Introduction to Deep Learning

Ludwig Krippahl

Introduction to Deep Learning

Summary

- The evolution of Neural Networks
- Deep Learning

- Why the success now
- Main advantages
- Problem of vanishing gradients

- Example: autoencoders

Evolution of Neural Networks

Evolution of Neural Networks

Neural Networks

- A very promising early start with neuron and perceptron:

- 1943: McCulloch & Pitts, model of neuron
- 1958: Rosenblatt, perceptron and learning algorithm

- But these turned out to be equivalent to generalized linear models

- And in 1969 Perceptrons (Minsky, Papert): need fully connected networks

1960-1980s: "AI Winter", in particular ANN

- 1970s: The equivalent of backpropagation appeared in other contexts
- 1986: Rumelhart, Hinton, Williams, backpropagation can be used for multi-layer networks

Evolution of Neural Networks

Machine Learning and Deep Learning

- In the 1990s, AI shifted from knowledge-driven to data-driven with new ML algorithms

- E.g. 1992 Vapnik et. al. publish the kernel trick for SVM

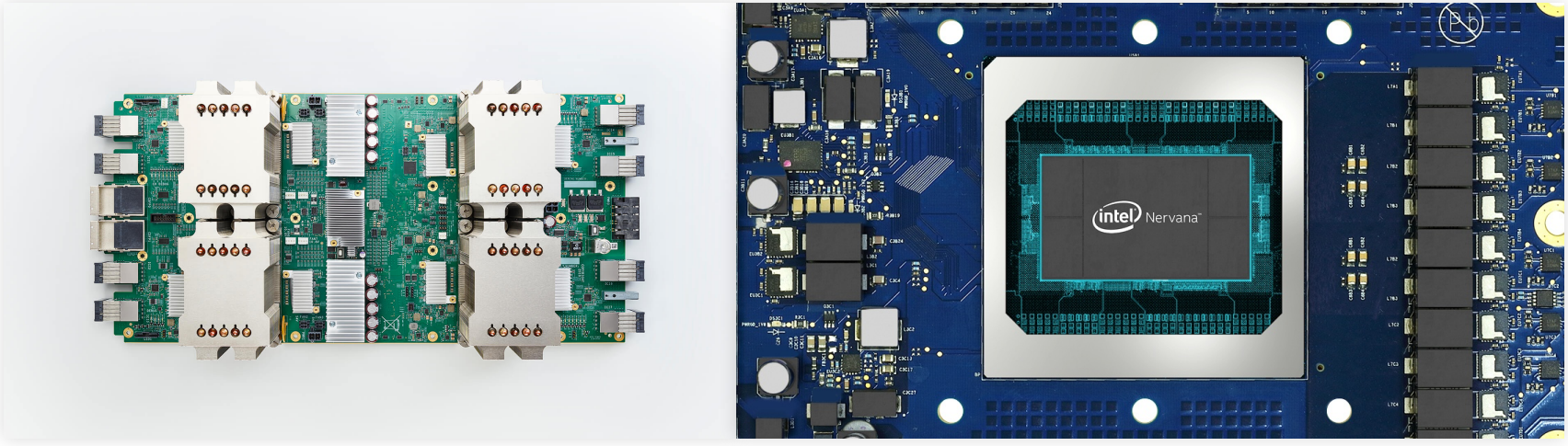
- 1997: Multi-layered and convolution networks for check processing USA (leCun)
- 1998: MNIST database (LeCun). Benchmarks, libraries and competitions

Evolution of Neural Networks

Deep Learning

- 2007: Hinton, pre-training of deep feedforward ANN
- 2011: Bengio, rectified linear unit (ReLU)
- New hardware: GPGPU

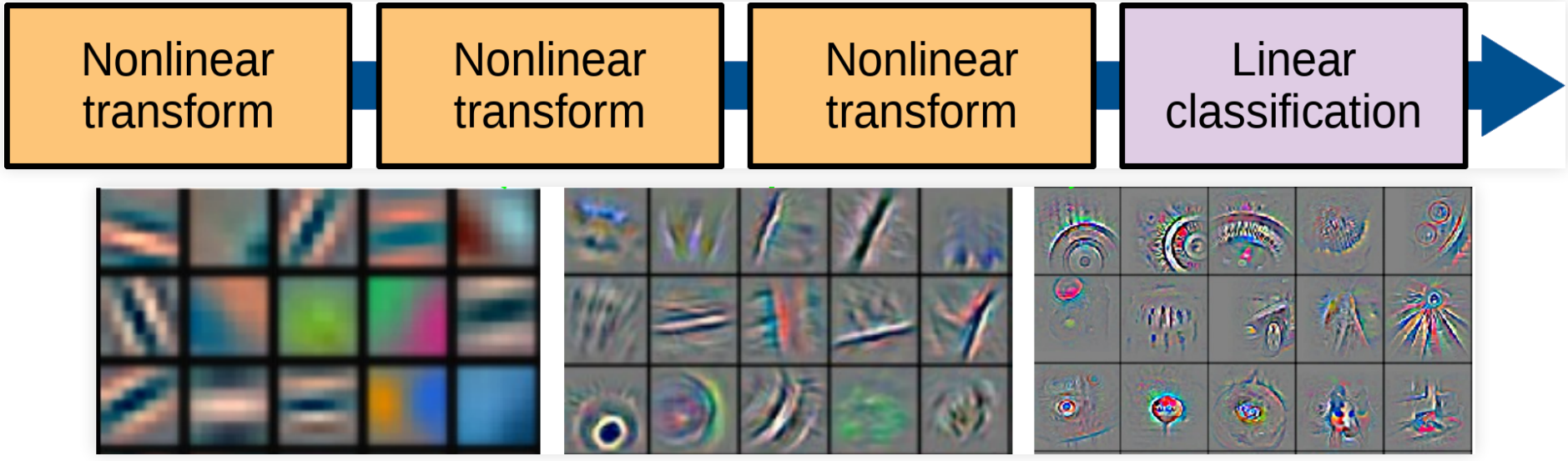
- Google Tensor Processing Unit
- Intel Nervana Neural Network Processor



Evolution of Neural Networks

Deep Learning

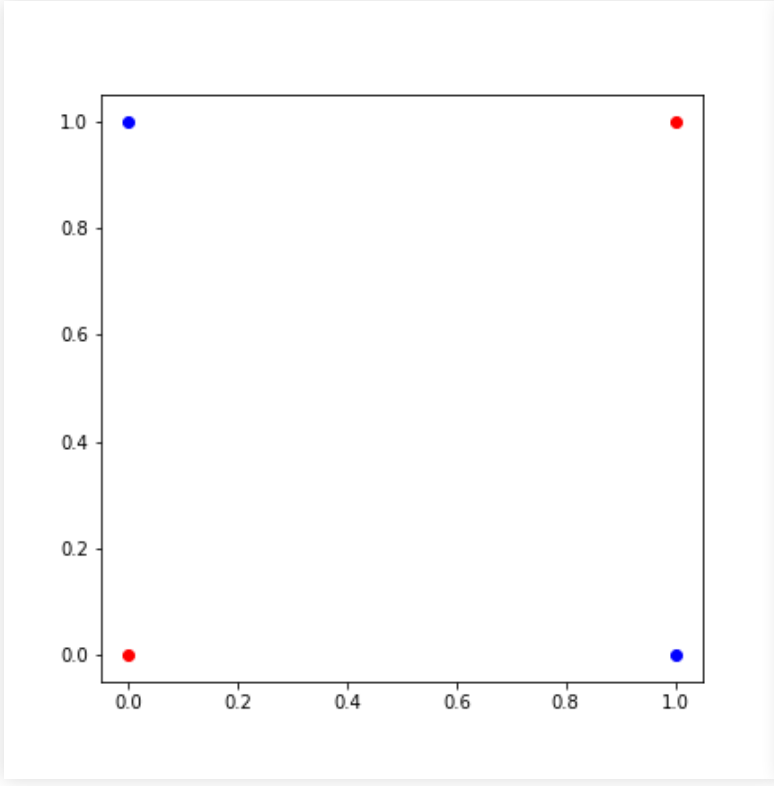
- Deep learning involves layers of nonlinear transformations
- These can learn features at different levels

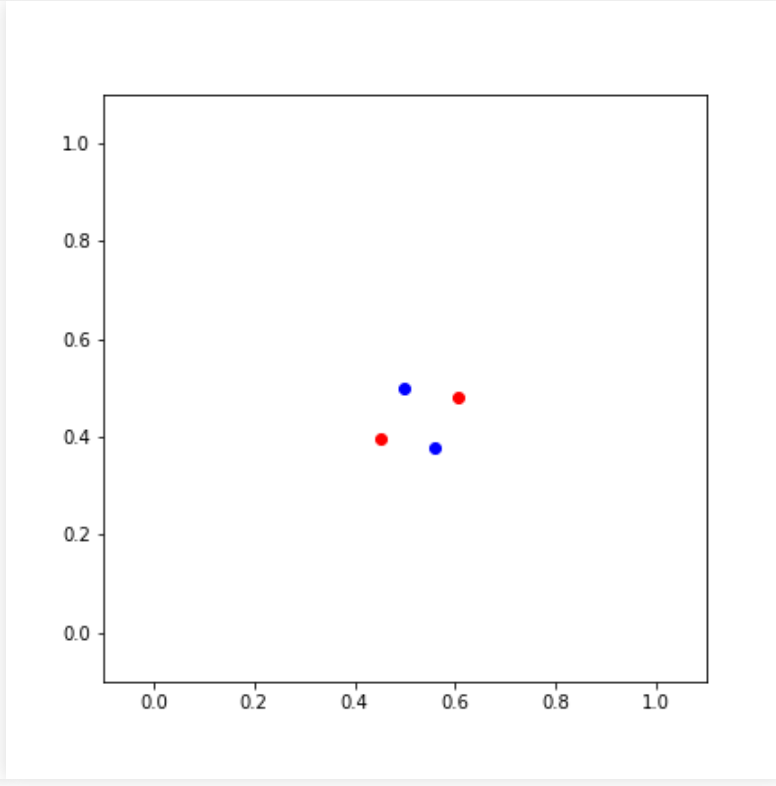


Zeitler, 2014, Visualizing and Understanding Convolutional Networks

Deep Learning

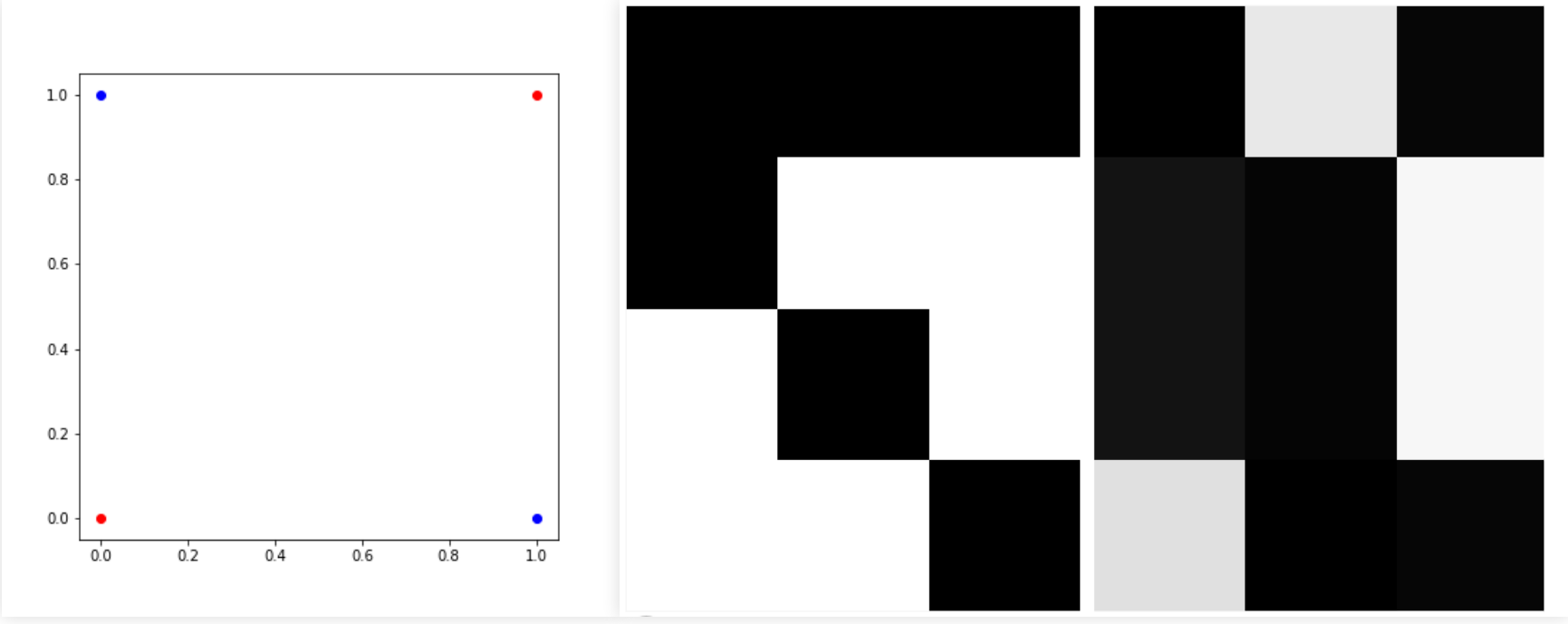
- Deep learning involves layers of nonlinear transformations
- XOR problem





Deep Learning

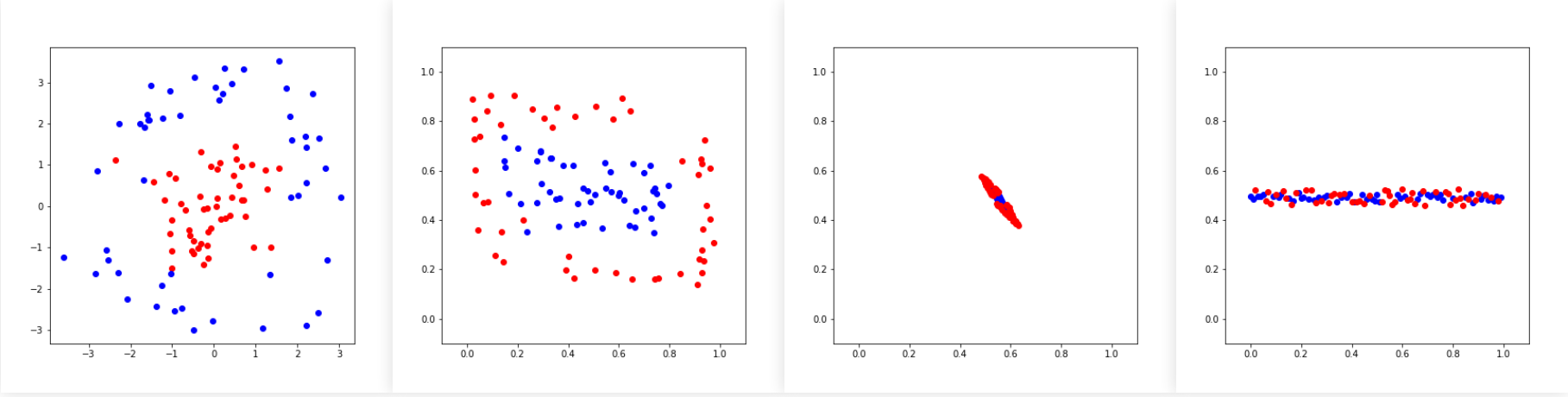
- Deep learning involves layers of nonlinear transformations
- XOR problem



