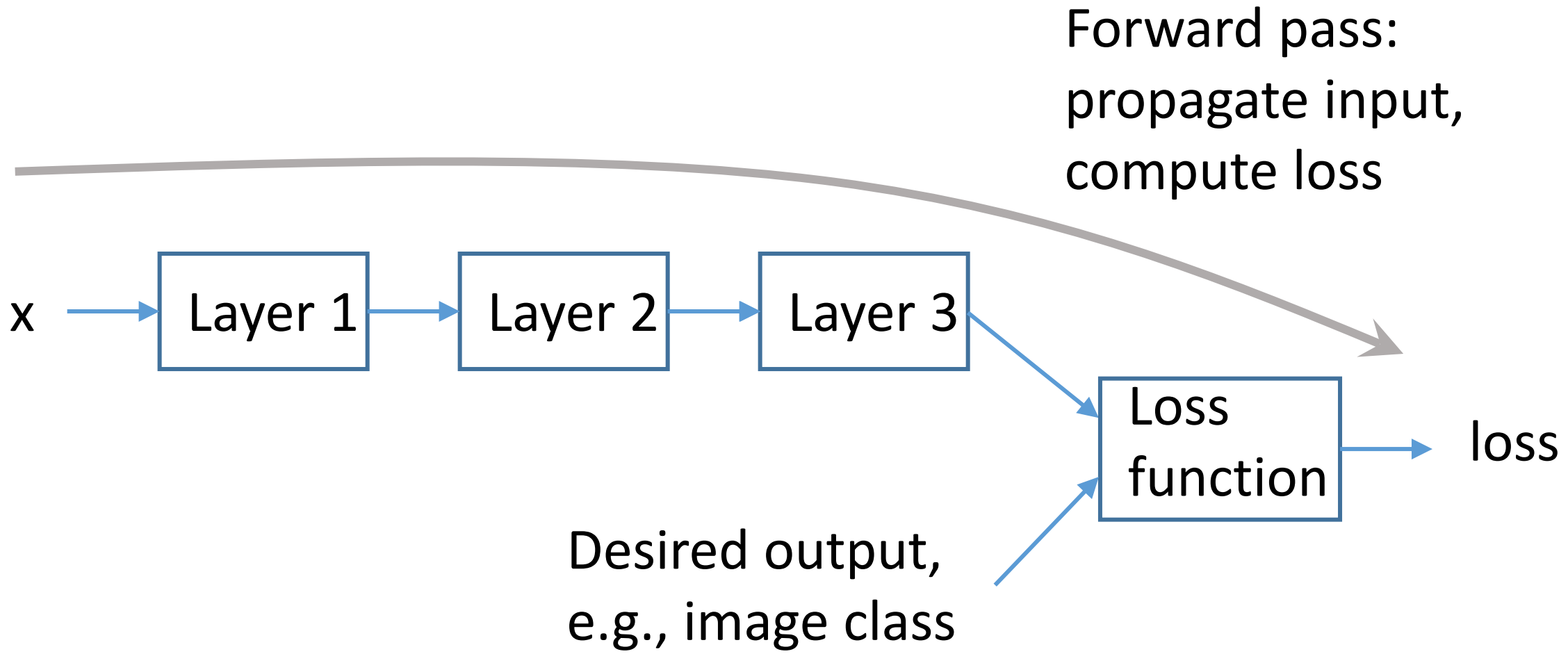
Contents

- 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

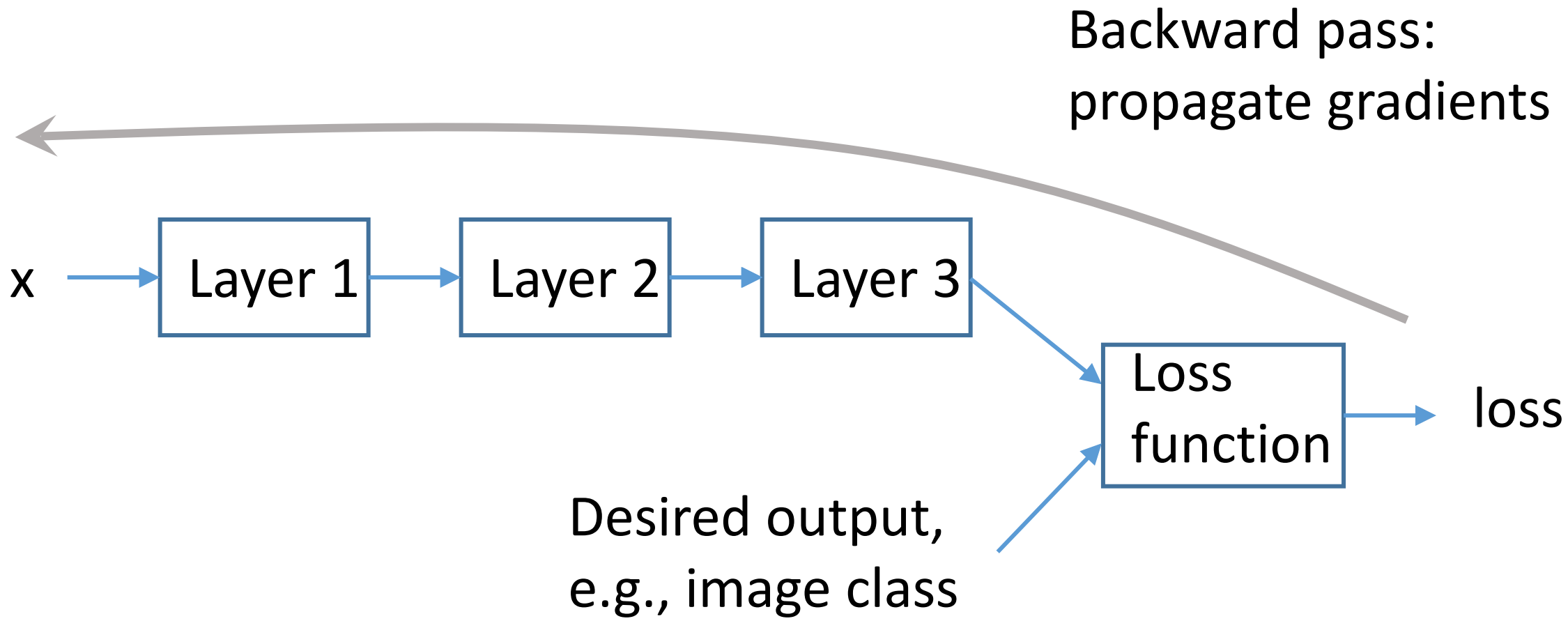
Problems with adding layers

- In practice, optimization uses gradients (more about that in the optimization lecture!)
- Gradients may vanish & explode with deep networks => optimization does not converge
- This is because gradient computation multiplies together matrices from all the layers (backpropagation, i.e., chain rule of calculus)

Backpropagation



Backpropagation

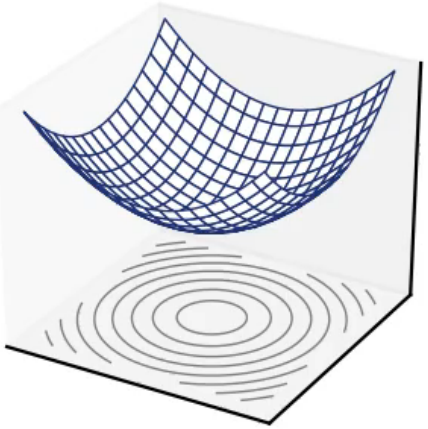


Compute graphs and gradients

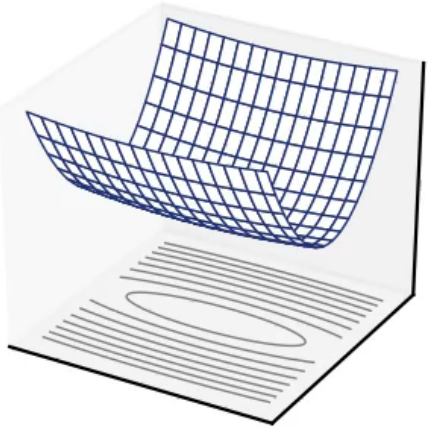
- Key: every operation is differentiable, i.e., can compute gradient of output w.r.t. params or input
- Gradient = which direction to nudge parameters to improve the loss function?
- *Tensorflow etc. are tools for building compute graphs out of operations that can automatically compute their gradients*
- Optimization depends on gradients because the high number of parameters makes random and/or global search for the optimum infeasible

Types of optimization problems

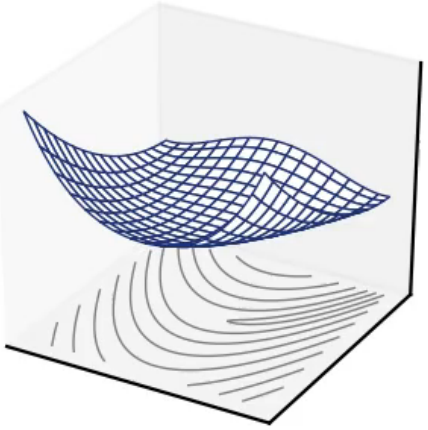
Convex,
well-conditioned



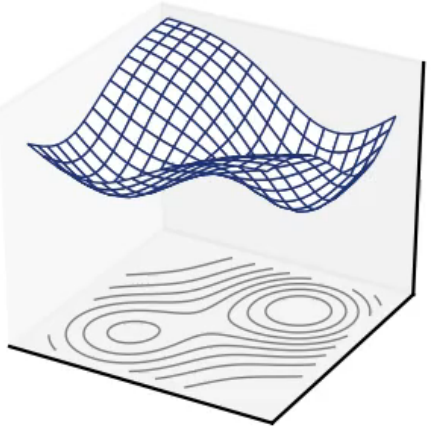
Convex,
ill-conditioned



Non-convex,
unimodal

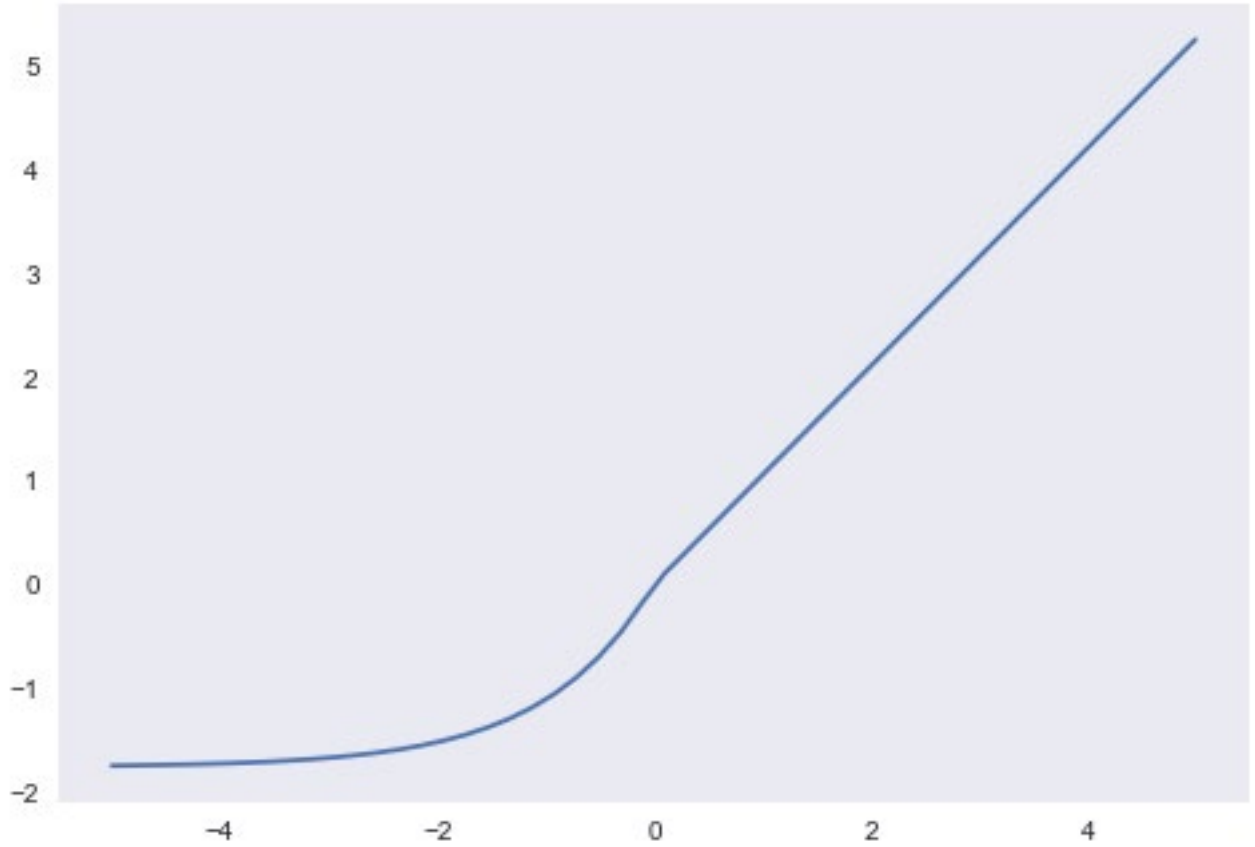


Non-convex,
multimodal



Solutions to gradient problems

- SELU activation function (self-normalizing signals to zero-mean unit variance)
- Skip-connections



Skip-connections: the ResNet

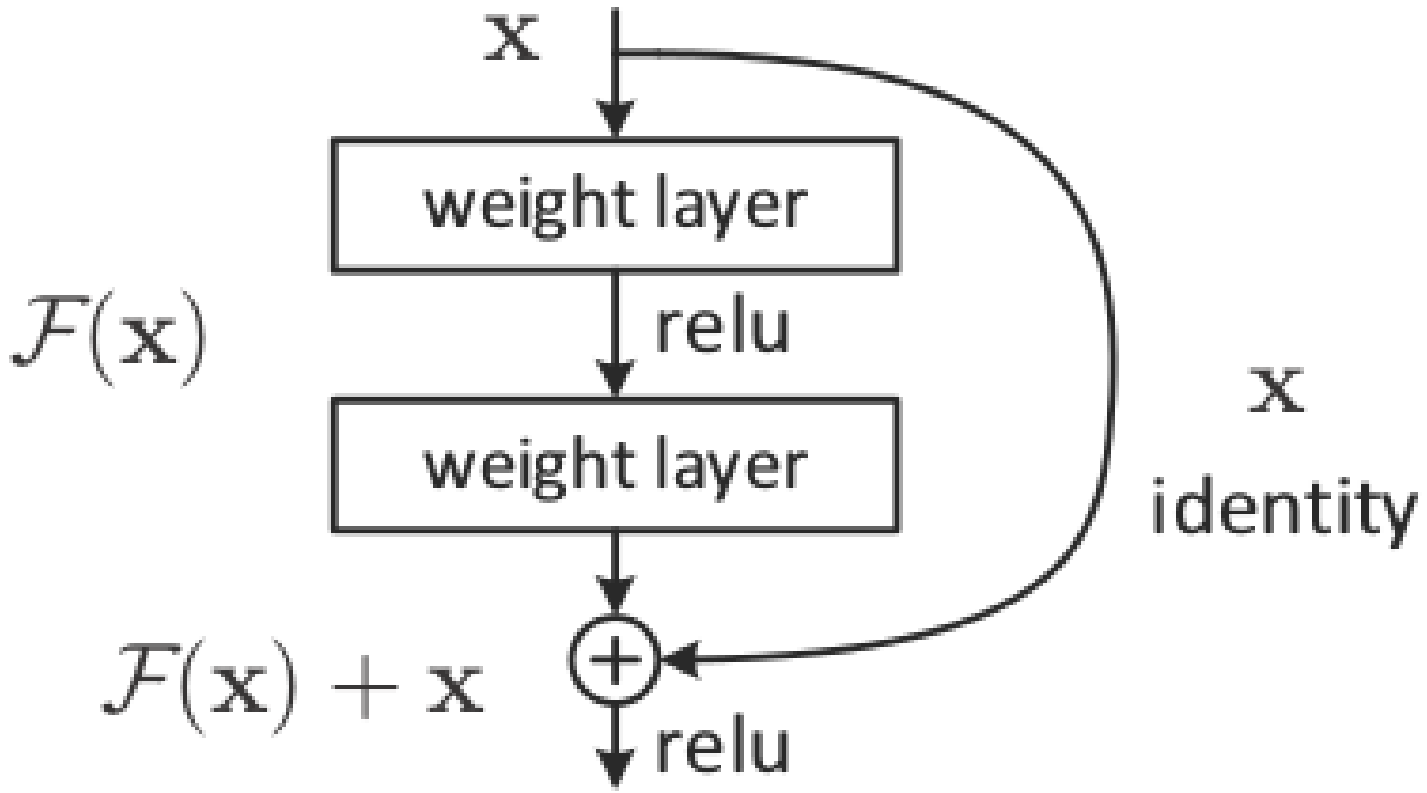


Figure 2. Residual learning: a building block.

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [40] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test *set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.*

The depth of representations is of central importance

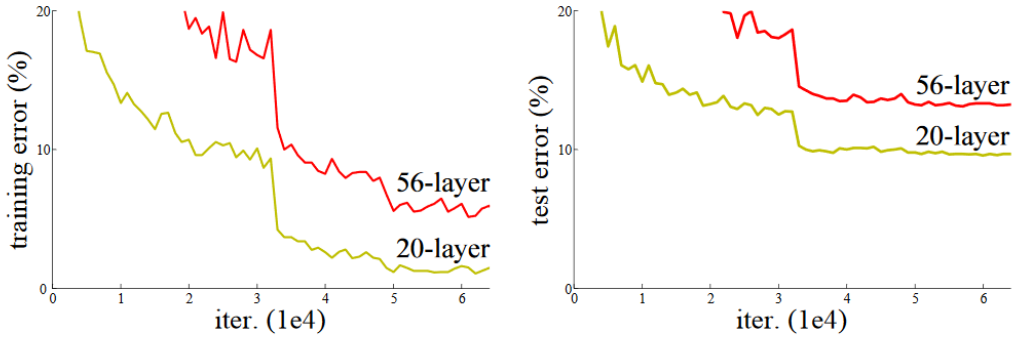


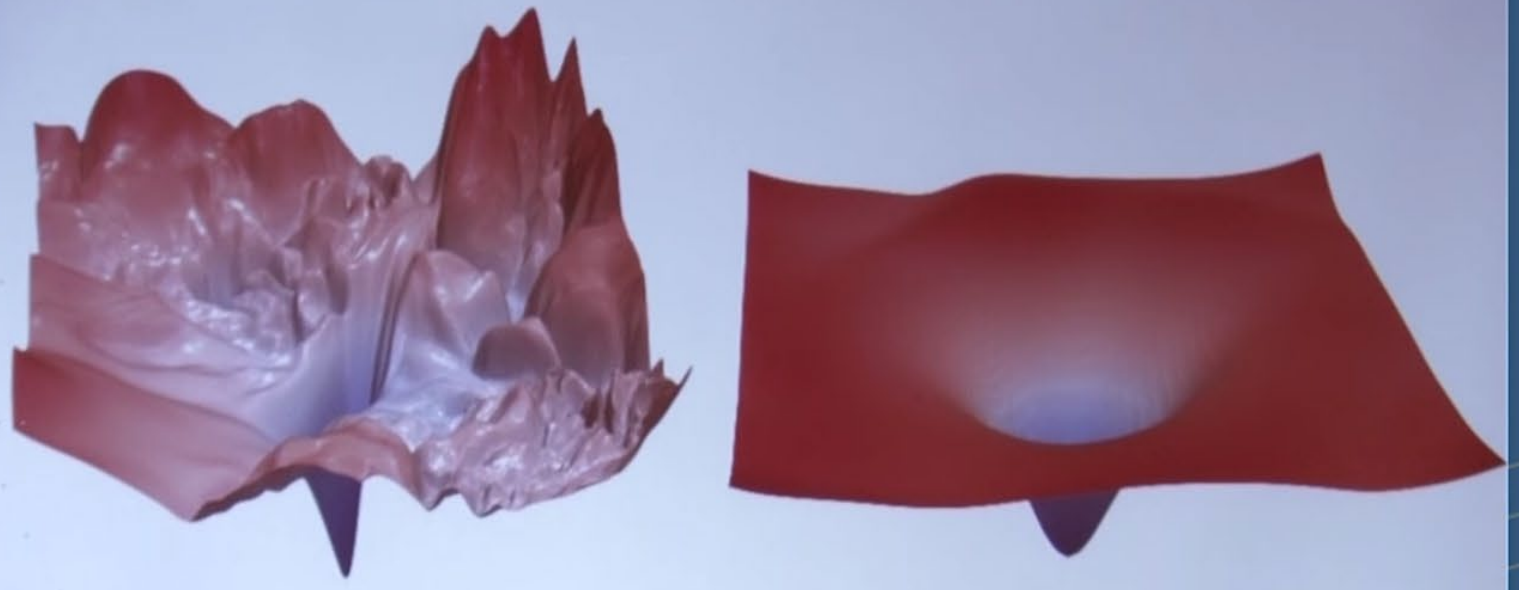
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [14, 1, 8], which hamper convergence from the beginning. This problem,

56 LAYER NEURAL NET

CIFAR-10

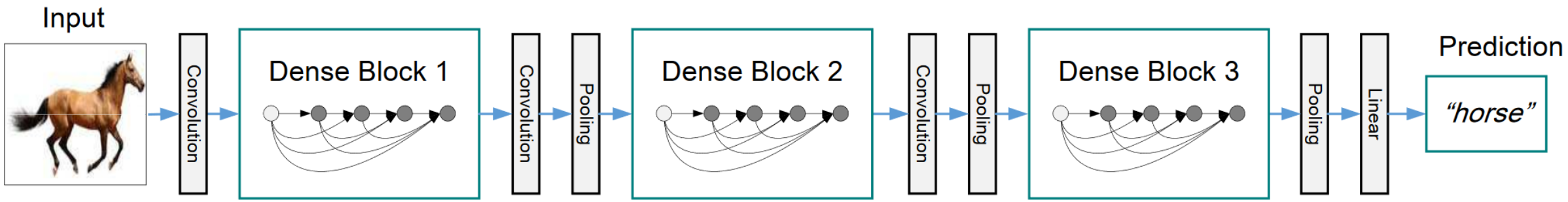


VGG-like

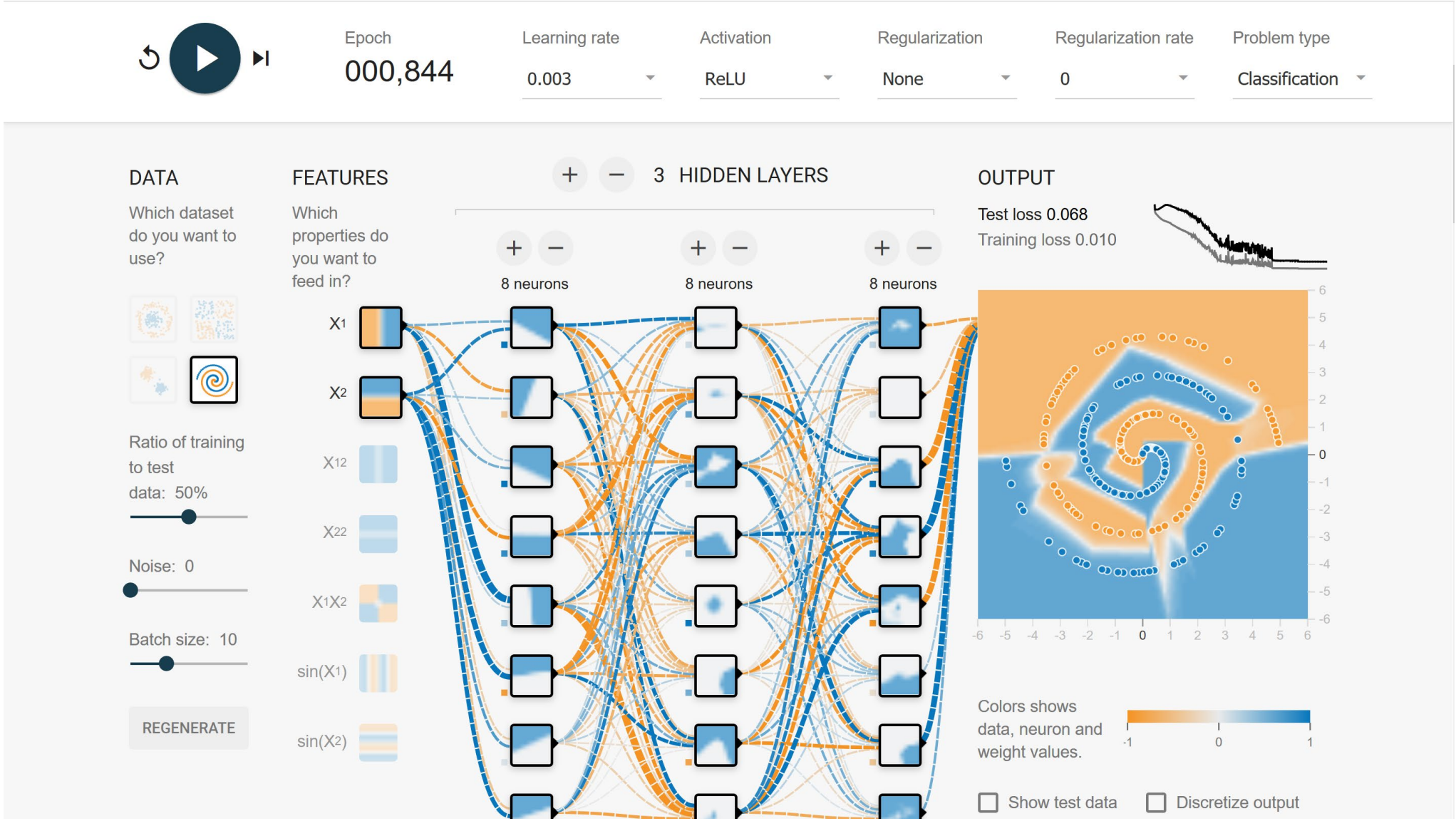
skip connections

ResNet

The DenseNet



This is deprecated for deep networks



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

Fully connected networks are often inoptimal

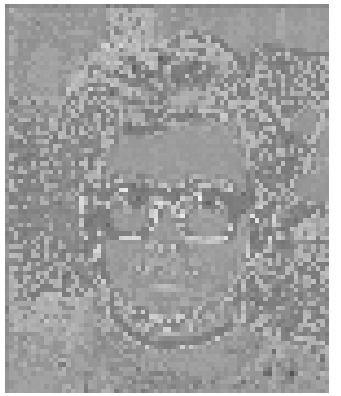
- Lots of parameters, slow to optimize
- Assumes that all variables are interrelated, or at least assumes that all interrelations are equally probable
- In real data, relations are often local, e.g., nearby pixels have stronger relationships
- Convolutional networks to the resque! (less parameters, assume locality)

Convolution

- Image blurring or edge detection is ”convolution” with a fixed filter
- Pixel *i* output = $\mathbf{w}^T\mathbf{x}_i$, where $\mathbf{w}$ is the filter kernel and $\mathbf{x}_i$ is an input image patch around the pixel. Filter kernel = neuron parameters
- A convolutional neural network learns such filters that help in minimizing the loss function.