Evolution of Neural Networks

Deep Learning

- Deep learning involves layers of nonlinear transformations
- More layers (2 input, 2 hidden layers of 2 neurons each, 1 output)



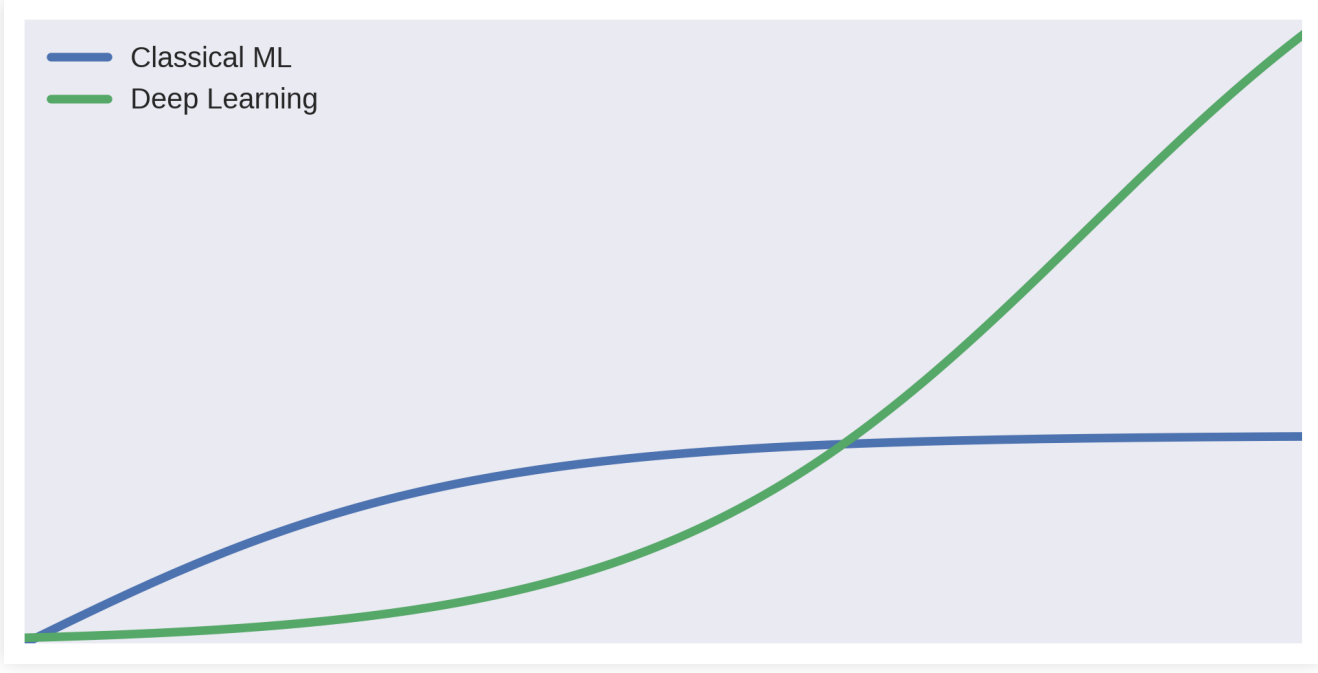
The Triumph of Deep Learning

Why now?

Example: computer vision

- 2006: Caltech 101 dataset, ~50 images per category (40-800)

- Classical CV models, 26% error
- Also, slow computers

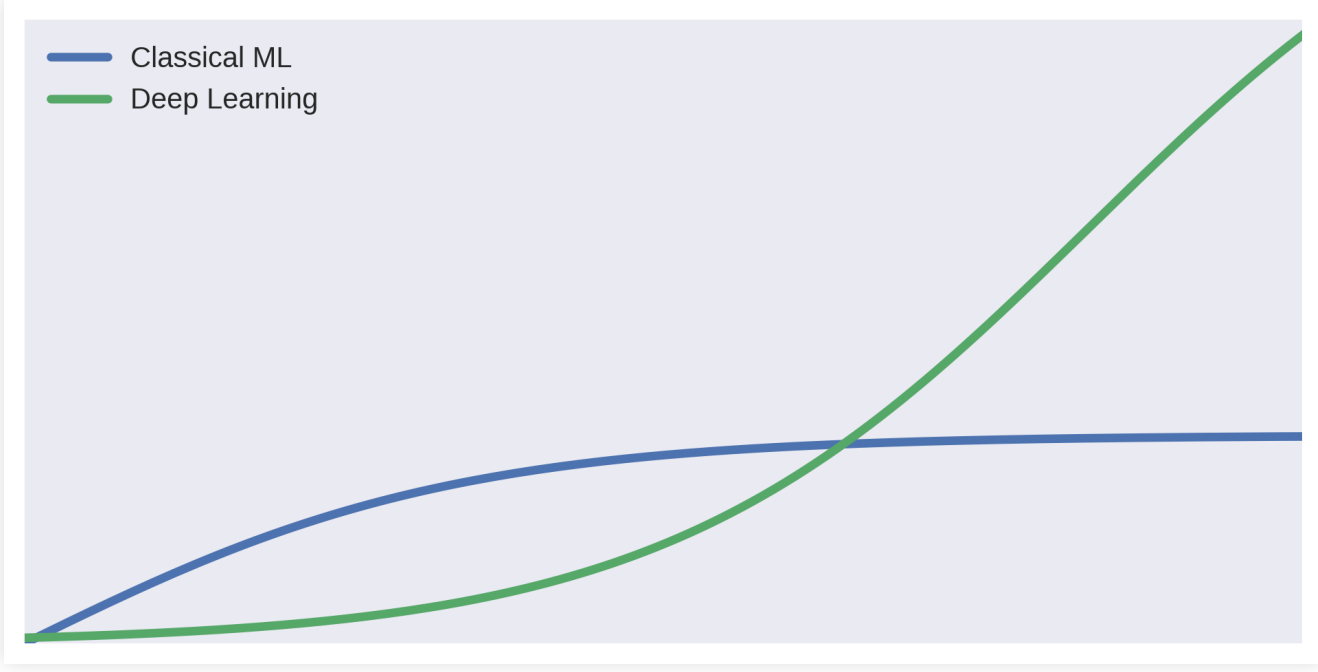


Why now?

Example: computer vision

- 2012: ImageNet dataset (1.2 M images, 1000 categories), GPGPU

- Large convolution networks became dominant



Why now?

Example: computer vision

- 2012: ImageNet dataset (1.2 M images, 1000 categories)
- More computing power (GPGPU)

Deep CNN became dominant:

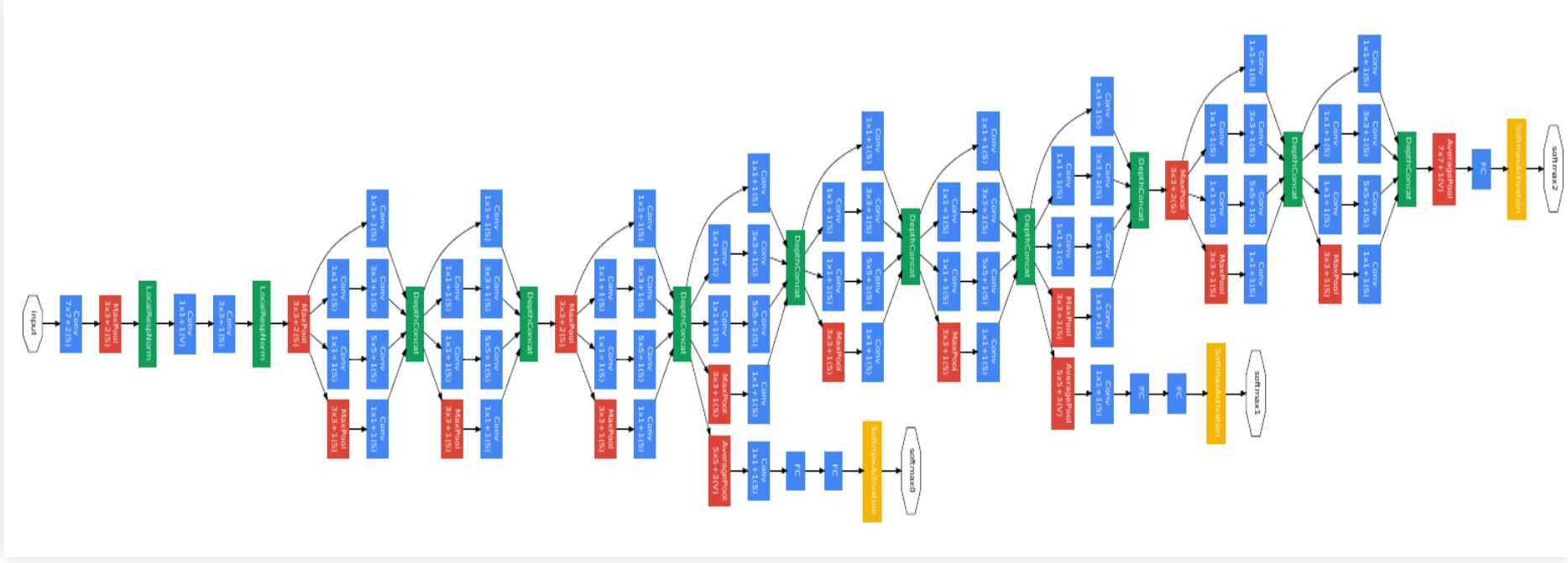
- AlexNet [Krizhevski, Sutskever, Hinton 2012], 15% top-5 error
- OverFeat [Sermanet et al. 2013], 13.8% error
- VGG Net [Simonyan, Zisserman 2014], 7.3% error
- GoogLeNet [Szegedy et al. 2014] 6.6% error
- ResNet [He et al. 2015] 5.7% error

(Yann Le Cun, CERN, 2016)

Why now?

Example: GoogleNet

- Szegedy et. al. 2015, 6.67% top-5 error in ImageNet



22 layers with parameters (+5 pooling) in 100 blocks

Feature Extraction

Feature Extraction

Classical approach: Feature engineering

- Example: Text mining

- Bag of words, clustering, concept extraction, sentiment analysis, ...

- Example: Image segmentation

- Edge detection, thresholding, histograms, ...

- Labour-intensive, takes years to perfect feature extraction methods

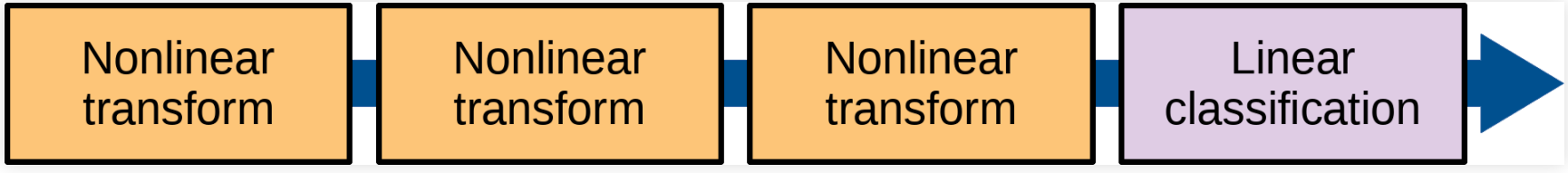
Feature Extraction

Classical approach

- Goal in AI: have the computer solve the problems, not us
- Custom-made feature extraction is not good for adapting quickly

- Need to respond to changing conditions
- Adversarial systems (credit fraud, fake social media accounts)

Deep learning solves these problems:



Feature Extraction

Nonlinear expansion of attributes

- We did this explicitely (by hand) with logistic regression
- Support Vector Machines do this automatically

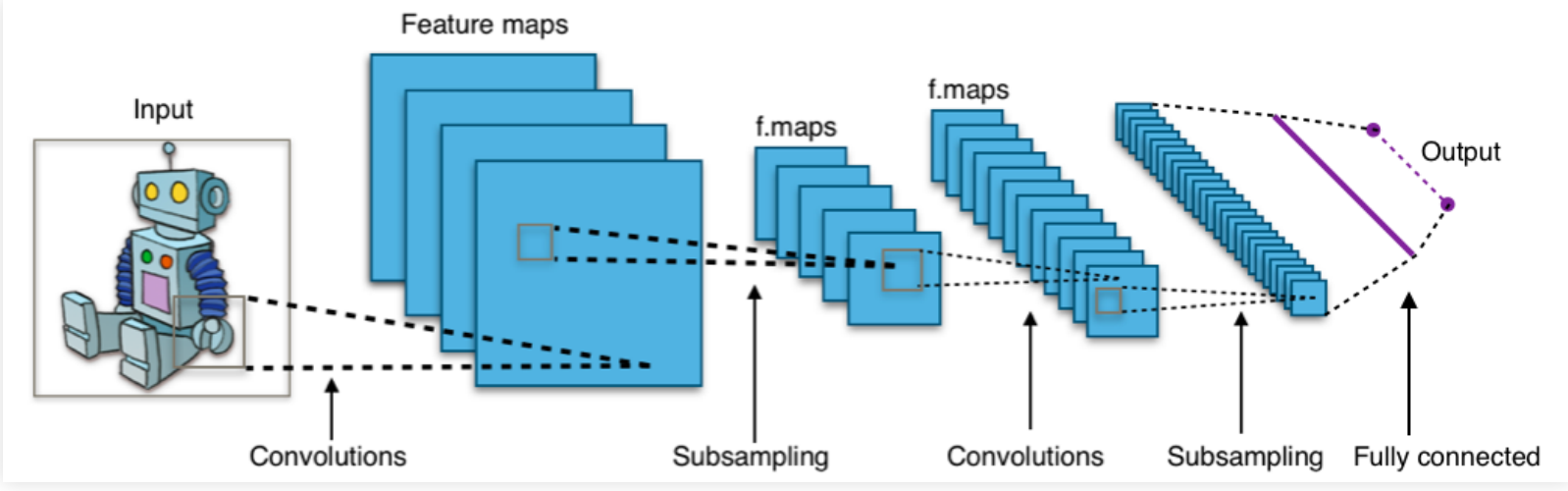
$$\arg\max_{\vec{\alpha}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(\vec{x}_n, \vec{x}_m)$$