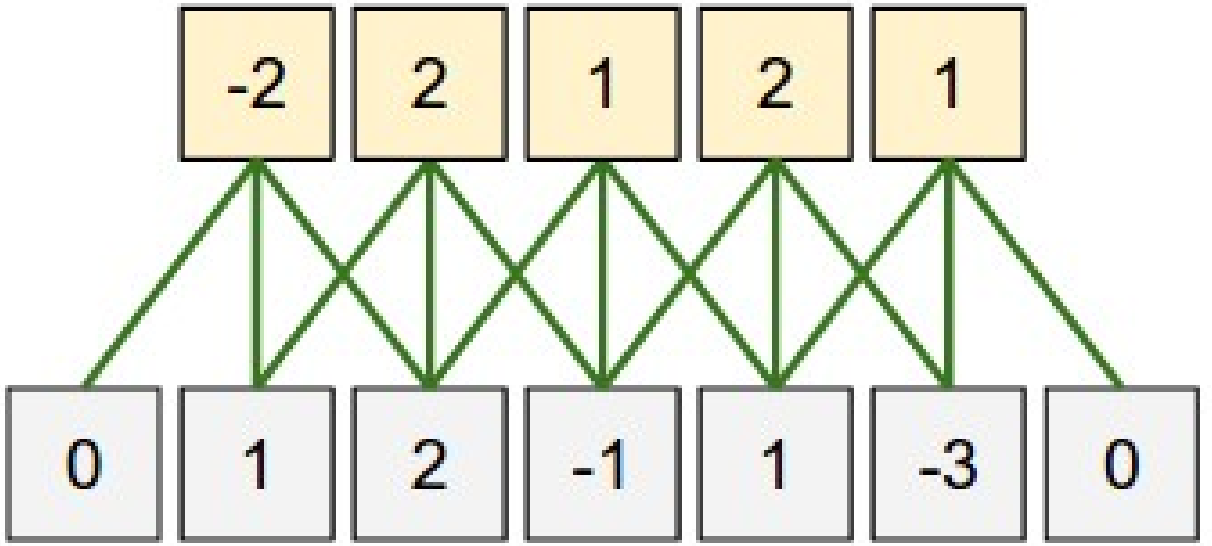
Conv with Laplacian filter

-1	-1	-1
-1	8	-1
-1	-1	-1



Convolution as a hierarchy, trees

Outputs of this layer's neurons



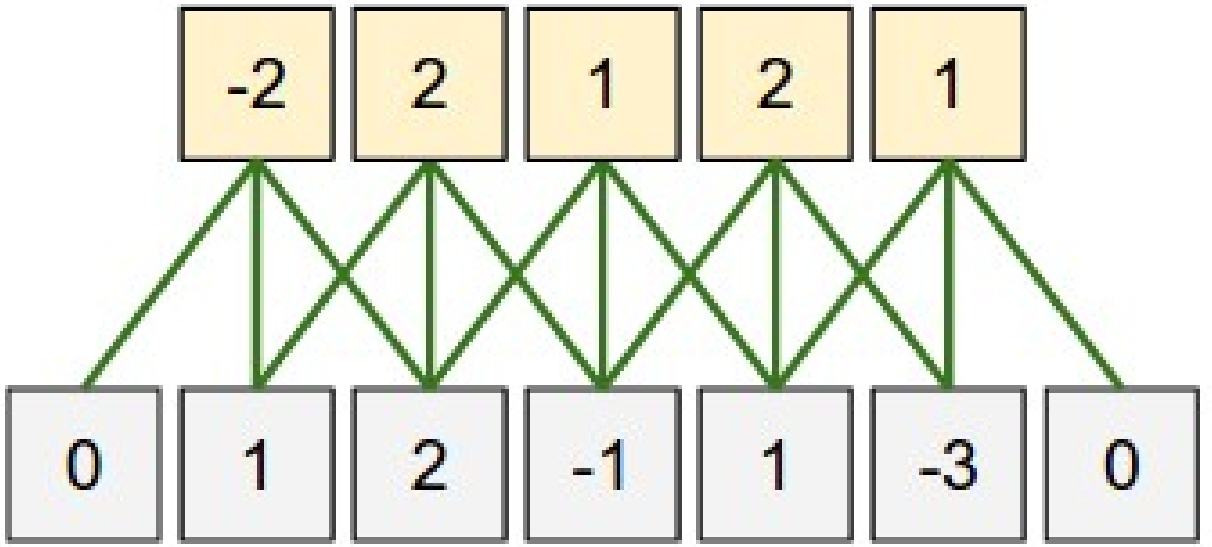
w=

1	0	-1

Inputs (image pixels, or outputs of each previous layer neuron)

Convolution as a hierarchy, trees

Convolutional networks: The neurons for each output pixel have the same weights. Implementation: a single filter sweeps over the input array



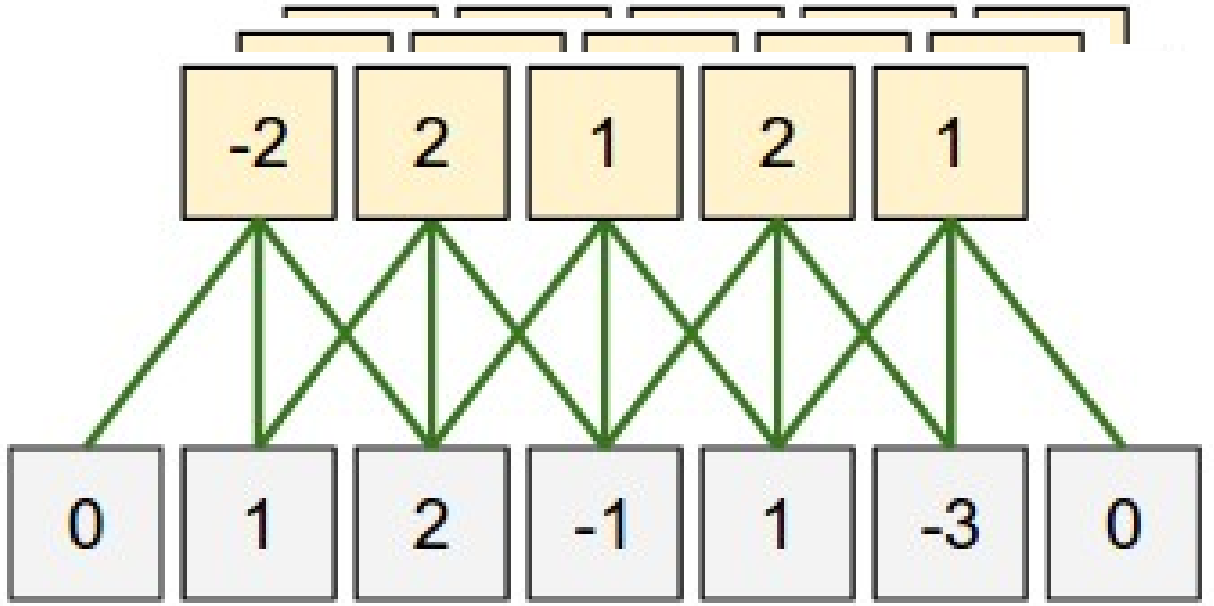
w=

1	0	-1

Inputs (image pixels, or outputs of each previous layer neuron)

Convolution as a hierarchy, trees

The same is repeated many times, producing multiple ”output channels”. Each channel has its own filter **w**



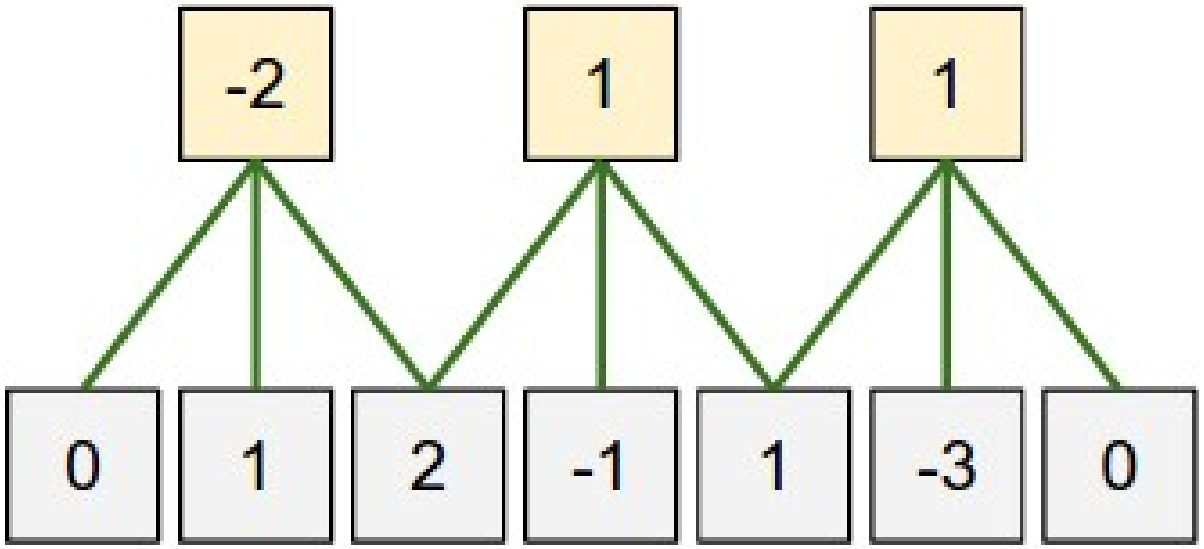
w=

1	0	-1

Inputs (image pixels, or outputs of each previous layer neuron)

Convolution as a hierarchy, trees

Outputs of this layer's neurons

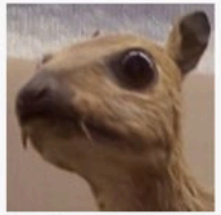
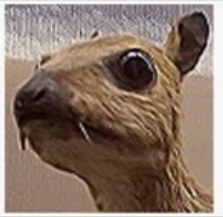
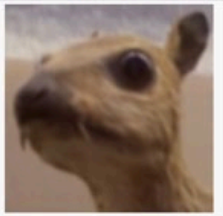


w=

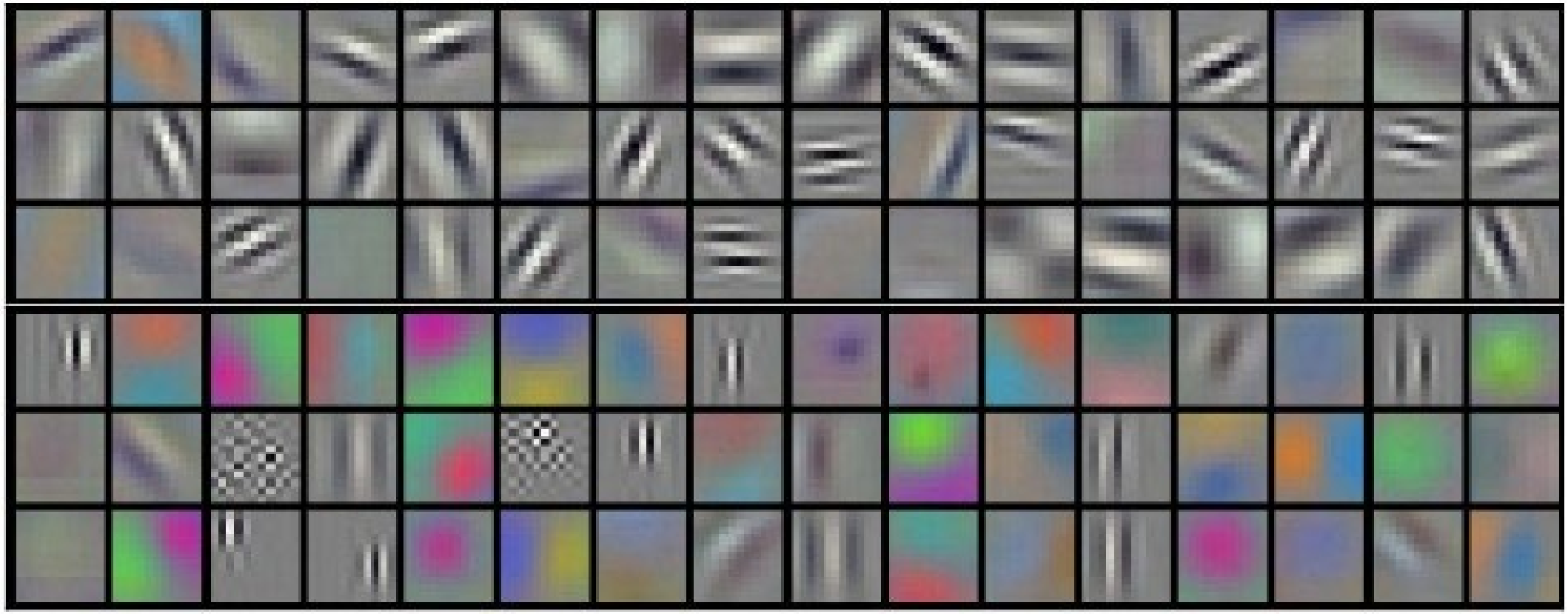
1	0	-1

Inputs (image pixels, or outputs of each previous layer neuron)

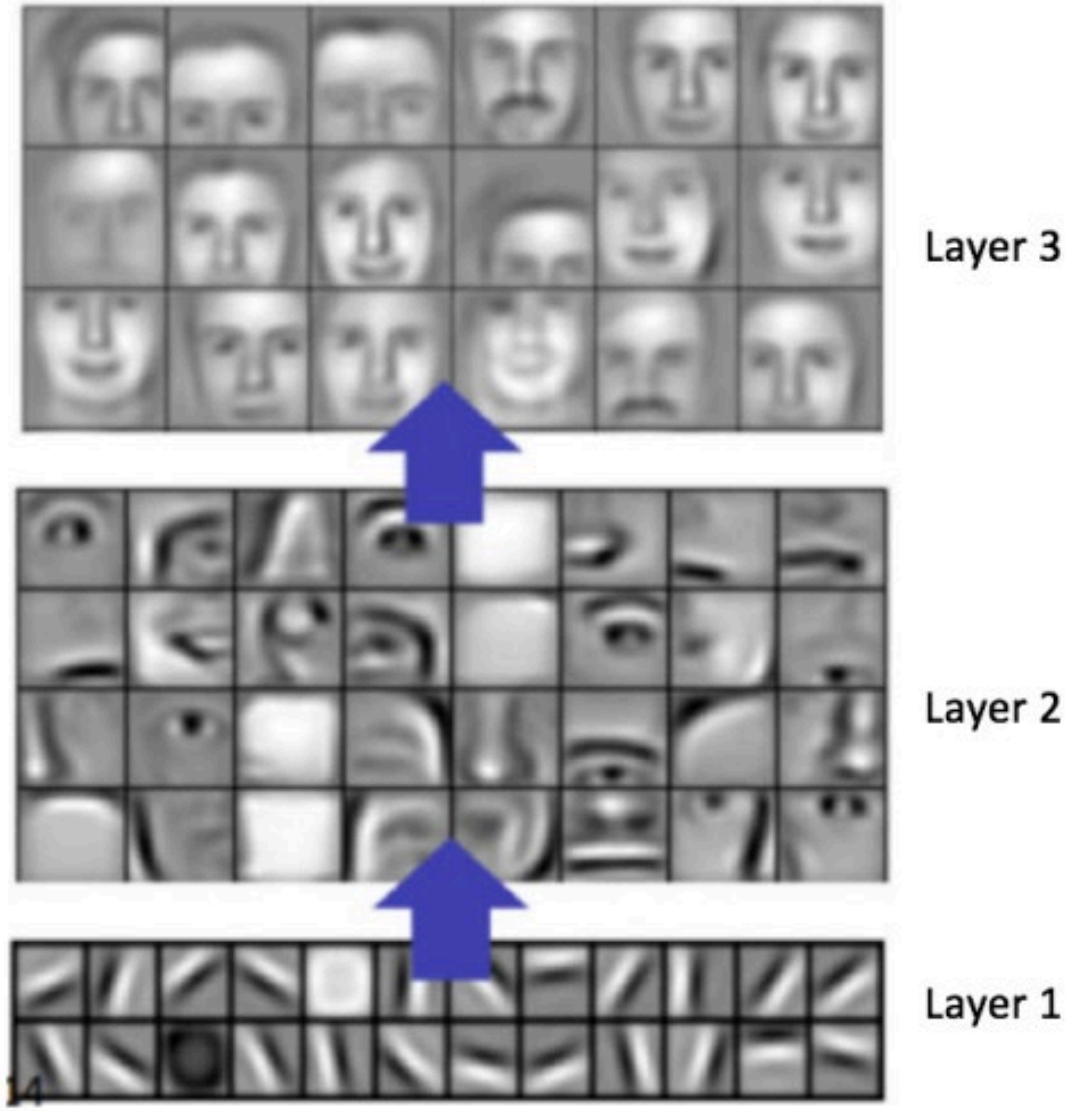
Examples of common filters in image processing

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Examples of learned filters in image classification



Examples of learned filters in image classification



Interactive exploration:
https://distill.pub/2020/circuits/early-vision/

Boundary 7%

Show all 36 neurons.

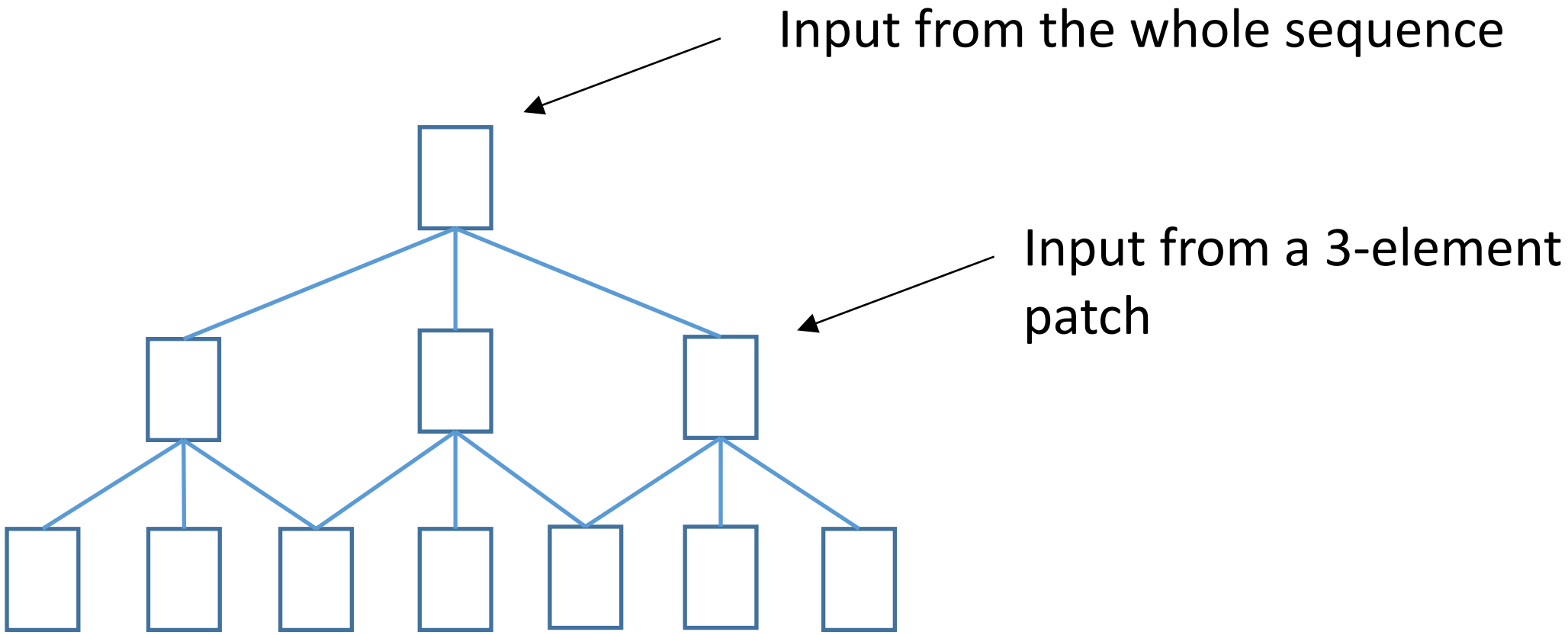
These units use multiple cues to detect the boundaries of objects. They vary in orientation, detecting convex/concave/straight boundaries, and detecting artificial vs fur foregrounds. Cues they rely on include line detectors, high-low frequency detectors, and color contrast.

Proto-Head 2%

Show all 12 neurons.

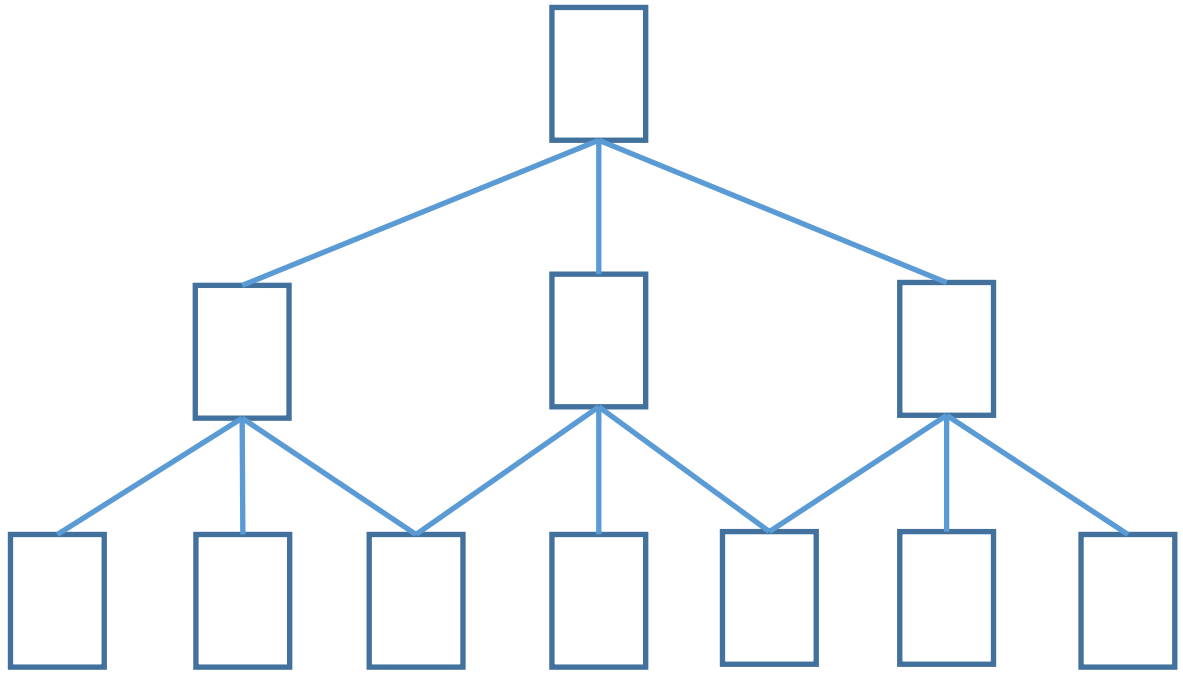
The tiny eye detectors, along with texture detectors for fur, hair and skin developed at the previous layer enable these early head detectors, which will continue to be refined in the next layer.