- Where $K(\vec{x}_n, \vec{x}_m)$ is the kernel function for some non-linear expansion ϕ of our original data

Feature Extraction

Deep ANN stack layers with nonlinear responses

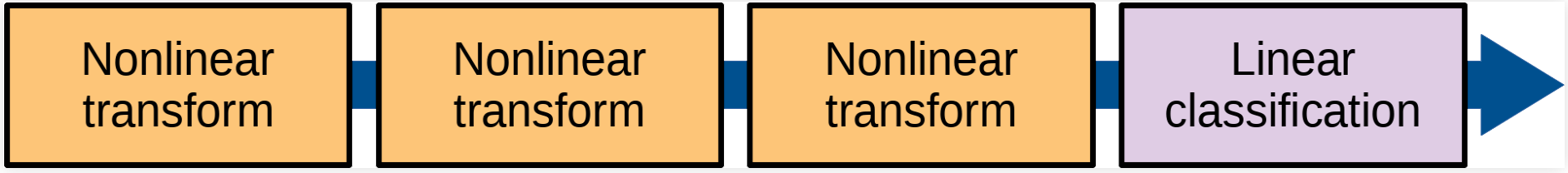
- Single neurons are linear classifiers, similar to logistic regression
- But deep networks can be extremely nonlinear, combining different representations



Feature Extraction

Deep ANN stack layers with nonlinear responses

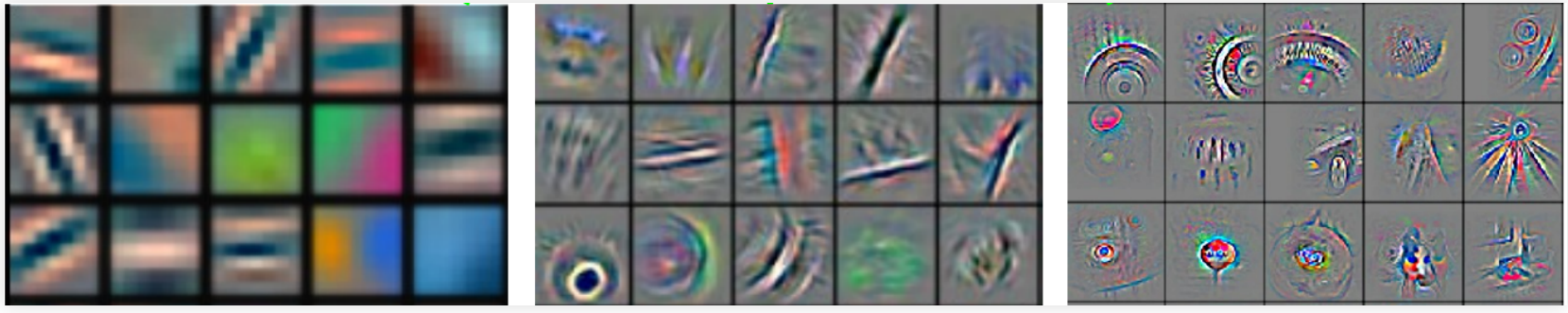
- Single neurons are linear classifiers, similar to logistic regression
- But large networks can be extremely nonlinear, combining different representations



Feature Extraction

Deep ANN stack layers with nonlinear responses

- These layers can find better representations than a single nonlinear transformation and the representations are learned



Zeitler, 2014, Visualizing and Understanding Convolutional Networks

- More efficient representations

- Instead of one transformation to arbitrarily large space

Feature Extraction

Universal approximation theorem

- One single layer can approximate any function, locally, with arbitrary precision
- In other words, all we need is one hidden layer (in theory)

Deep networks

- However, deep networks can oscilate more with fewer neurons
- Oscillations are related to Vapnik-Chervonenkis dimension

- (largest set that can be shattered)

Problem: Vanishing gradients

Vanishing gradients

Backpropagation in Deep Networks

- Output neuron n of layer k receives input from m from layer i through weight j

$$\Delta w^j_{mkn} = -\eta \frac{\delta E^j_{kn}}{\delta s^j_{kn}} \frac{\delta s^j_{kn}}{\delta net^j_{kn}} \frac{\delta net^j_{kn}}{\delta w_{mkn}} = \eta (t^j - s^j_{kn}) s^j_{kn} (1 - s^j_{kn}) s^j_{im} = \eta \delta_{kn} s^j_{im}$$

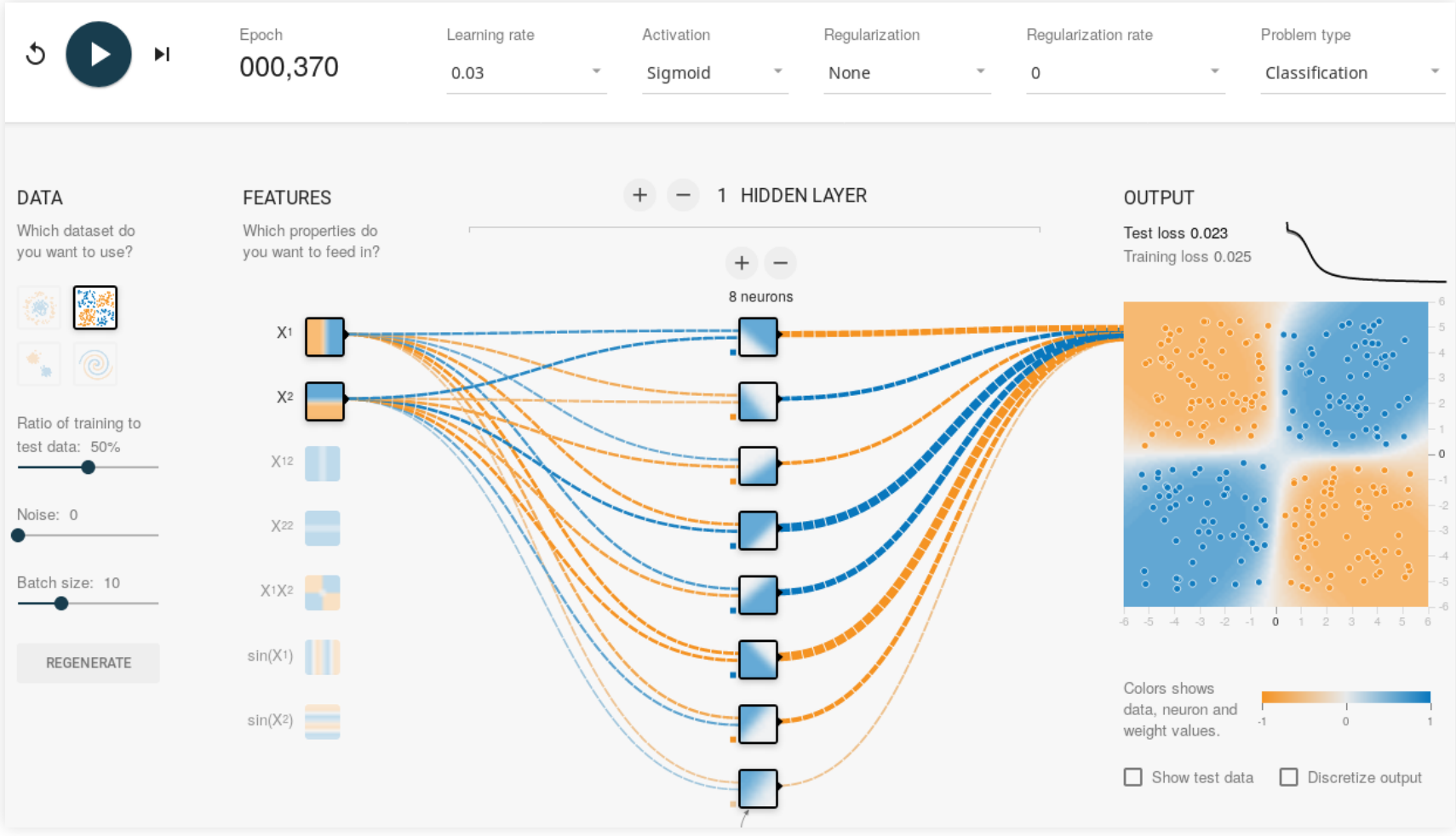
- For a weight m on hidden layer i, we must propagate the output error backwards from all neurons ahead

$$\Delta w^j_{min} = -\eta \left(\sum_p \frac{\delta E^j_{kp}}{\delta s^j_{kp}} \frac{\delta s^j_{kp}}{\delta net^j_{kp}} \frac{\delta net^j_{kp}}{\delta s^j_{in}} \right) \frac{\delta s^j_{in}}{\delta net^j_{in}} \frac{\delta net^j_{in}}{\delta w_{min}}$$

- If δs is small (vanishing gradient) backpropagation becomes ineffective as we increase depth
- This happens with logistic activation (or similar, such as TanH)