Later layers have larger receptive fields



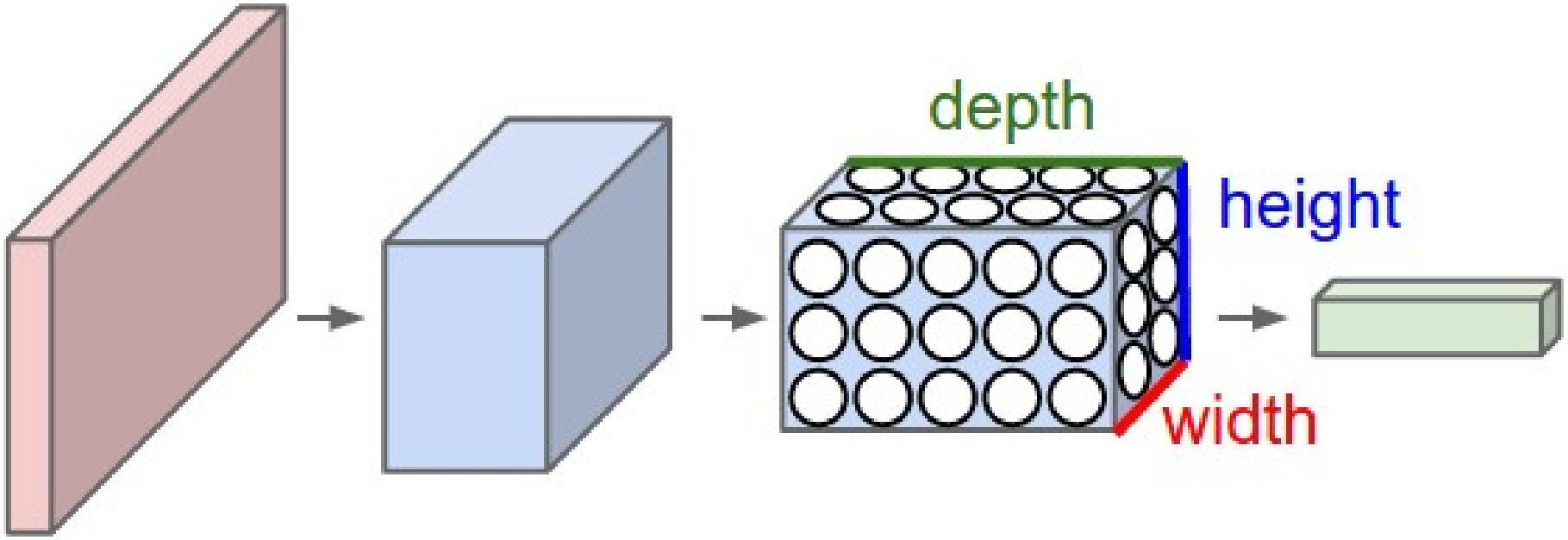
Why does it work?

- Weight sharing (the neurons receiving input from different input regions all have the same parameters)
- This greatly reduces the number of optimized parameters in a layer
- At low layers, pixels far from each other are assumed to be more independent

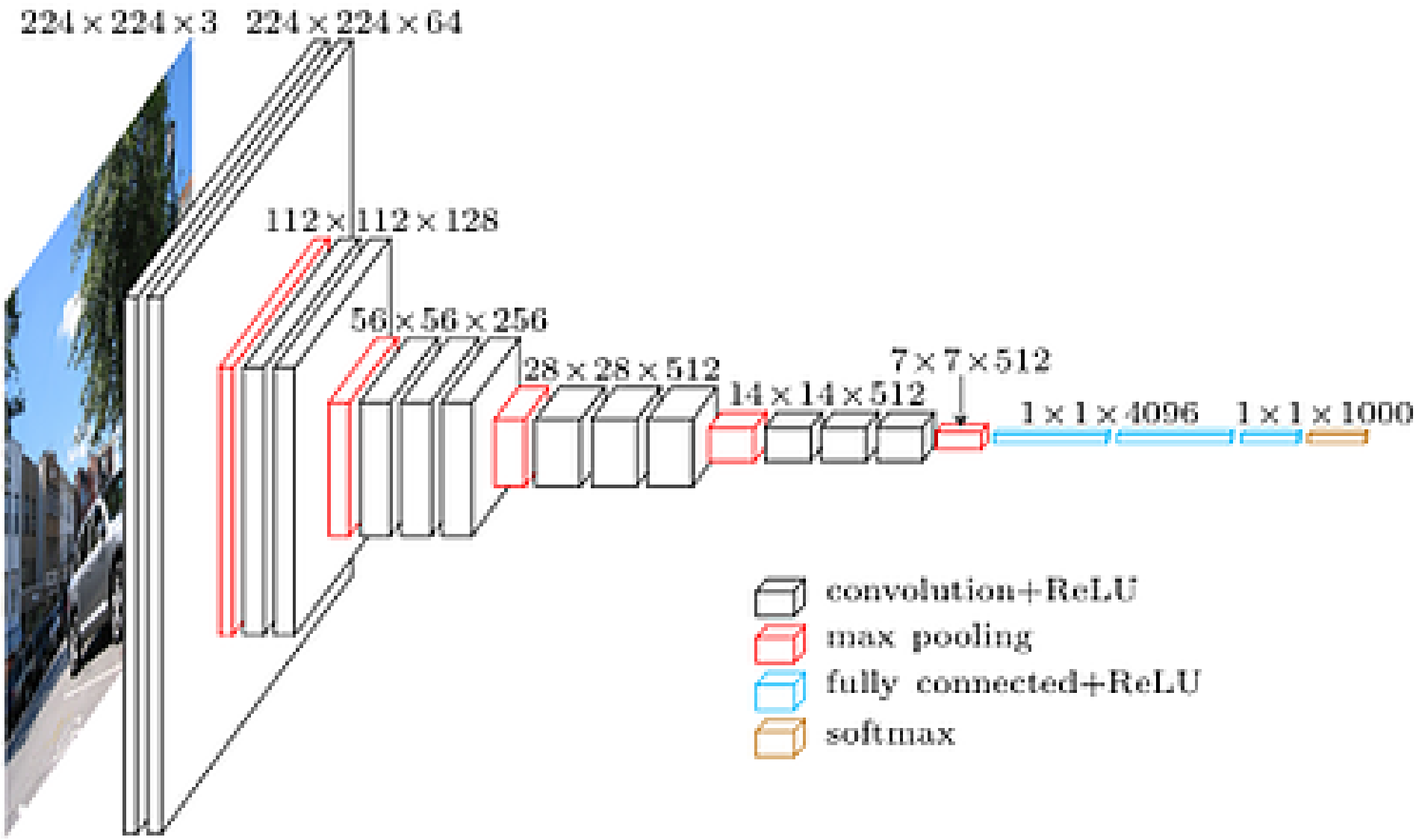


Convolutional networks with images

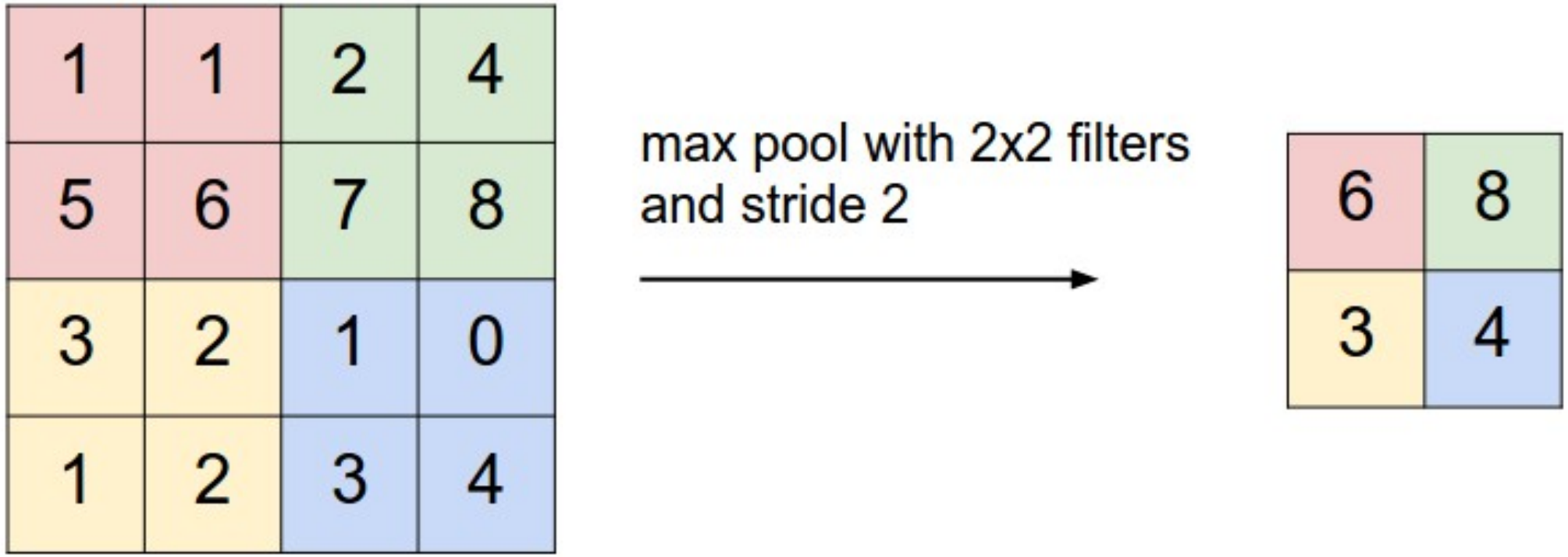
- Inputs and outputs are 4D tensors with shape (number of images, height,width,depth). Depth=number of image channels
- A single neuron takes as input a multichannel image patch, outputs one pixel in its own channel



The VGG16 model



Pooling


1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET

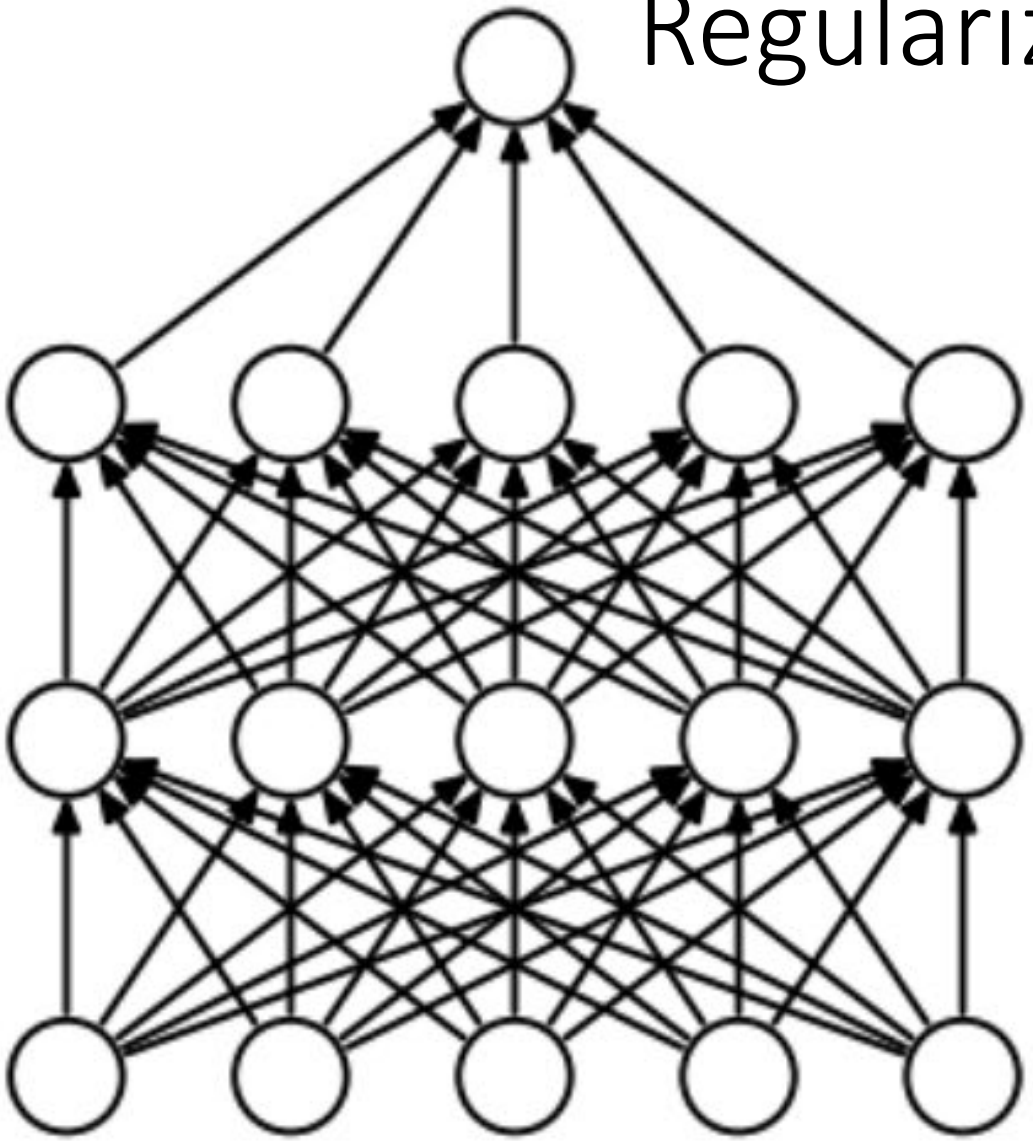
Jost Tobias Springenberg*, **Alexey Dosovitskiy***, **Thomas Brox, Martin Riedmiller**
Department of Computer Science
University of Freiburg
Freiburg, 79110, Germany
{springj, dosovits, brox, riedmiller}@cs.uni-freiburg.de