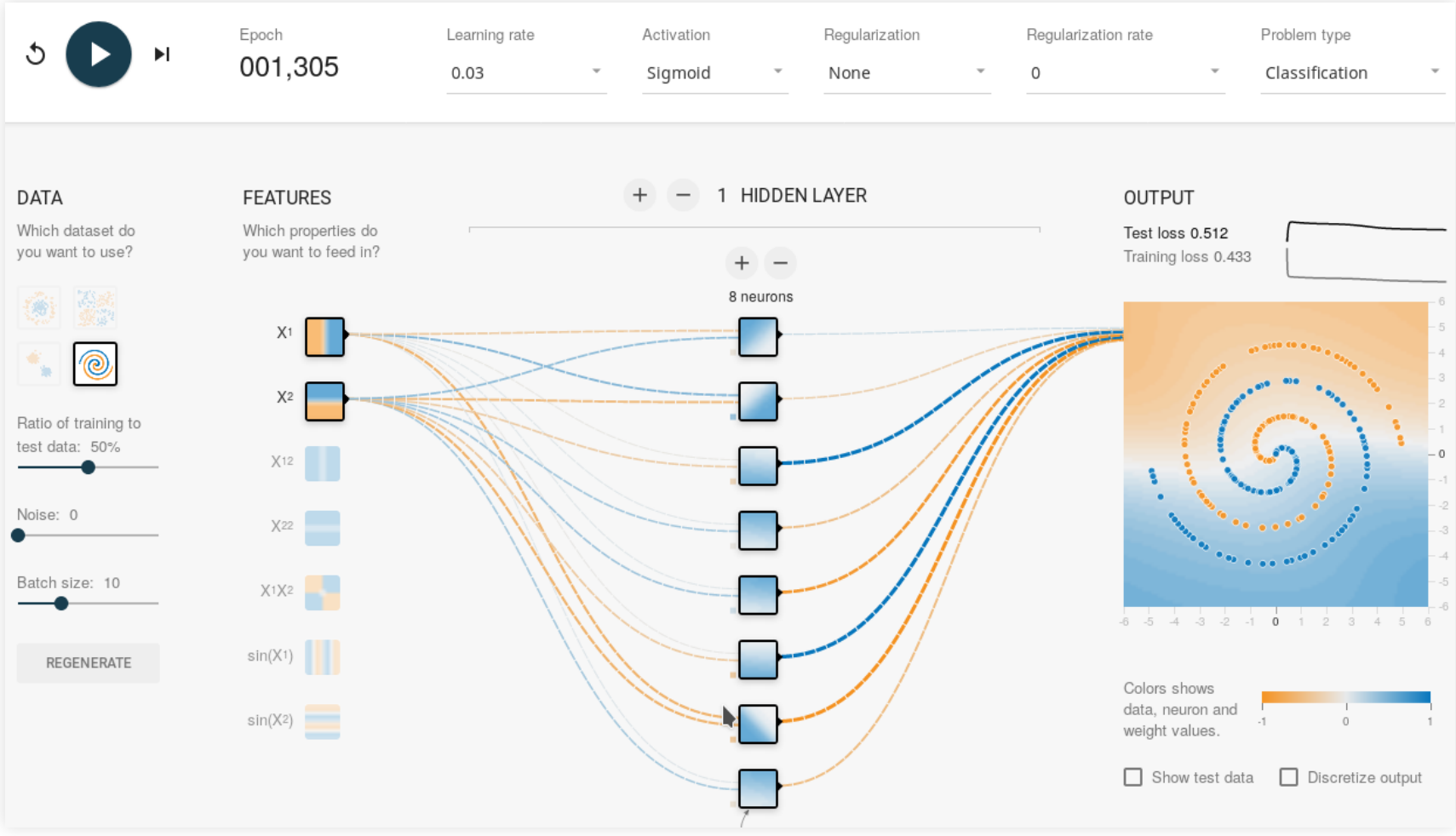
Vanishing gradients

- Single hidden layer, sigmoid, works fine here



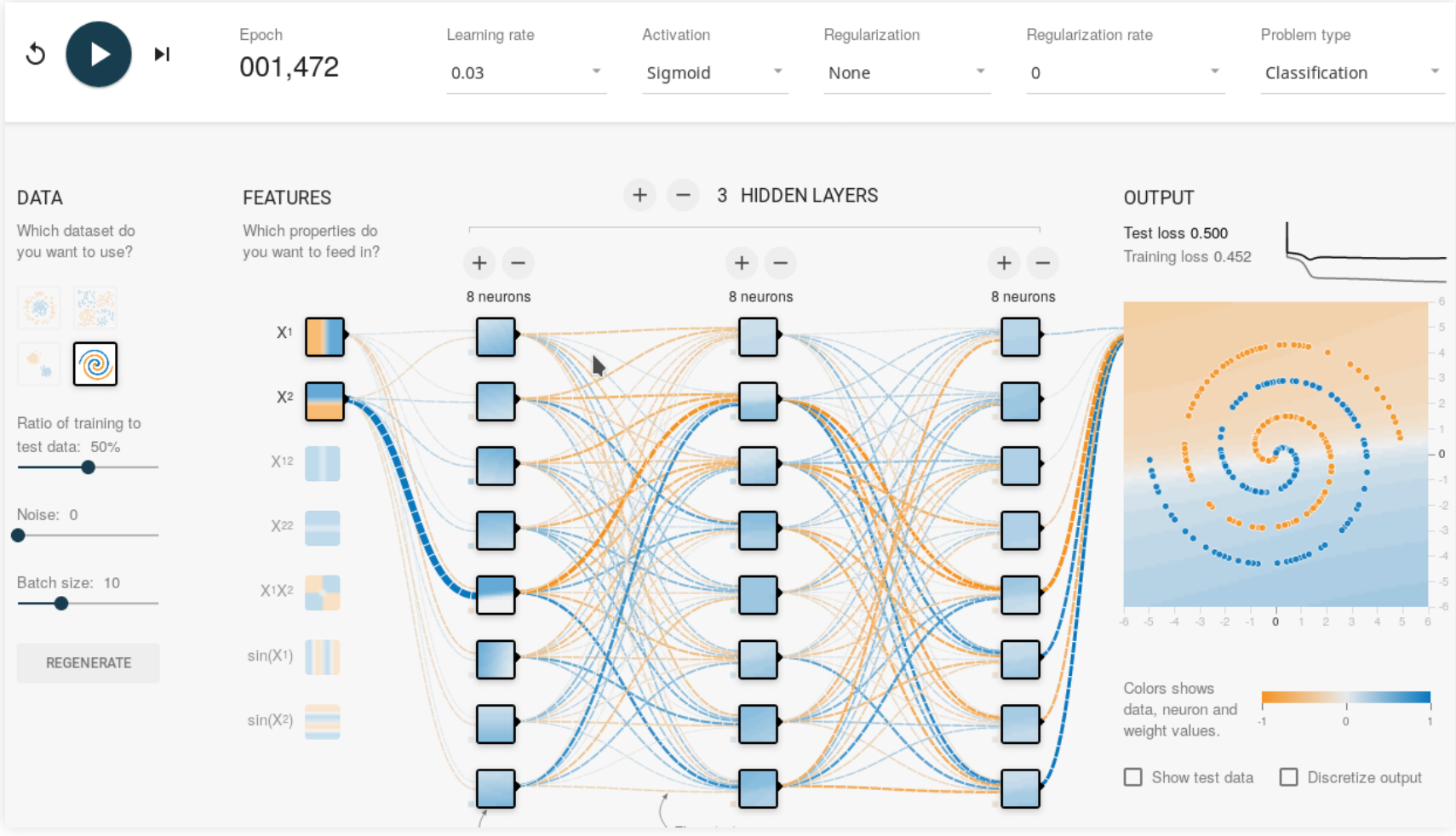
Vanishing gradients

- Single hidden layer, sigmoid, doesn't work here with 8 neurons



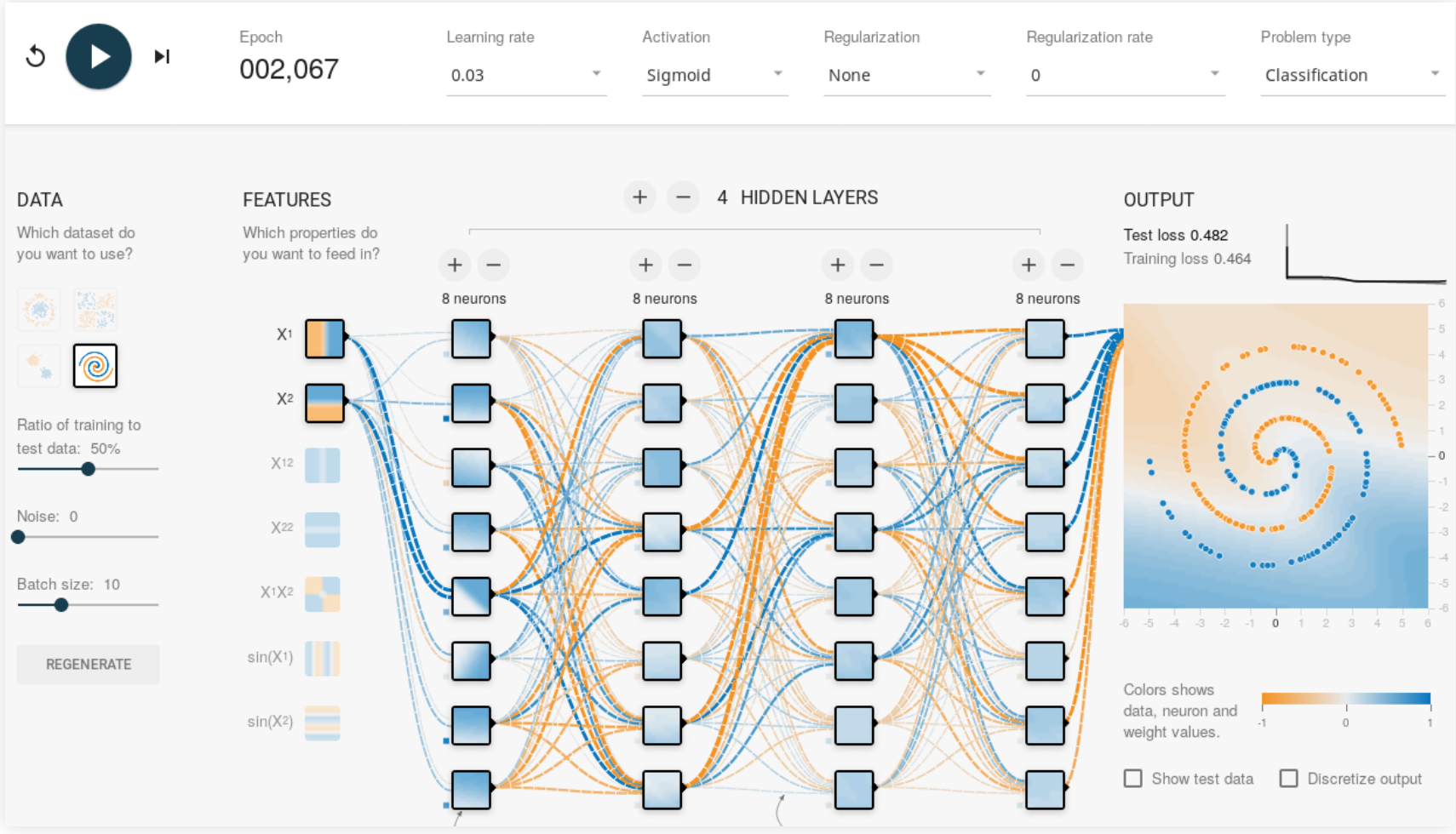
Vanishing gradients

- Increasing depth does not seem to help



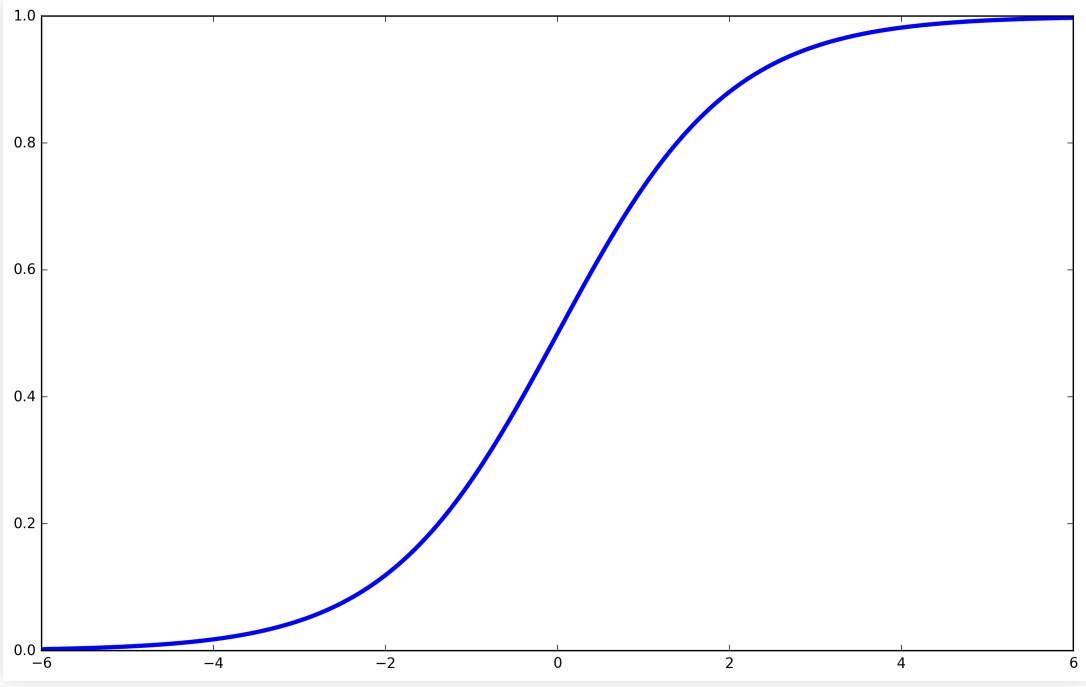
Vanishing gradients

- Increasing depth does not seem to help



Vanishing gradients

- Increasing depth does not seem to help
- Sigmoid activation saturates and gradients vanish with large coefs.

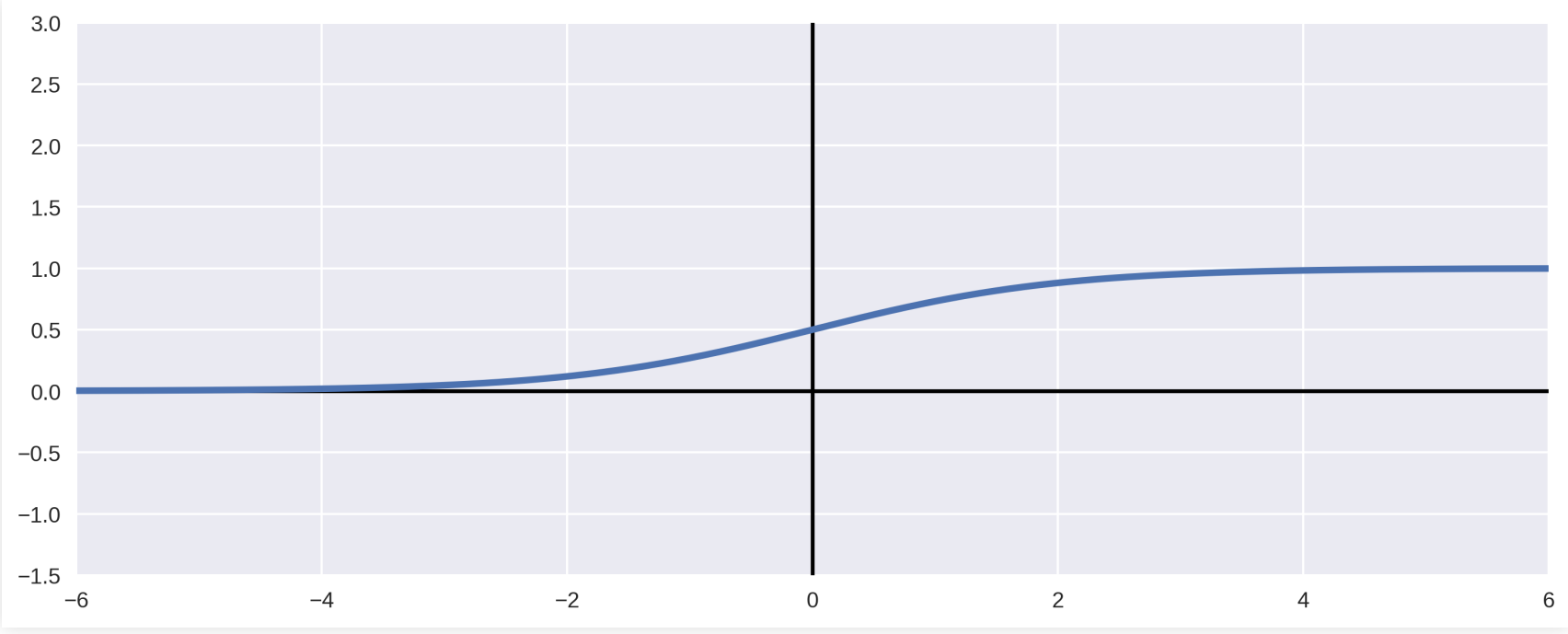


Rectified Linear Unit

Sigmoid

- Sigmoid activation units saturate

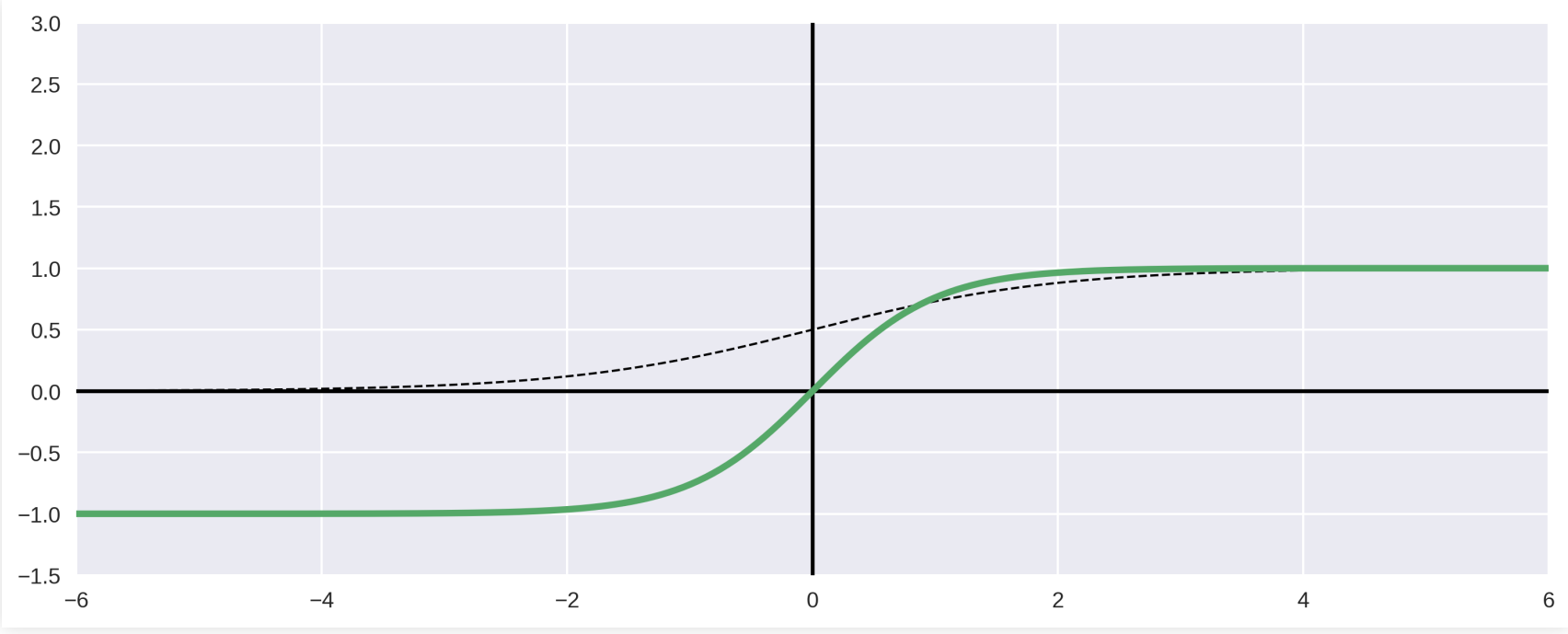
$$y_i = \frac{1}{1 + e^{-x_i}}$$



Other similar activations

- The same happens with hyperbolic tangent

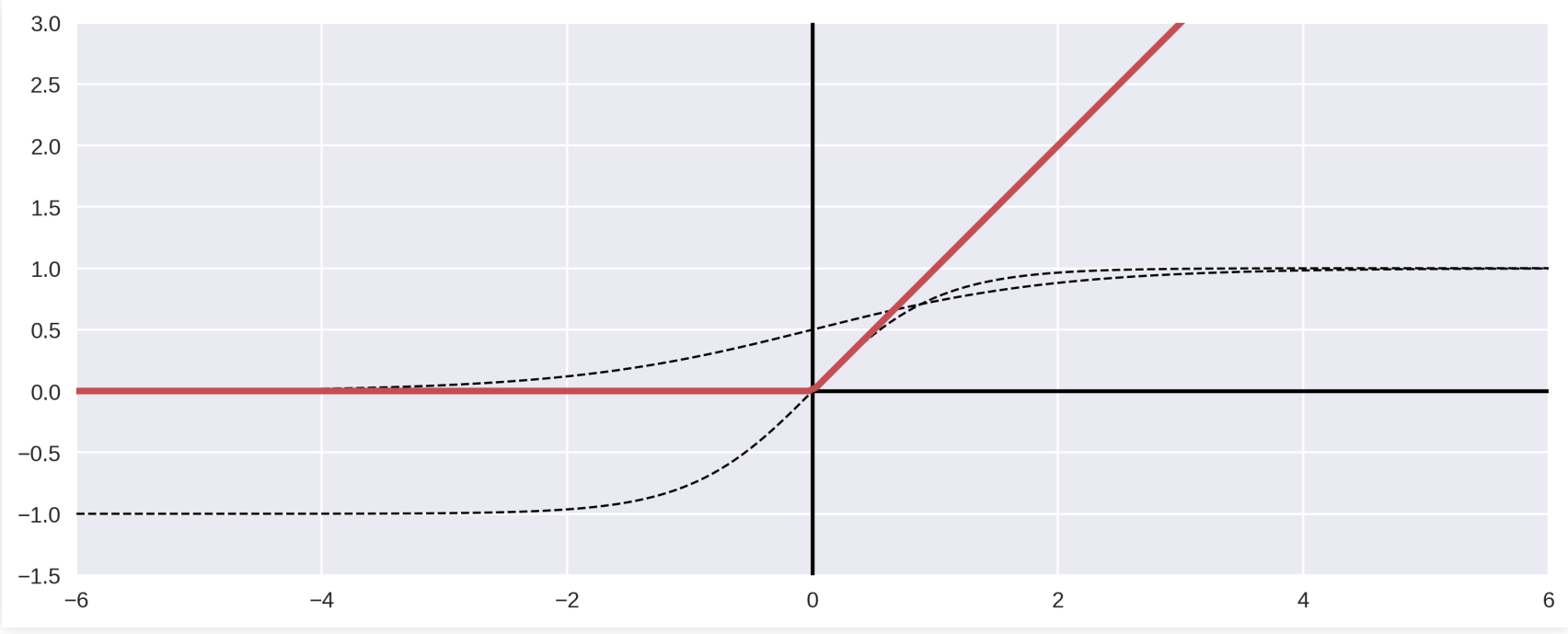
$$y_i = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Rectified Linear Unit (ReLU)

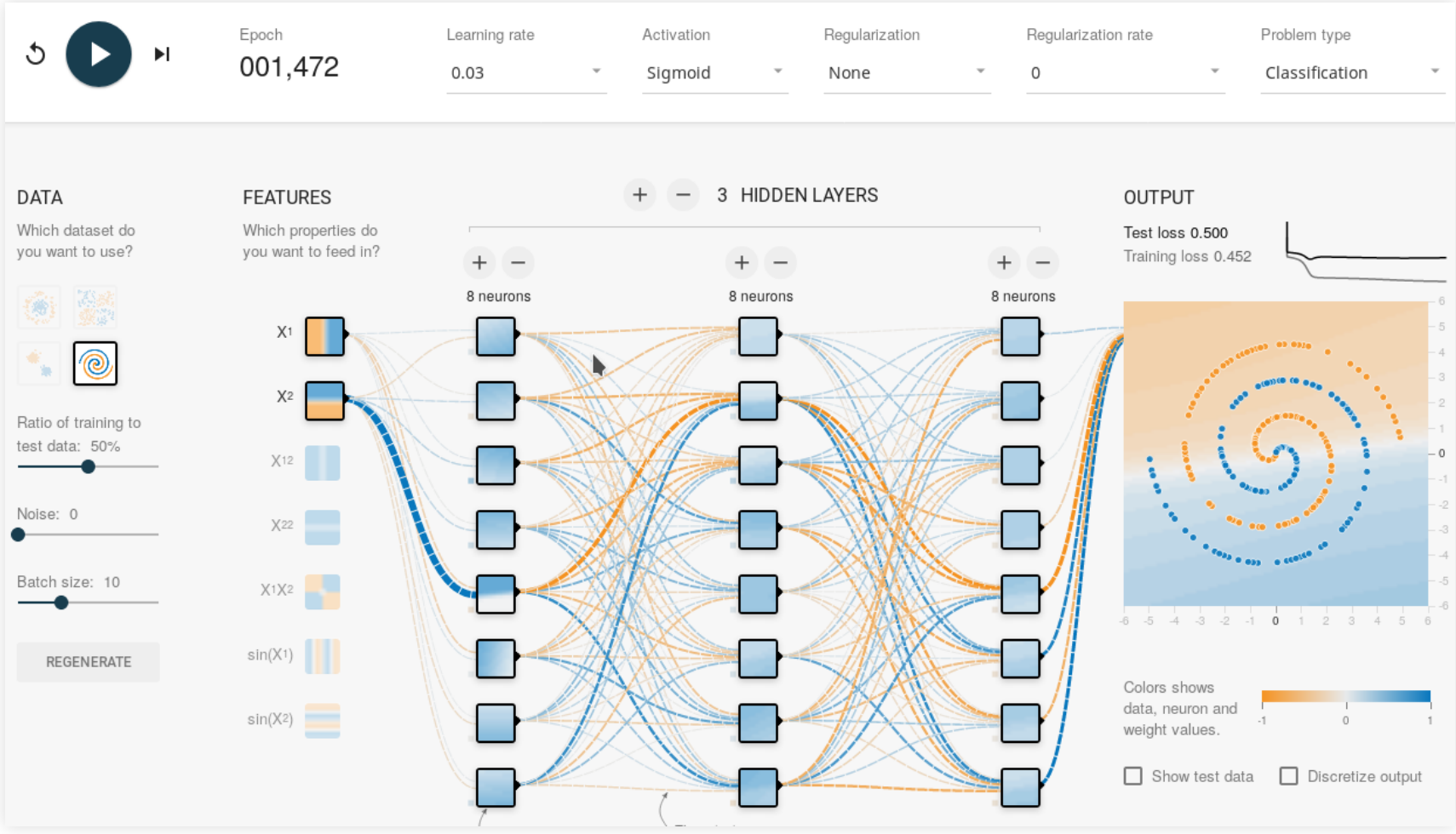
- Rectified linear units do not have this problem