ABSTRACT

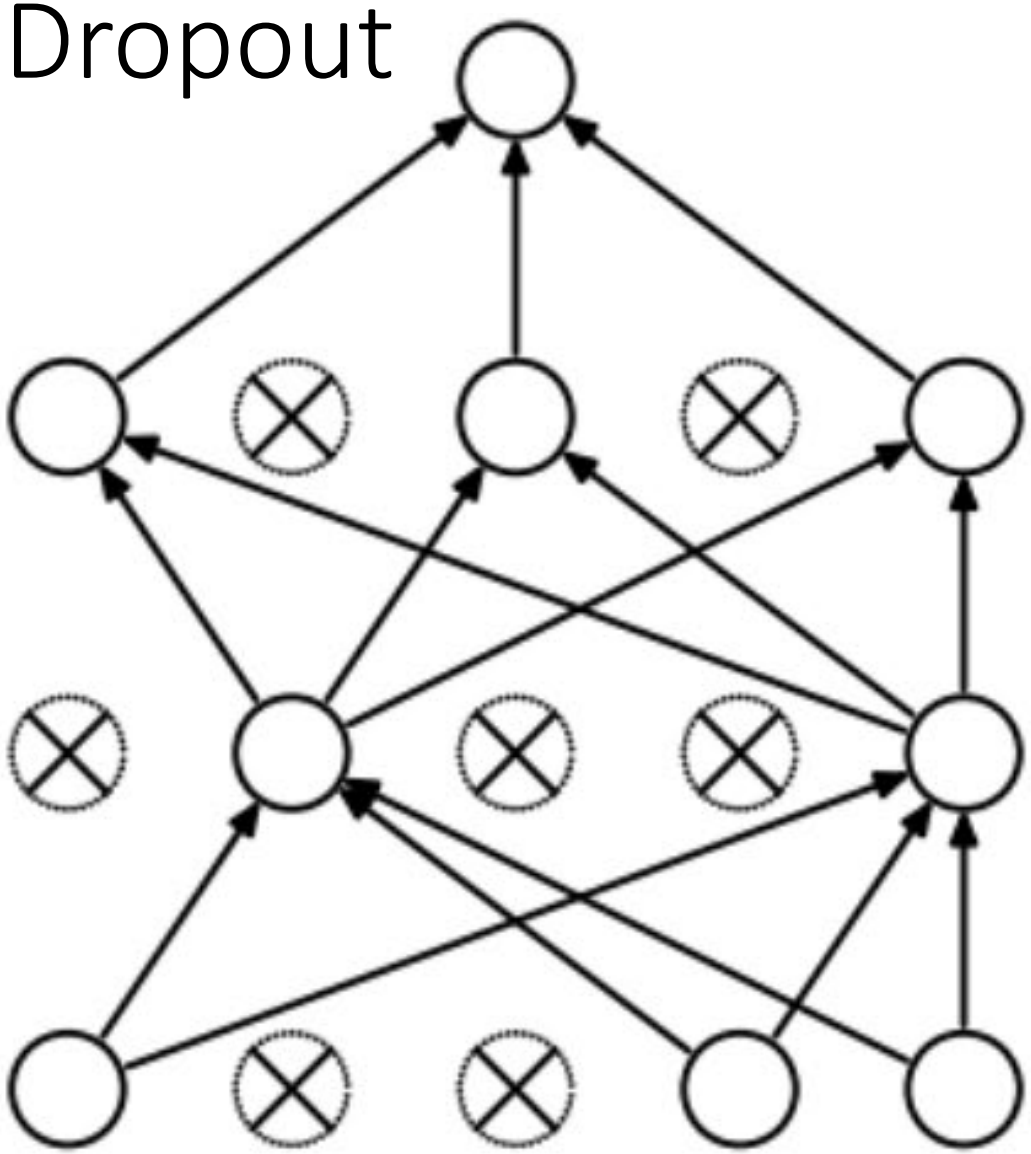
Most modern convolutional neural networks (CNNs) used for object recognition are built using the same principles: Alternating convolution and max-pooling layers followed by a small number of fully connected layers. We re-evaluate the state of the art for object recognition from small images with convolutional networks, questioning the necessity of different components in the pipeline. We find that max-pooling can simply be replaced by a convolutional layer with increased stride without loss in accuracy on several image recognition benchmarks. Following this finding – and building on other recent work for finding simple network structures – we propose a new architecture that consists solely of convolutional layers and yields competitive or state of the art performance on several object recognition datasets (CIFAR-10, CIFAR-100, ImageNet). To analyze the network we introduce a new variant of the "deconvolution approach" for visualizing features learned by CNNs, which can be applied to a broader range of network structures than existing approaches.

```
model = keras.models.Sequential()
model.add(Conv2D(16, kernel_size=(5, 5), strides=[2,2],
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(32, (5, 5), activation='relu', strides=[2,2]))
model.add(Conv2D(32, (3, 3), activation='relu', strides=[2,2]))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Regularization: Dropout



(a) Standard Neural Net



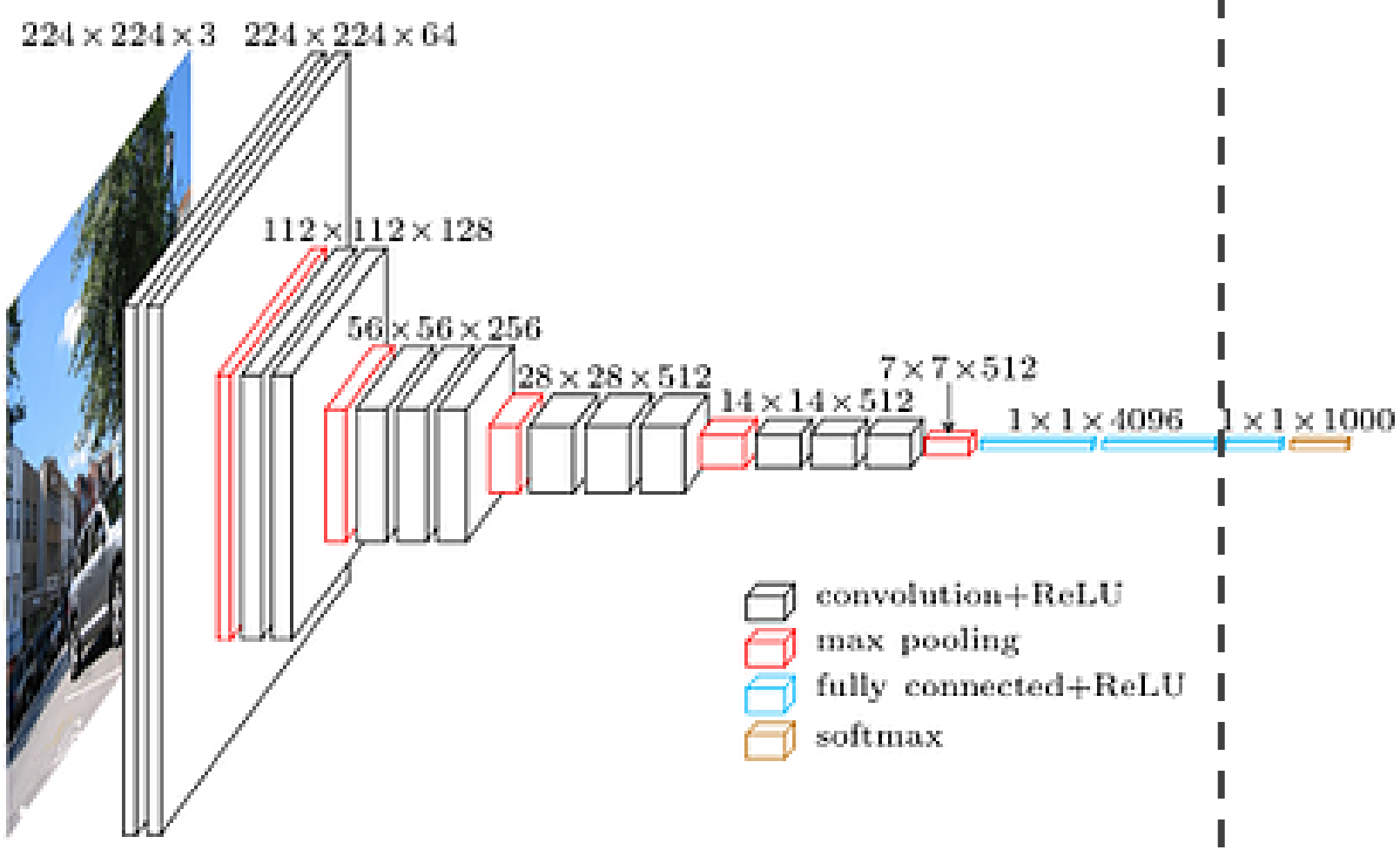
(b) After applying dropout.

Contents

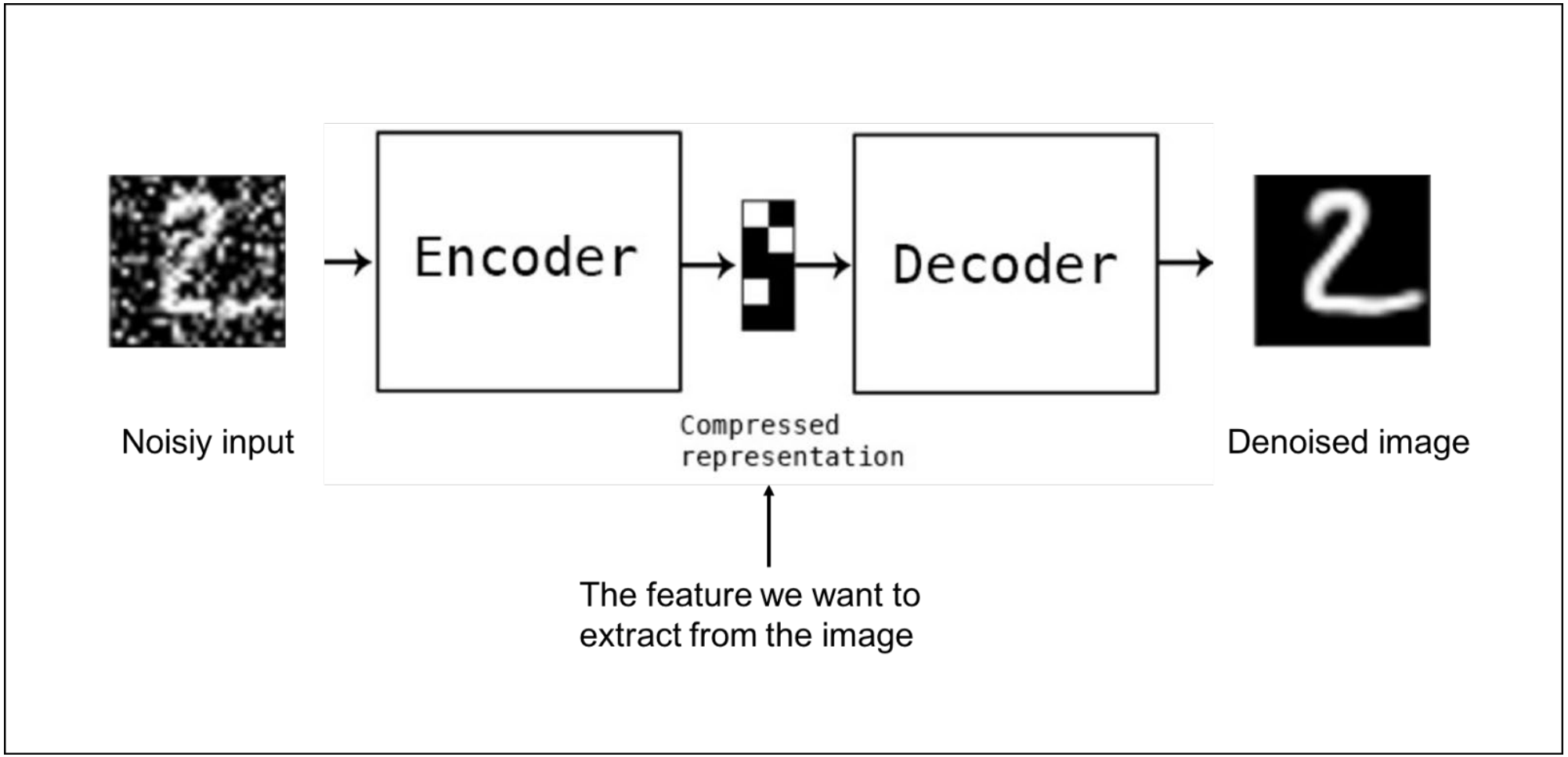
- Preliminaries: compute graphs, artificial neurons, activation functions, loss functions
- Understanding nonlinear activations
- Understanding skip-connections
- Convolutional neural networks
- **Encoder-decoder architectures**
- Applications, software packages
- Transfer learning

Encoder, a compact representation of image contents

Simple classifier: One neuron per Image class

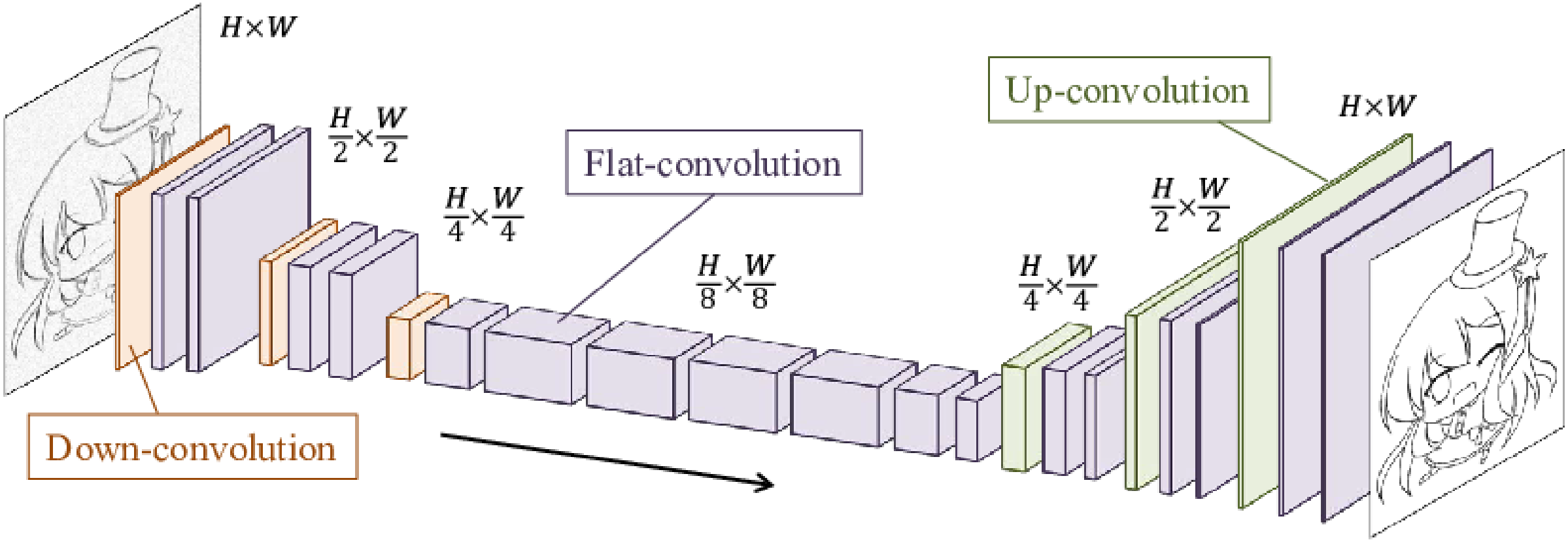


Encoder-decoder architectures



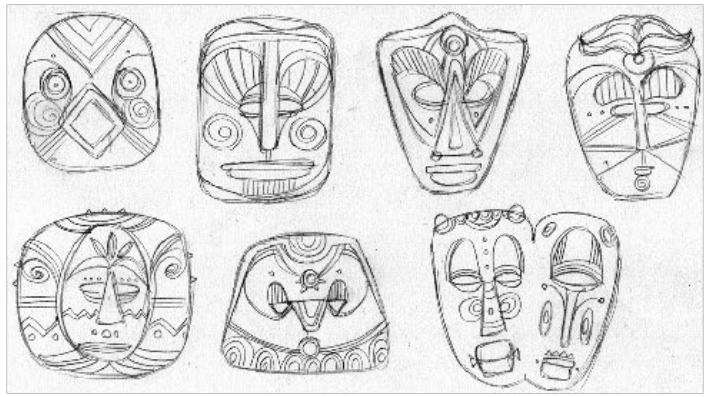
Autoencoder = encoder-decoder network trained with output=input

Convolutional encoder-decoder



(a) Animals

(b) Kimono

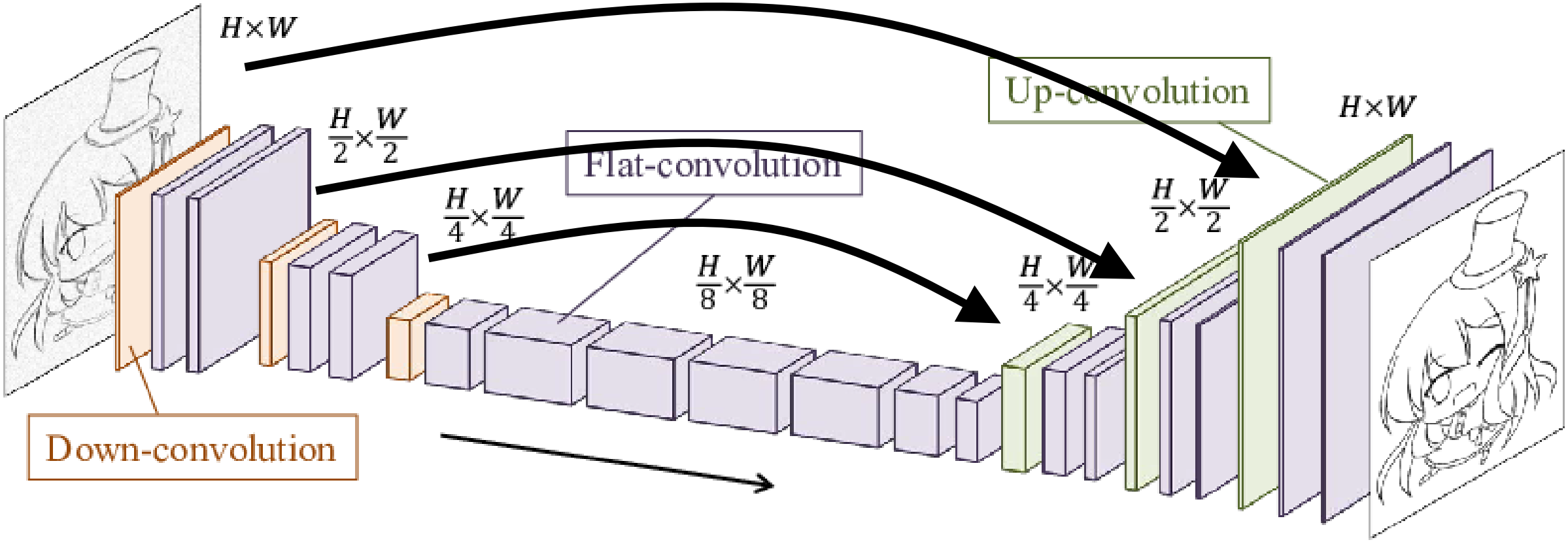


(c) Masks

(d) Book

(e) Standing girl

Skip-connections and convolutions: high-resolution local decisions utilizing local input info



Skip-connections and convolutions: U-Net (2015)

U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies,
University of Freiburg, Germany
ronneber@informatik.uni-freiburg.de,
WWW home page: http://lmb.informatik.uni-freiburg.de/

Abstract. There is large consent that successful training of deep networks requires many thousand annotated training samples. In this paper, we present a network and training strategy that relies on the strong use of data augmentation to use the available annotated samples more efficiently. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. We show that such a network can be trained end-to-end from very few images and outperforms the prior best method (a sliding-window convolutional network) on the ISBI challenge for segmentation of neuronal structures in electron microscopic stacks. Using the same network trained on transmitted light microscopy images (phase contrast and DIC) we won the ISBI cell tracking challenge 2015 in these categories by a large margin. Moreover, the network is fast. Segmentation of a 512x512 image takes less than a second on a recent GPU. The full implementation (based on Caffe) and the trained networks are available at http://lmb.informatik.uni-freiburg.de/people/ronneber/u-net.