$$y_i = \begin{cases} x_i & x_i > 0 \\ 0 & x_i \leq 0 \end{cases}$$



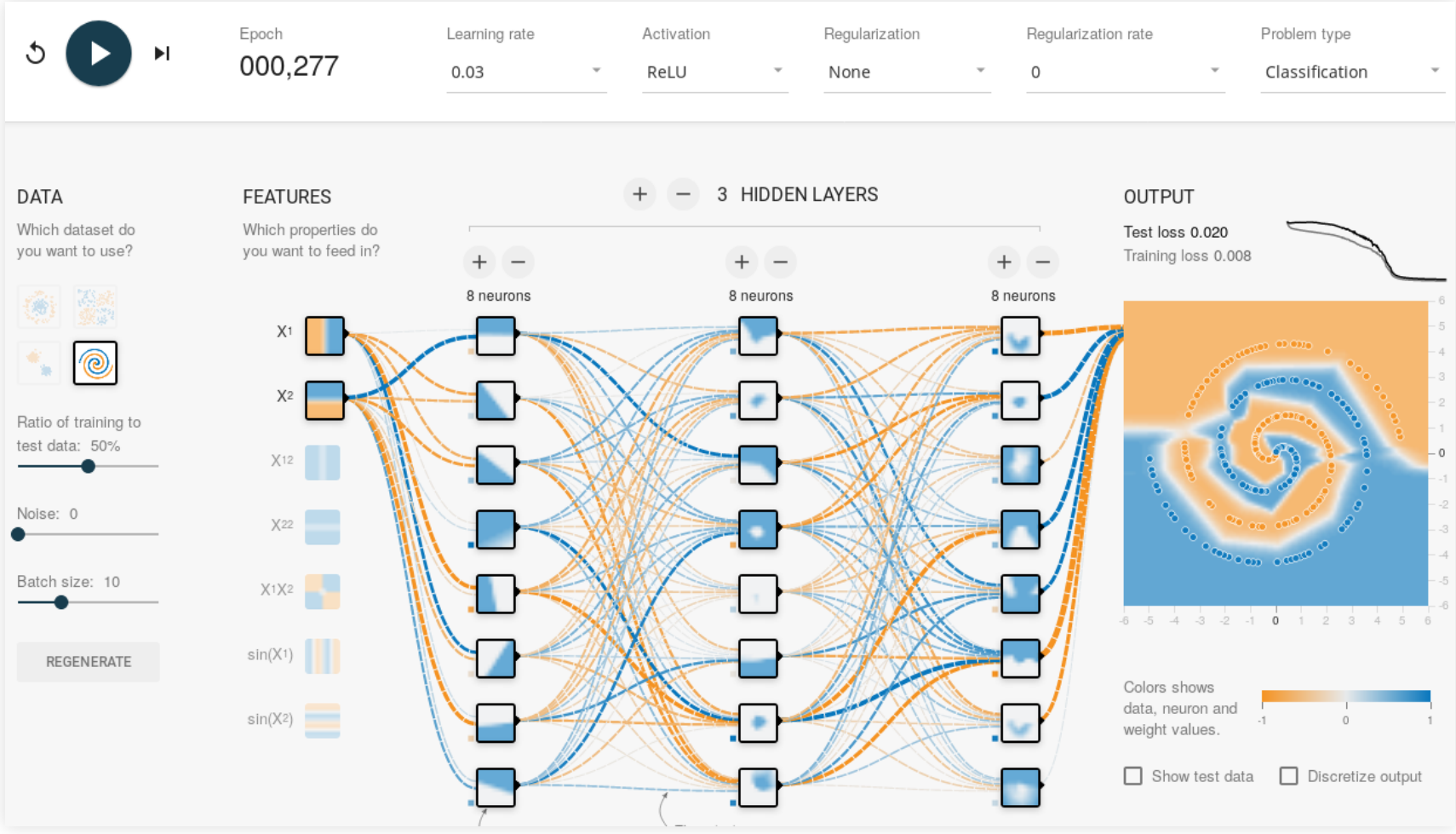
ReLU

- Sigmoid activation, 3 layers



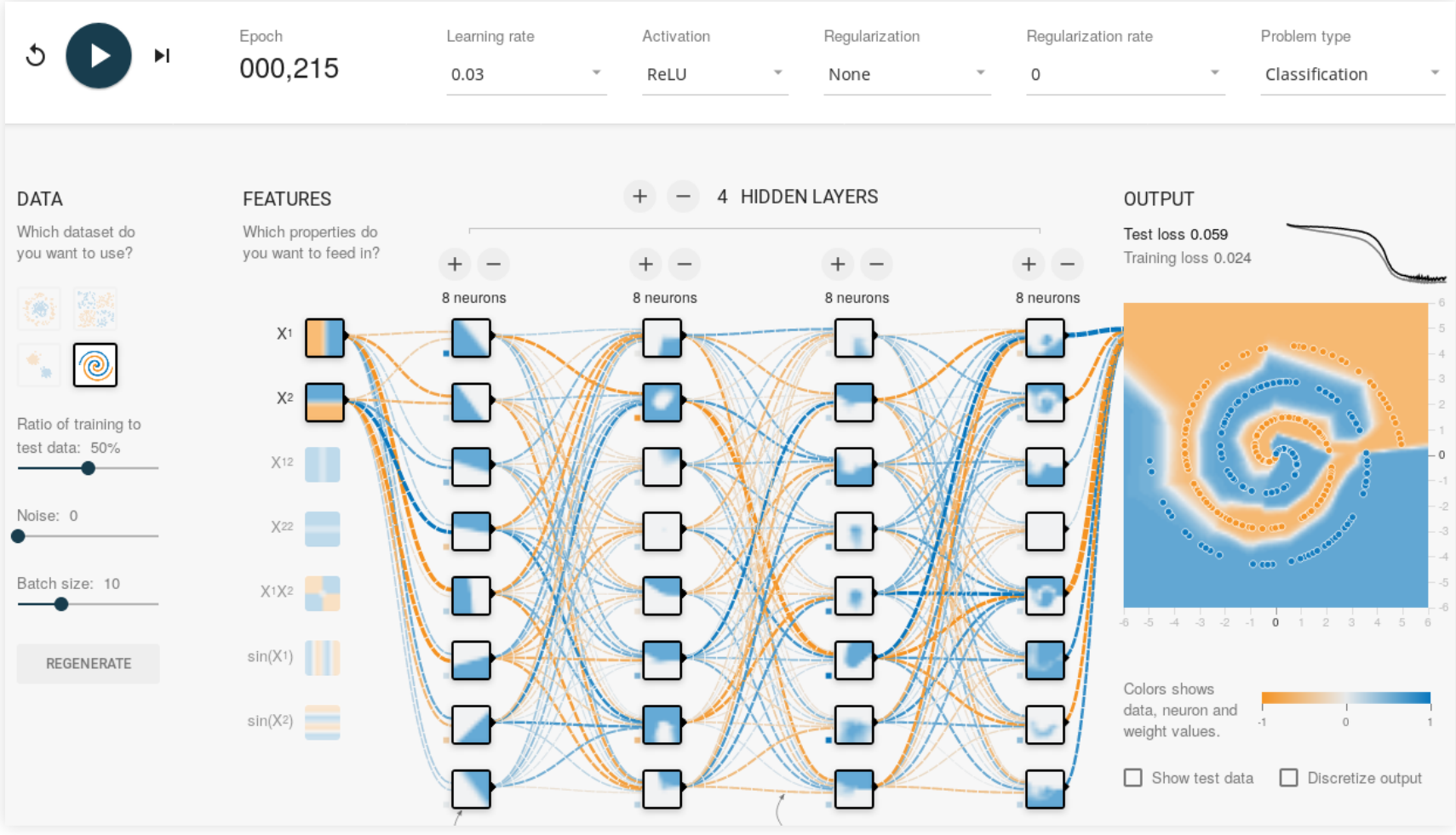
ReLU

- ReLU activation, 3 layers



ReLU

- ReLU activation, 4 layers

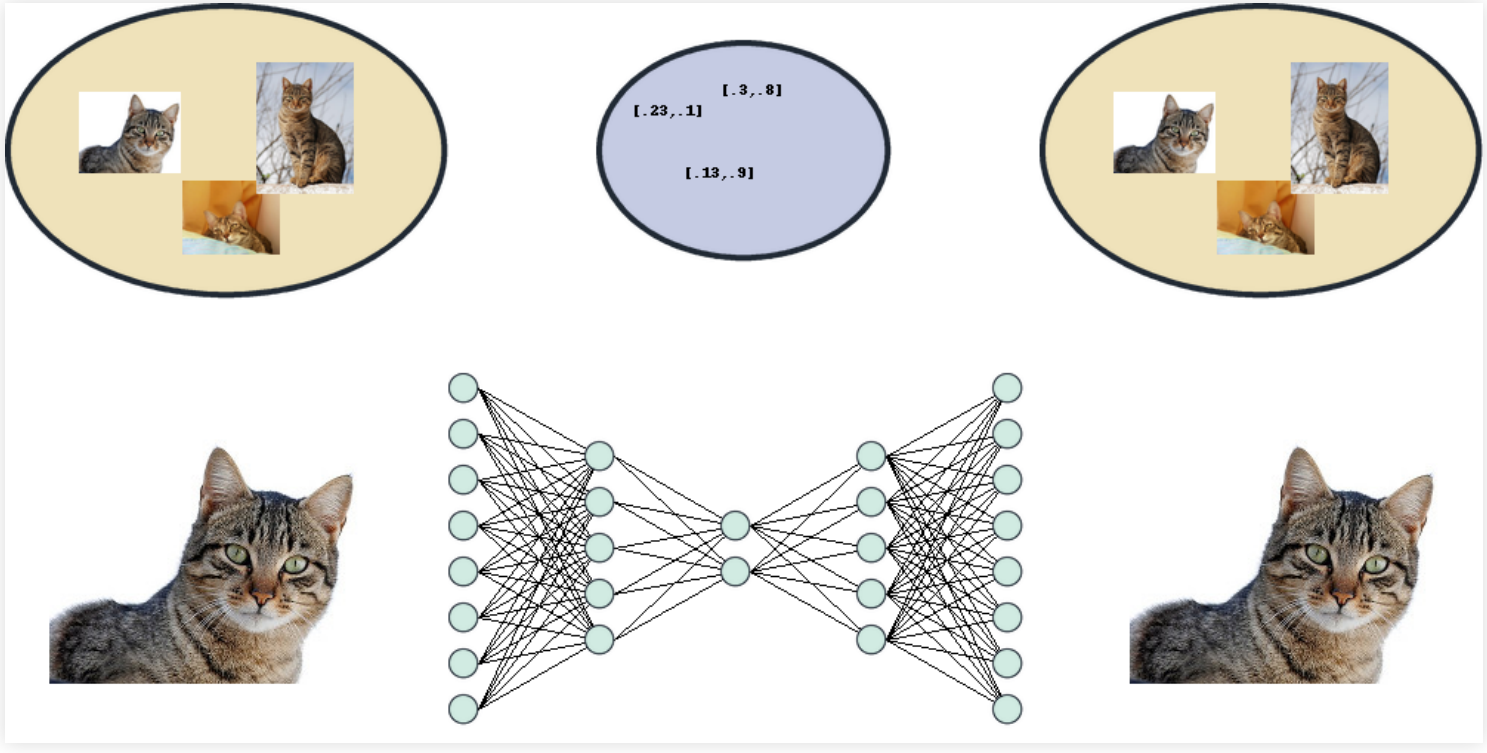


Example: Autoencoders

Autoencoders

Network trained to output the input (unsupervised)

- In the hidden layers, one layer learns a **code** describing the input

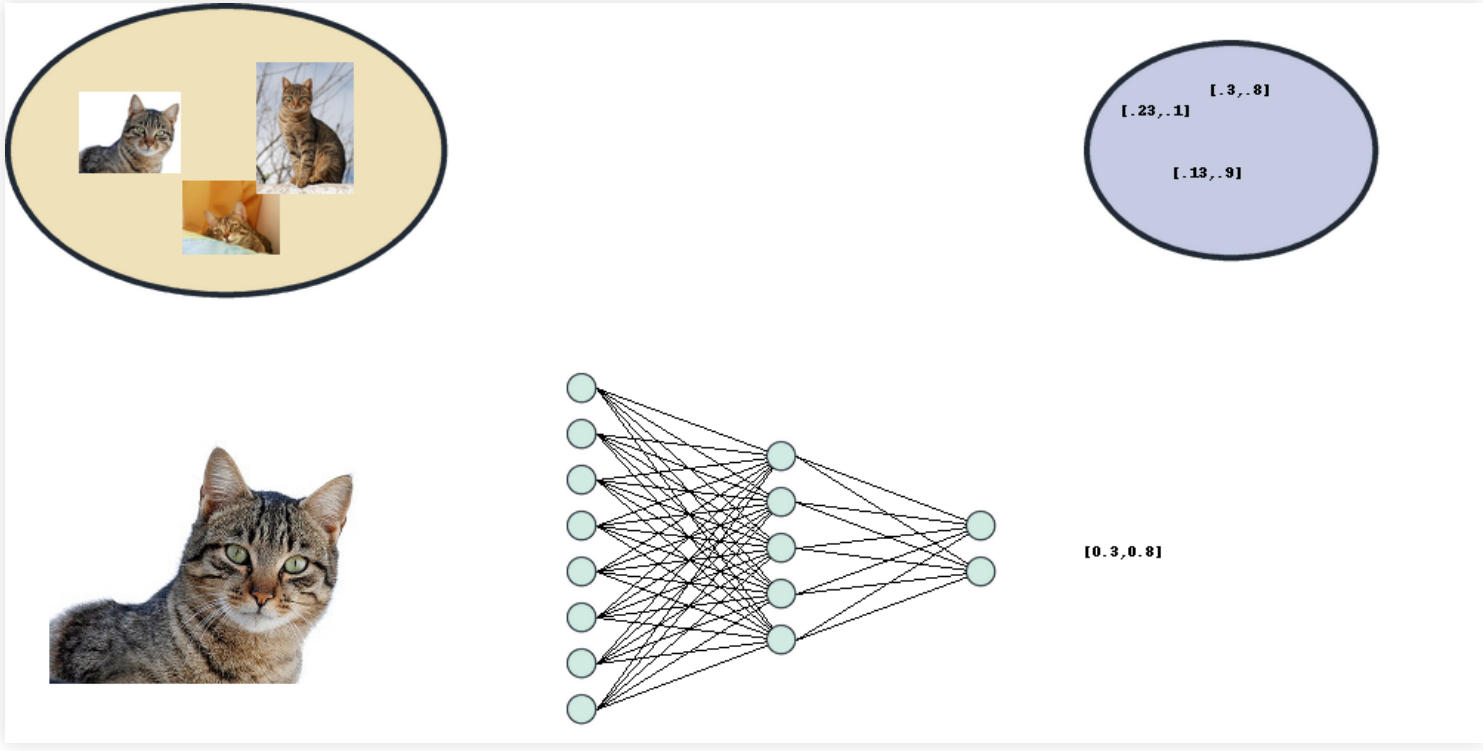


Cat images: Joaquim Alves Gaspar CC-SA

Autoencoders

Network trained to output the input (unsupervised)

- The encoder maps from input to **latent space**

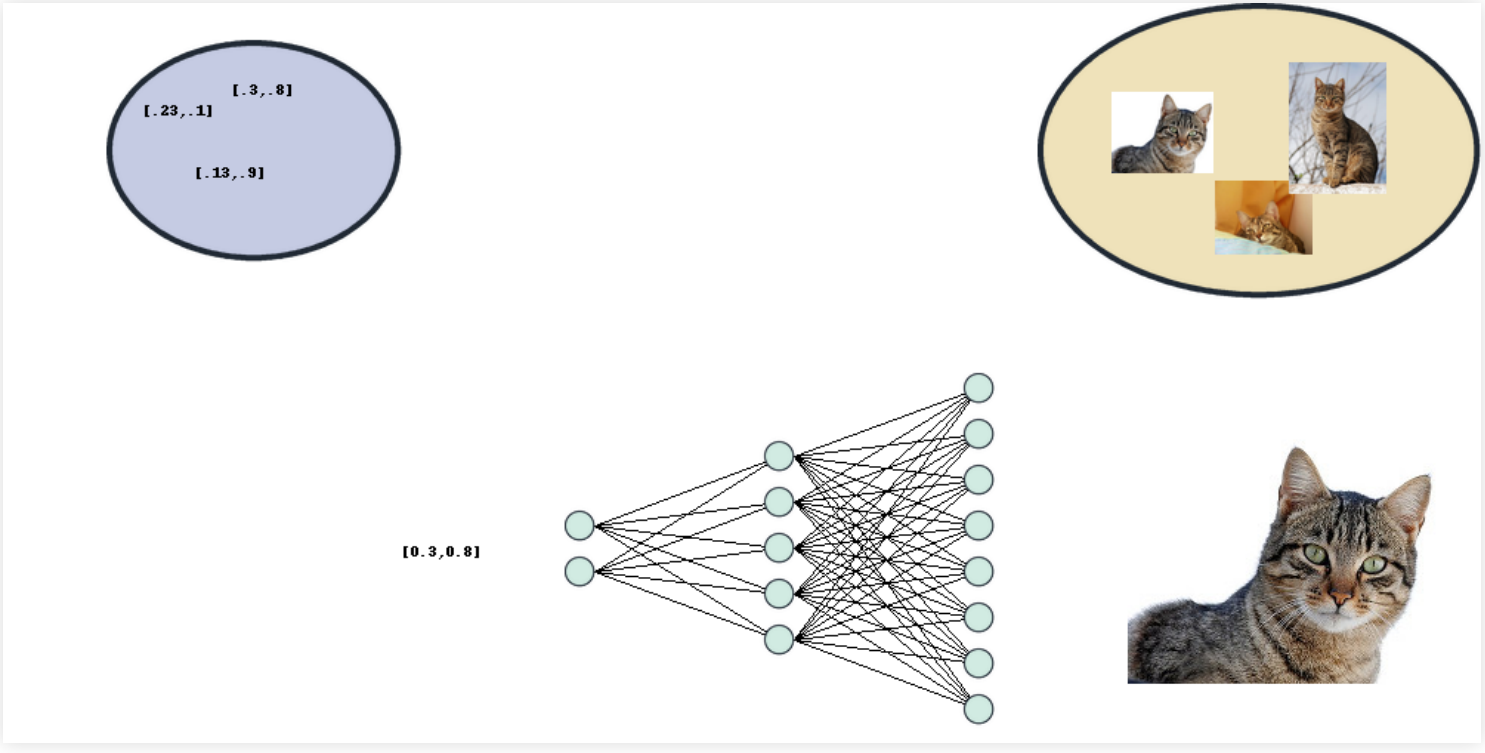


Cat images: Joaquim Alves Gaspar CC-SA

Autoencoders

Network trained to output the input (unsupervised)

- The decoder maps from **latent space** back to input space

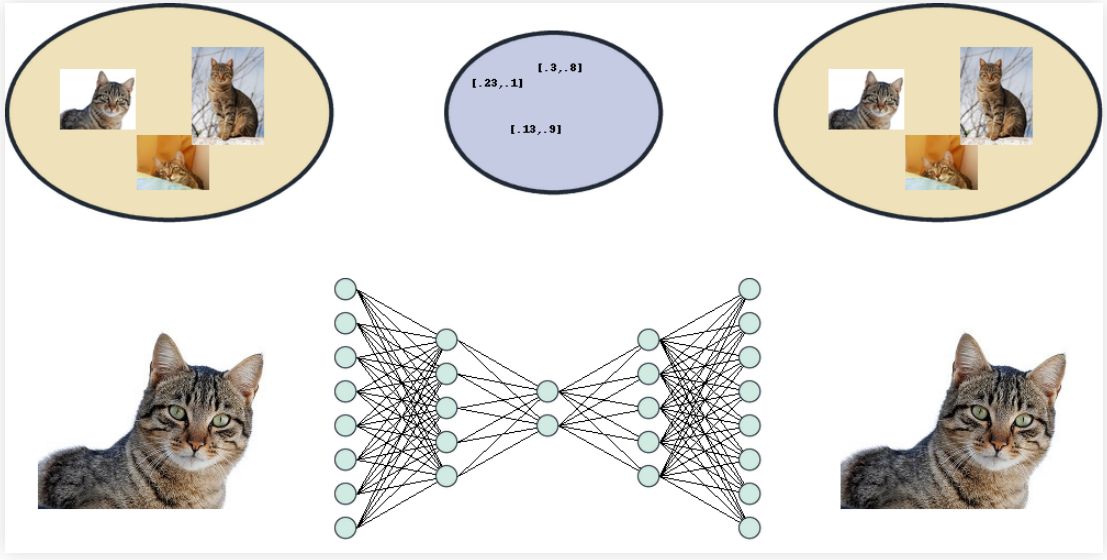


Cat images: Joaquim Alves Gaspar CC-SA

Autoencoders

Network trained to output the input (unsupervised)

- Encoder, $h = f(x)$, and decoder, $x = g(h)$
- No need for labels, since the target is the input
- Why learn $x = g\,(f\,(x))$?



Cat images: Joaquim Alves Gaspar CC-SA

Autoencoders

Network trained to output the input (unsupervised)

- Encoder, $h = f(x)$, and decoder, $x = g(h)$
- Why learn $x = g(f(x))$?
- Latent representation can have advantages

- Lower dimension
- Capture structure in the data
- Data compression

- As long as we can force the autoencoder to do something useful

Autoencoders

Network trained to output the input (unsupervised)

- Encoder, $h = f(x)$, and decoder, $x = g(h)$
- Autoencoders are just feedfoward networks

- Can be trained with the same algorithms, such as backpropagation

- But since the target is x, they are unsupervised learners
- Need some "bottleneck" to force a useful representation

- Otherwise just copies values

More like how we learn?

- Children do not learn with labelled examples
- We learn by exposure to mostly unlabelled examples

Different types of autoencoders

Undercomplete Autoencoders

Autoencoder is undercomplete if h is smaller than x

- Forces the network to learn reduced representation of input
- Trained by minimizing a loss function

$$L(x, g(f(x)))$$

  that penalizes the difference between x and $g(f(x))$
- If linear it is similar to PCA (without orthogonality constraint)
- With nonlinear transformations, an undercomplete autoencoder can learn more powerful representations
- However, we cannot overdo it

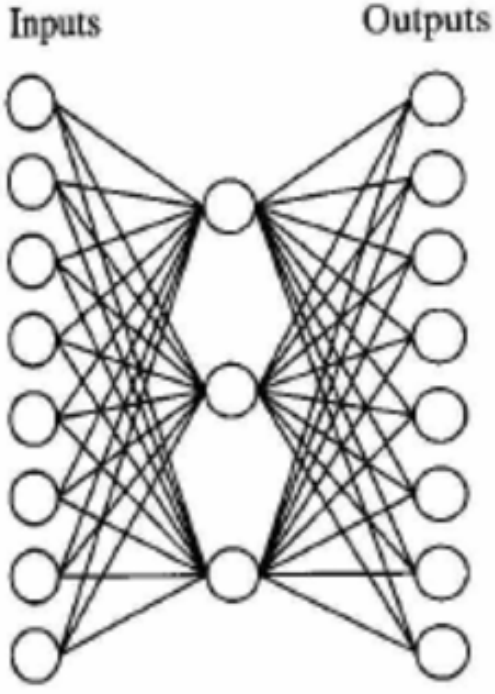
- With too much power, autoencoder can just index each training example and learn nothing useful:

$$f(x_i) = i, \quad g(i) = x_i$$

Undercomplete Autoencoders

Autoencoder is undercomplete if h is smaller than x

- Mitchell's autoencoder, hidden layer of 3 neurons



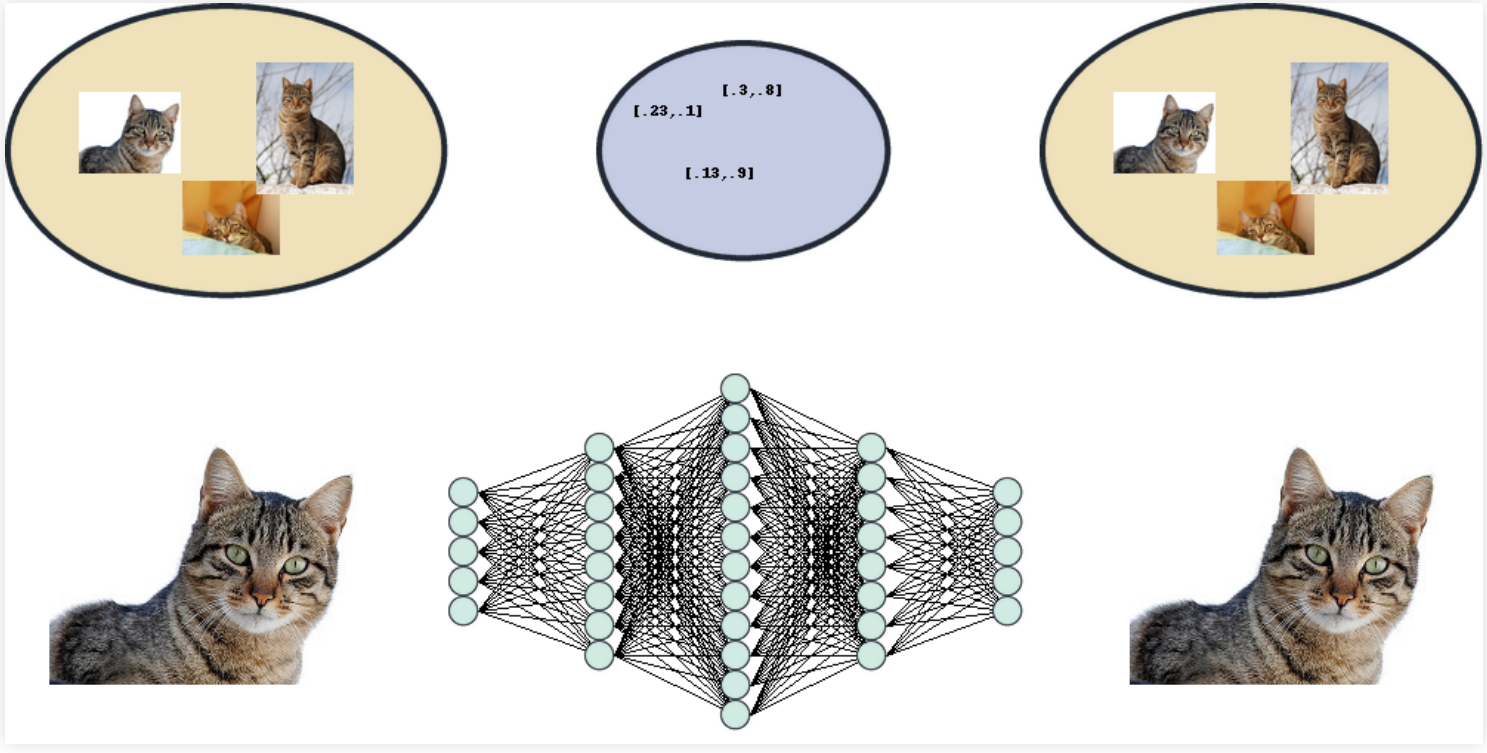
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Tom M. Mitchell, Machine Learning, McGraw Hill 1997

Regularized Autoencoders

An overcomplete autoencoder has $h \geq x$

- This, by itself, is a bad idea as h will not represent anything useful



Cat images: Joaquim Alves Gaspar CC-SA

Contractive Autoencoders

An overcomplete autoencoder has $h \geq x$

- But we can restrict h with regularization
- This way the autoencoder also learns how restricted h should be

Example: contractive autoencoder

- Penalize the derivatives of activation w.r.t. inputs

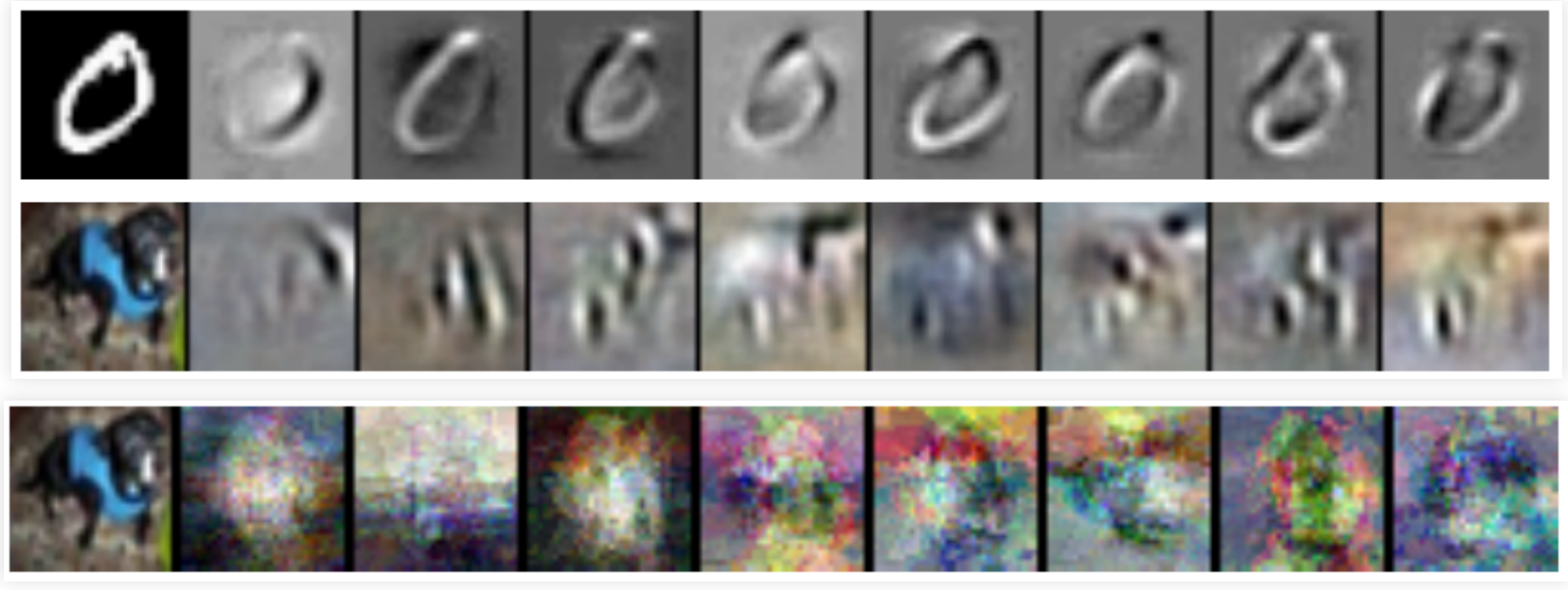
$$L(x, g(f(x))) + \lambda \sum_i \|\nabla_x h_i\|^2$$

- This makes h less sensitive to small variations in the input
  - More robust encoding; similar examples are close together
- It also favours smaller weights
  - (the derivative of the activation w.r.t. the input depends on the weights)
- This restricts h to important aspects of the input distribution

Contractive Autoencoders

- CAE approximate the manifold where data is distributed

- Allows for better learning with fewer data, compared to e.g. PCA



Leading SV of the Jacobian (Rifai et. al., Manifold Tangent Classifier, 2011)

Sparse Autoencoders

Force h to have few activations (sparse)

- Example: we want the probability of h_i firing

$$\hat{\rho}_i = \frac{1}{m} \sum_{j=1}^{m} h_i(x_j)$$

  to be equal to ρ (the sparseness parameter)

- We can use the Kullback-Leibler divergence between Bernoulli variables as a regularization penalty

$$L(x, g(f(x))) + \lambda \sum_i \left(\rho \log \frac{\rho}{\hat{\rho}_i} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_i} \right)$$

- Note: sparseness may be due to small, nonzero, activations

- But it can correspond to zero activation in neurons if using ReLU

Sparse Autoencoders

Sparse autoencoders make neurons specialize

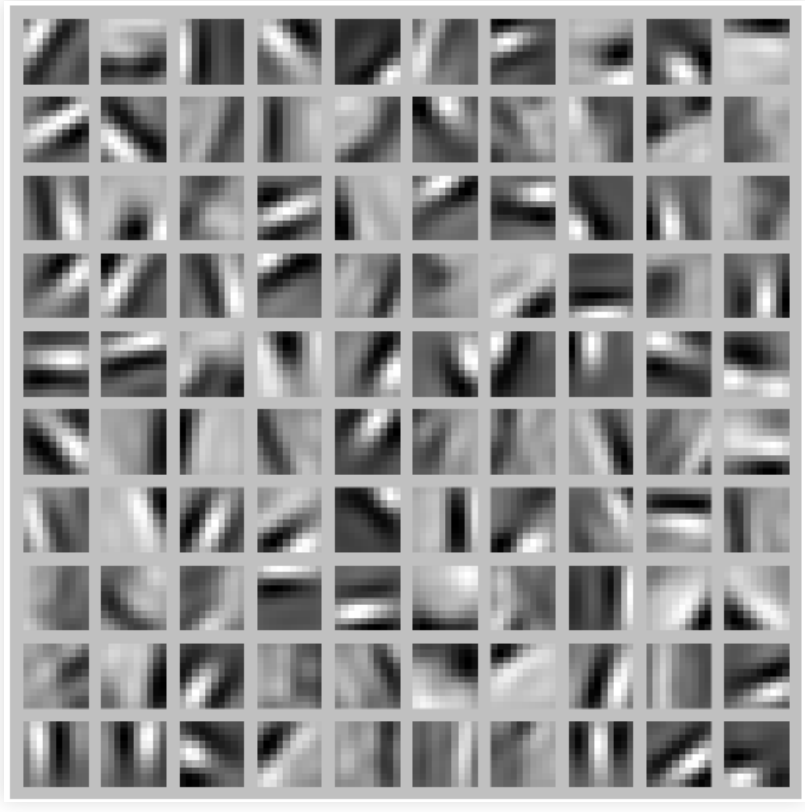


Image: Andrew Ng

- Trained on 10x10 images
- 100 neurons on h
- Images (norm-bounded) that maximize activation