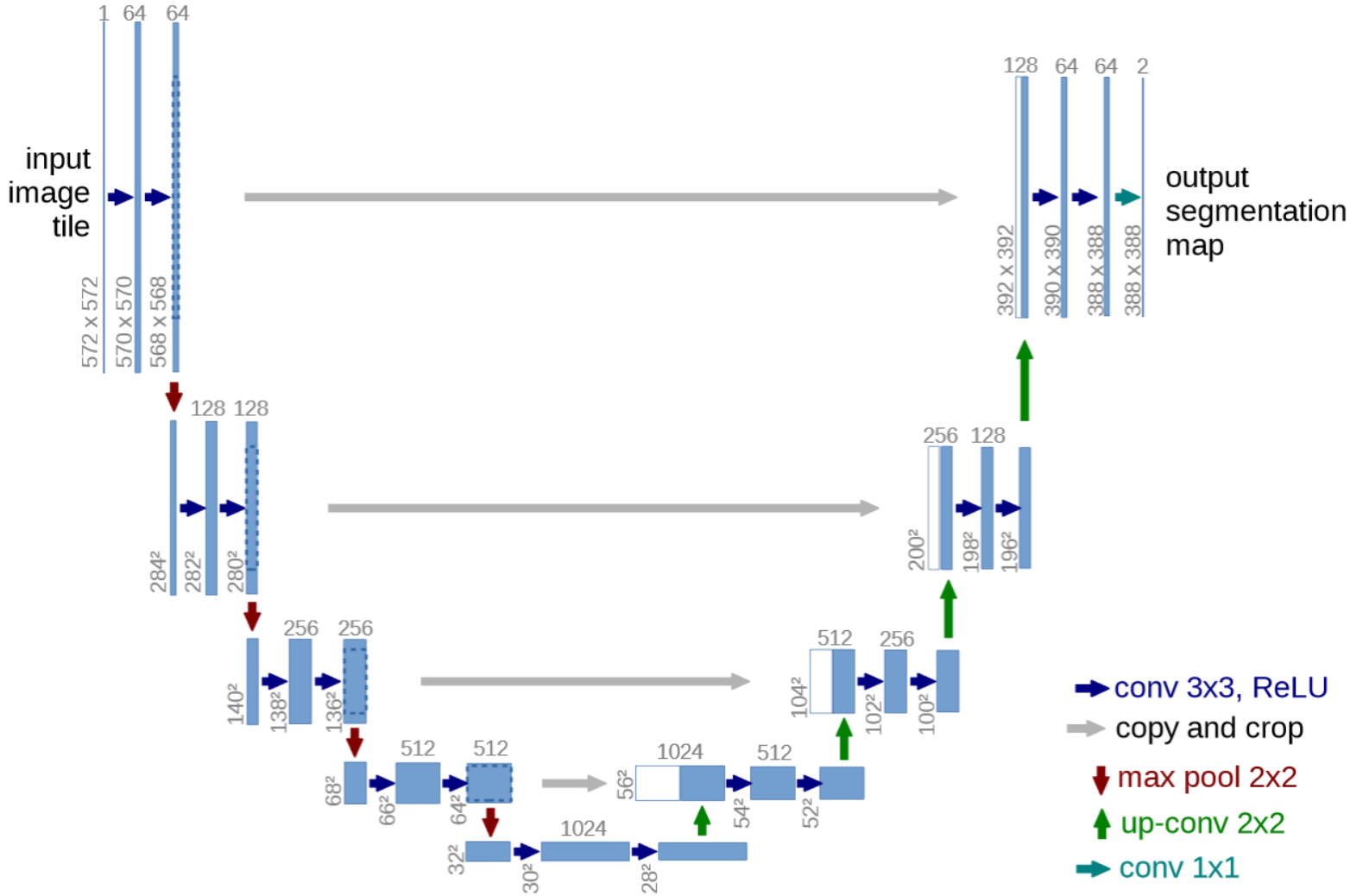


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Pixels to pixels is easy

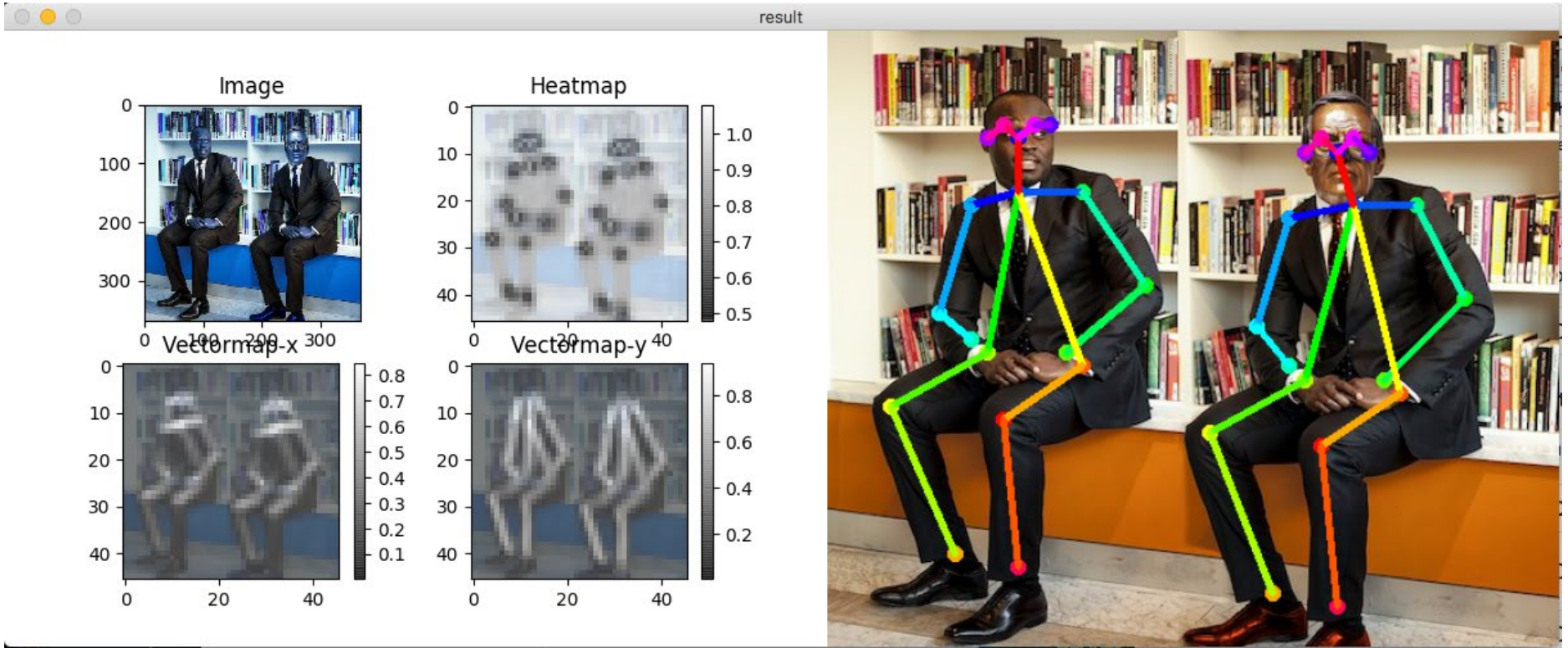
- For a convolutional neural network, it may be easier to map input pixels to output pixels instead of arbitrary values.
- Especially true for U-Nets
- For example, easier to color a recognized/tracked object with some predetermined color than to output the object's coordinates
- Better to encode everything as pixels than mix data types
- Local decisions based local information are easier to learn

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

OpenPose

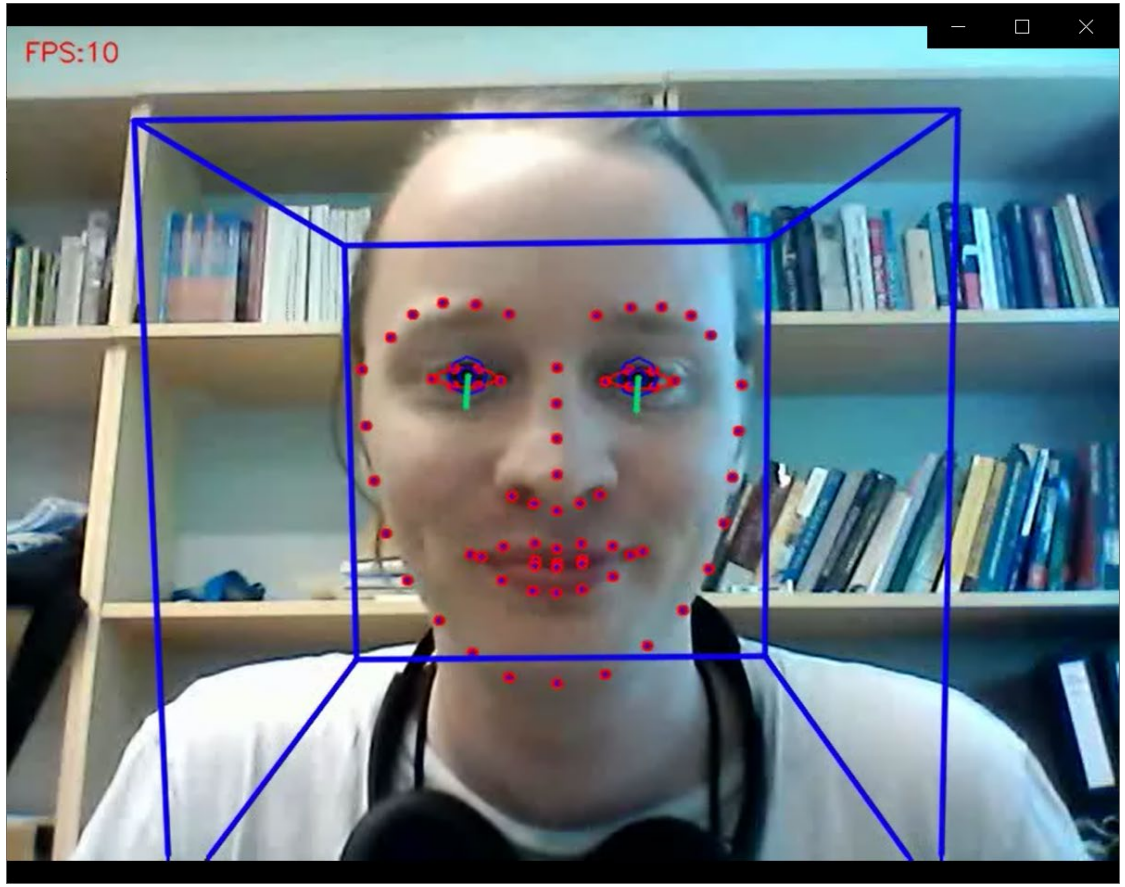
- Real-time tracking of multiple people, works even with a moving camera
- Pixel-to-pixel labeling of body parts, then parsing the skeletons
- https://github.com/ildoonet/tf-pose-estimation
- https://github.com/CMU-Perceptual-Computing-Lab/openpose_unity_plugin
- Test this in your browser (also works on mobile): https://urly.fi/1dfJ



OpenFace: a facial behavior analysis toolkit

https://github.com/TadasBaltrusaitis/OpenFace/wiki

- An easy to use opensource toolkit that detects facial landmarks, head pose, eye-gaze direction and facial action units
- E.g., for analyzing user test videos



Action unit estimation

- Openface can estimate facial movements based on Facial Action Coding System (FACS)
- In FACS, facial movements are coded as different action units (AU): https://imotions.com/blog/facial-action-coding-system/
- For example: Smile = contraction of zygomatic major (pulls the corner of lips) and inferior part of orbicularis oculi (raises cheeks, causes wrinkles in the corner of the eye)
- Openface estimates the presence and the intensity (scale from 0 to 5) of different AUs in each frame of the video.
- For the accuracy of AU estimates, see Baltrusaitis, T., Zadeh, A., Lim, Y. C., & Morency, L. (2018). OpenFace 2.0: Facial Behavior Analysis Toolkit. *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, 59–66. https://doi.org/10.1109/FG.2018.00019

How to use, step by step (Windows Powershell)

- Openface is operated trough command line interface: e.g. with PowerShell (Windows) or xterm (Unix)
- 1. Install: https://github.com/TadasBaltrusaitis/OpenFace/wiki/Windows-Installation
- 2. Open Windows Powershell
- 3. Change the directory to the openface folder, for example:
  - cd C:
  - cd \...\...\...\OpenFace_2.0.5_win_x64\

Useful command line arguments

- 4: Execute a command, examples:
  - Extract features (CSV-file): .\FeatureExtraction.exe -f "C:\...\...\filename.avi”
  - Extract AU estimates (CSV-file): \FeatureExtraction.exe -aus –f "C:\...\...\...\filename.avi”
  - Visualize the data: .\FeatureExtraction.exe –verbose "C:\...\...\filename.avi”
- 5. Analyze the data ☺
- Also possible to use GUI with the argument ./OpenFaceOffline.exe
- List of all possible arguments: https://github.com/TadasBaltrusaitis/OpenFace/wiki/Command-line-arguments

Administrator: Windows PowerShell

```
PS C:\Openface\OpenFace_2.0.5_win_x64> .\FeatureExtraction.exe -f "C:\my videos\video.avi" -verbose
Reading the landmark detector/tracker from: model/main_ceclm_general.txt
Reading the landmark detector module from: model\cen_general.txt
Reading the PDM module from: model\pdms/In-the-wild_aligned_PDM_68.txt....Done
Reading the Triangulations module from: model\tris_68.txt....Done
Reading the intensity CEN patch experts from: model\patch_experts/cen_patches_0.25_of.dat....Done
Reading the intensity CEN patch experts from: model\patch_experts/cen_patches_0.35_of.dat....Done
Reading the intensity CEN patch experts from: model\patch_experts/cen_patches_0.50_of.dat....Done
Reading the intensity CEN patch experts from: model\patch_experts/cen_patches_1.00_of.dat....Done
Reading part based module....left_eye_28
Reading the landmark detector/tracker from: model\model_eye/main_clnf_synth_left.txt
Reading the landmark detector module from: model\model_eye\clnf_left_synth.txt
Reading the PDM module from: model\model_eye\pdms/pdm_28_l_eye_3D_closed.txt....Done
Reading the intensity CCNF patch experts from: model\model_eye\patch_experts/left_ccnf_patches_1.00_synth_lid_.txt....Done
Reading the intensity CCNF patch experts from: model\model_eye\patch_experts/left_ccnf_patches_1.50_synth_lid_.txt....Done
Done
Reading part based module....right_eye_28
Reading the landmark detector/tracker from: model\model_eye/main_clnf_synth_right.txt
Reading the landmark detector module from: model\model_eye\clnf_right_synth.txt
Reading the PDM module from: model\model_eye\pdms/pdm_28_eye_3D_closed.txt....Done
Reading the intensity CCNF patch experts from: model\model_eye\patch_experts/ccnf_patches_1.00_synth_lid_.txt....Done
Reading the intensity CCNF patch experts from: model\model_eye\patch_experts/ccnf_patches_1.50_synth_lid_.txt....Done
Done
Reading the landmark validation module....Done
Reading the AU analysis module from: AU_predictors/main_dynamic_svms.txt
Reading the AU predictors from: AU_predictors\AU_all_best.txt... Done
Reading the PDM from: AU_predictors\In-the-wild_aligned_PDM_68.txt... Done
Reading the triangulation from:AU_predictors\tris_68_full.txt... Done
Attempting to read from file: C:\my videos\video.avi
Device or file opened
Starting tracking
Reading the MTCNN face detector from: model/mtcnn_detector/MTCNN_detector.txt
Reading the PNet module from: model/mtcnn_detector\PNet.dat
Reading the RNet module from: model/mtcnn_detector\RNet.dat
Reading the ONet module from: model/mtcnn_detector\ONet.dat
0% 10% 20% 30% 40% 50% 60% 70% Closing output recorder
Closing input reader
Closed successfully
Postprocessing the Action Unit predictions
PS C:\Openface\OpenFace_2.0.5_win_x64> .\FeatureExtraction.exe -f "C:\my videos\video.avi" -verbose
```

Add-in Express

GPowerWin_3.1.9.2

18.42
11.6.2019

Detectron 2

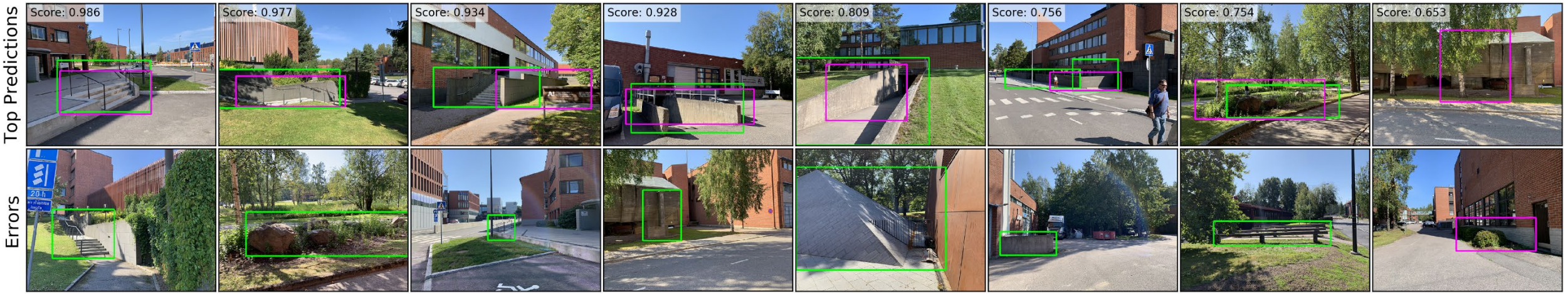
https://github.com/facebookresearch/detectron2

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**

Transfer learning with Detectron 2

- Transfer learning: learning task 1 that makes learning task 2 easier
- Typically: train a neural network with a large amount of generic data, then continue training with custom per-application data
- We recently used this to recognize parkour spots from street level photography.
- We used 10k images labeled in LabelBox, about 2 days of labeling work



Parkour is a form of exercise and physical play
suitable for a wide range of skills and fitness

Summary

- Preliminaries: compute graphs, artificial neurons, activation functions, loss functions
- Understanding nonlinear activations
- Understanding skip-connections
- Convolutional neural networks
- Encoder-decoder architectures

Key takeaways

- Preliminaries: **Neural networks are compute graphs.** Tensorflow, Pytorch etc. are tools for building and optimizing the graphs
- Loss functions: **Mean squared error for regression** (line fitting as the simple example)**, softmax cross-entropy for classification** (and other tasks where last layer outputs a discrete probability distribution)
- Understanding nonlinear activations: **Deeper networks can partition the input space in exponentially more detailed manner**, but width also helps and makes optimizing network parameters easier.
- Understanding skip-connections: **Key to training very deep networks, makes the optimization landscape more smooth**
- Convolutional neural networks: **Useful whenever input variables close to each other (in space or time) have stronger relationships**
- Encoder-decoder architectures: **The U-Net** is the standard tool in mapping pixels to pixels (or audio to audio) in a deterministic way, i.e., with clear ground truth output. If there are multiple possible outputs, one needs a generative model like GANs (next lecture)
- **Transfer learning** is a way to utilize large and powerful networks with your own small datasets.