Denoising Autoencoders

We can force h to be learned with noisy inputs

- Output the original x from corrupted $\tilde{x}$: $L(x, g(f(\tilde{x})))$

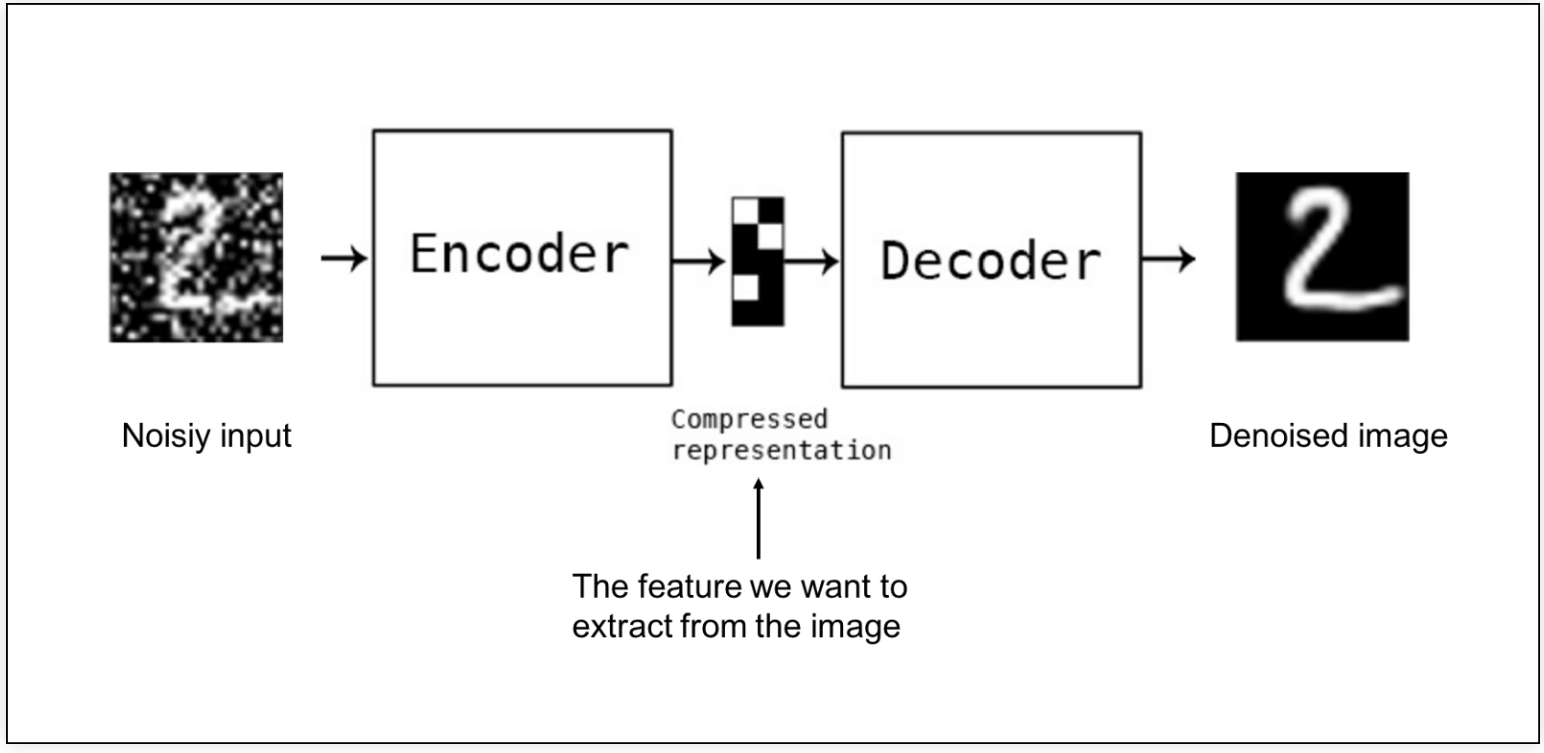
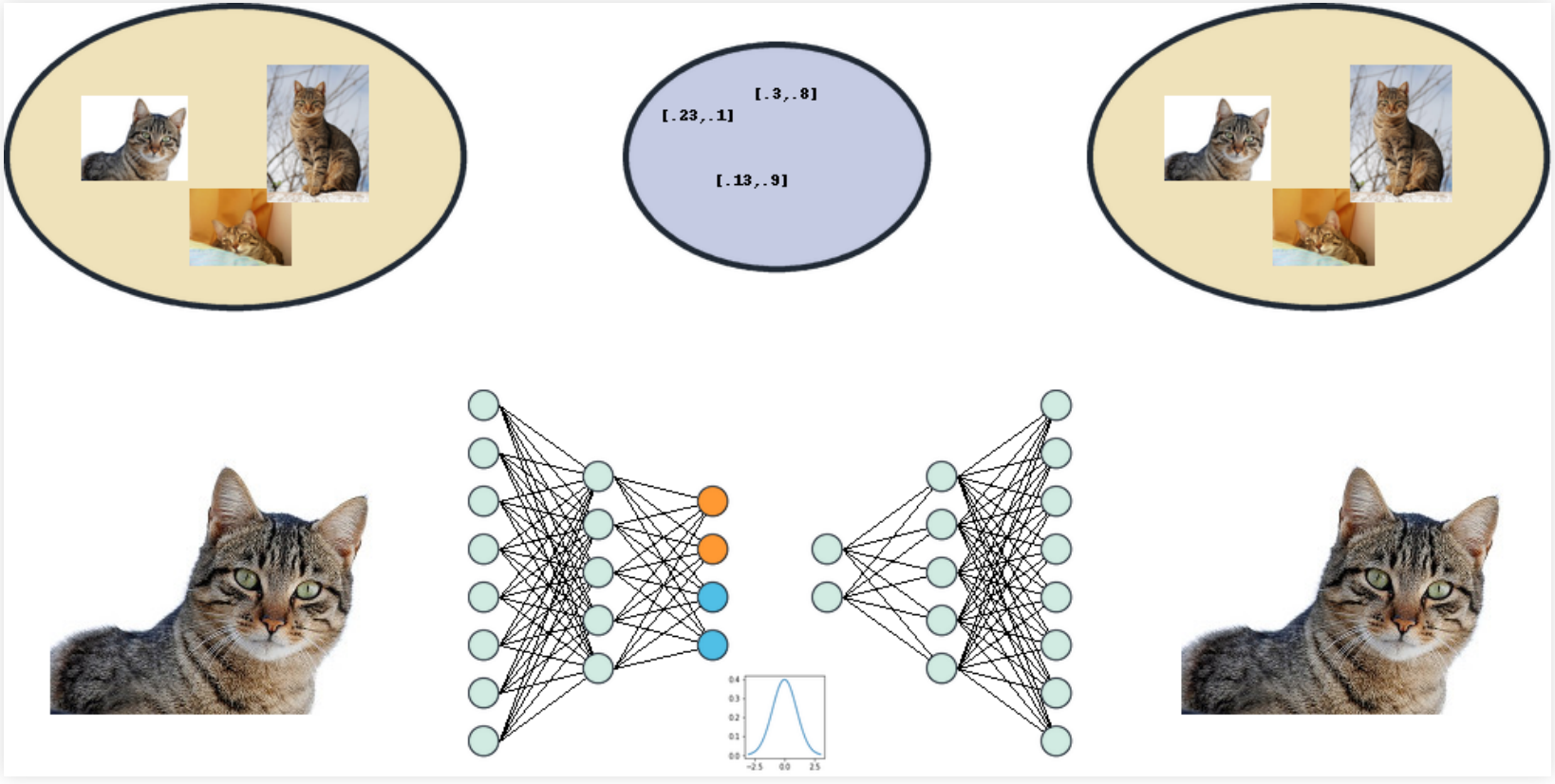


Image: Adil Baaj, Keras Tutorial on DAE

Variational Autoencoders

- Train the autoencoder to encode into a known distribution

- E.g. mixture of independent Gaussians



Variational Autoencoders

- How do we backpropagate through random sampling?
- Reparametrize: z is deterministic apart from a normally distributed error

$$z = \mu + \sigma \odot \epsilon \qquad \epsilon \sim \mathcal{N}(0, 1)$$

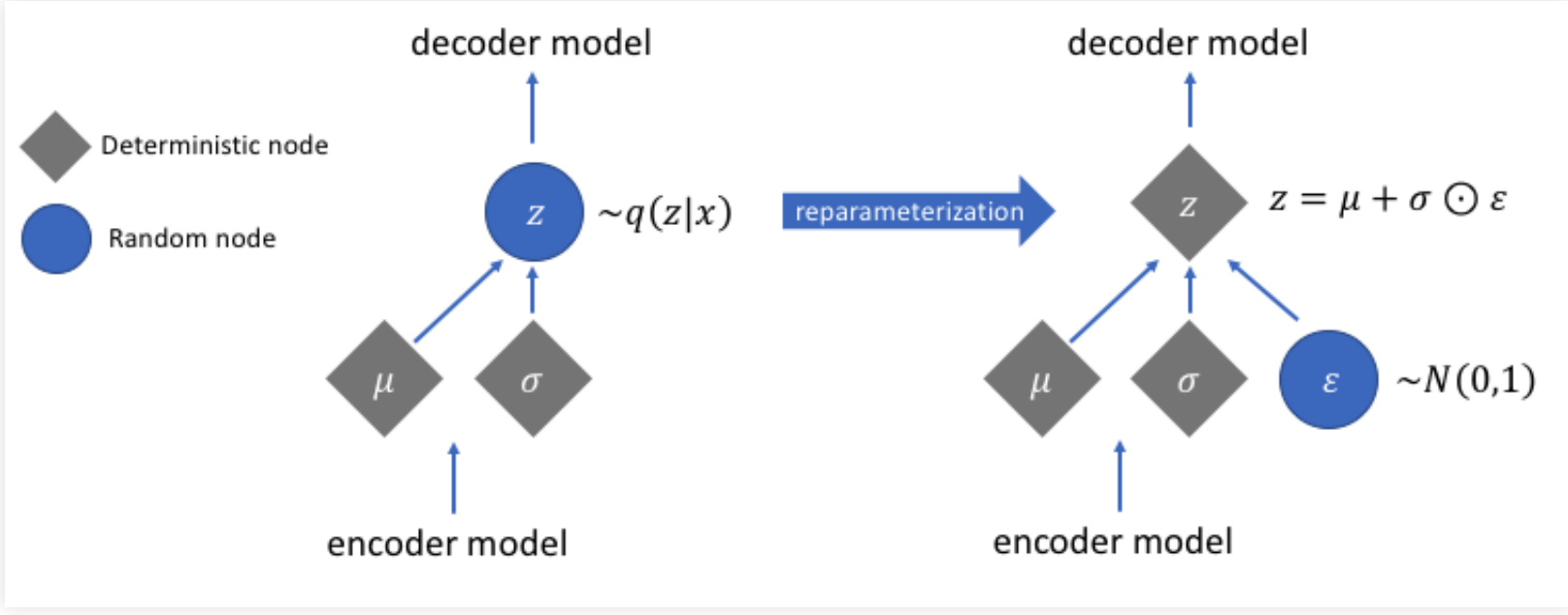


Image: Jeremy Jordan, Variational autoencoders.

Variational Autoencoders

- VAE can learn to disentangle meaningful attributes

- We can force the independence of the latent variables

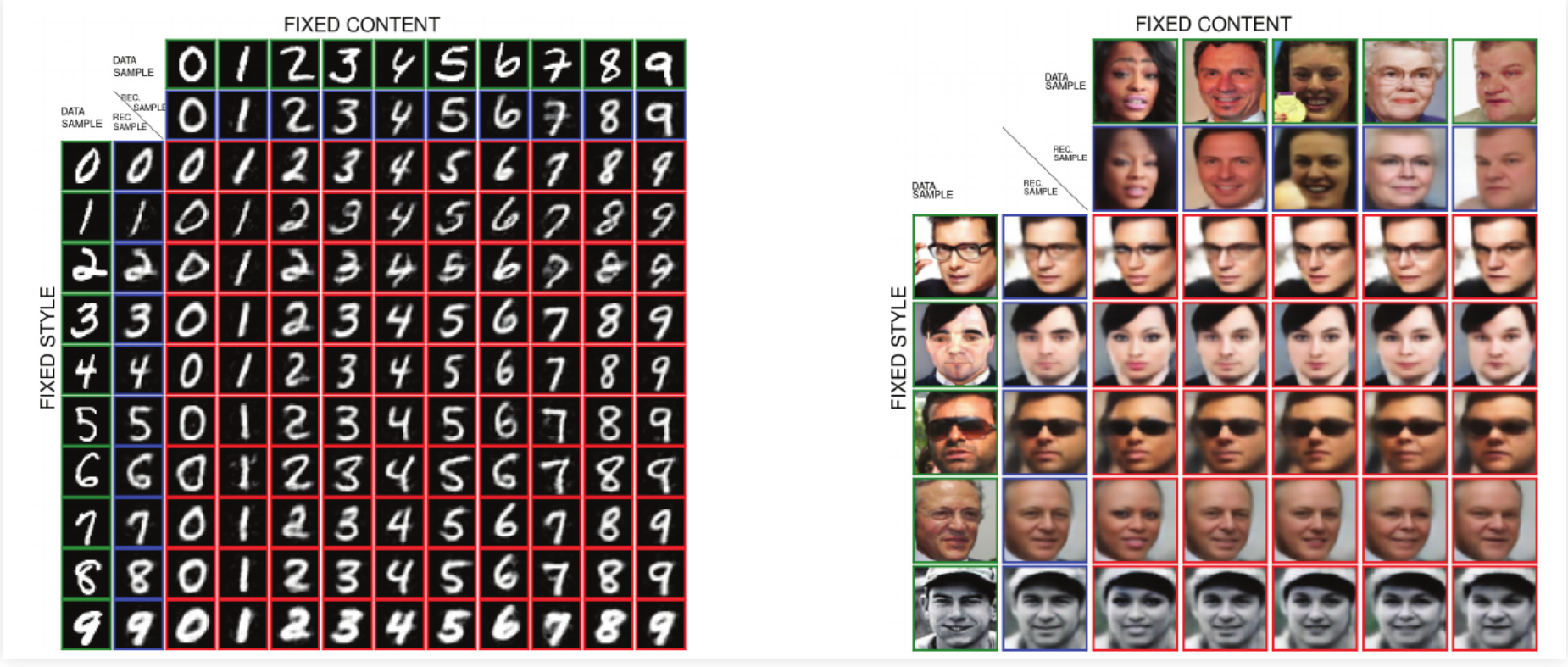
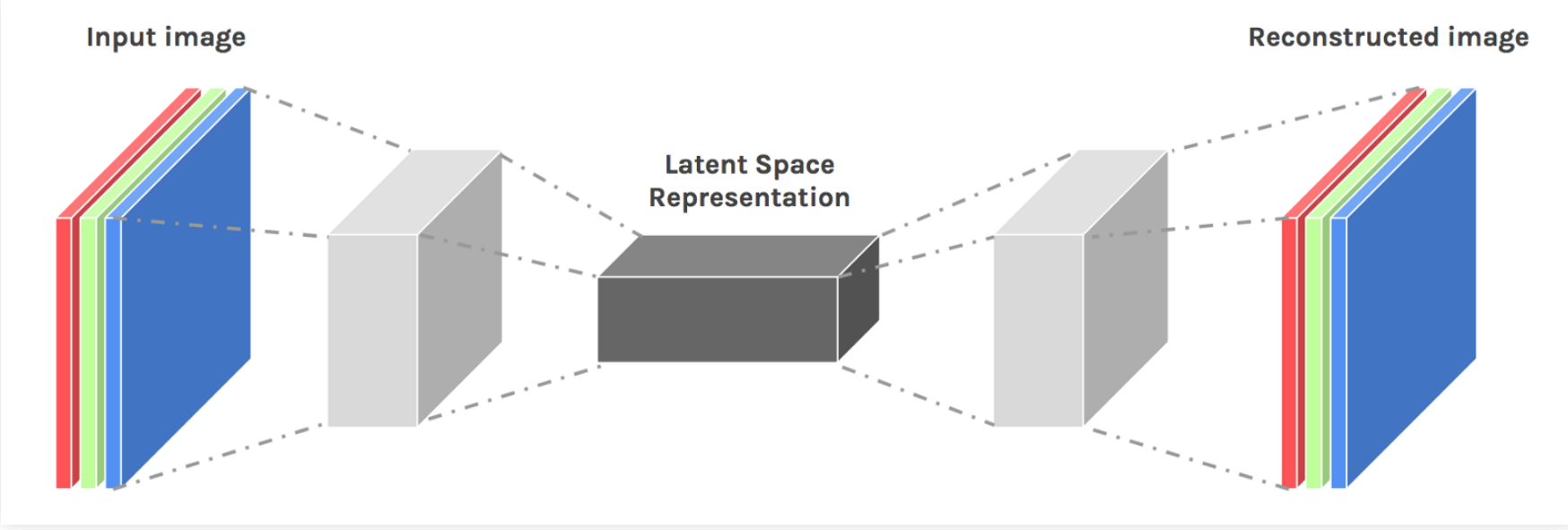


Image: Bouchacourt et. al., Multi-level variational autoencoder, 2018

Convolutional Autoencoders

Use convolutions and "deconvolutions" to reconstruct

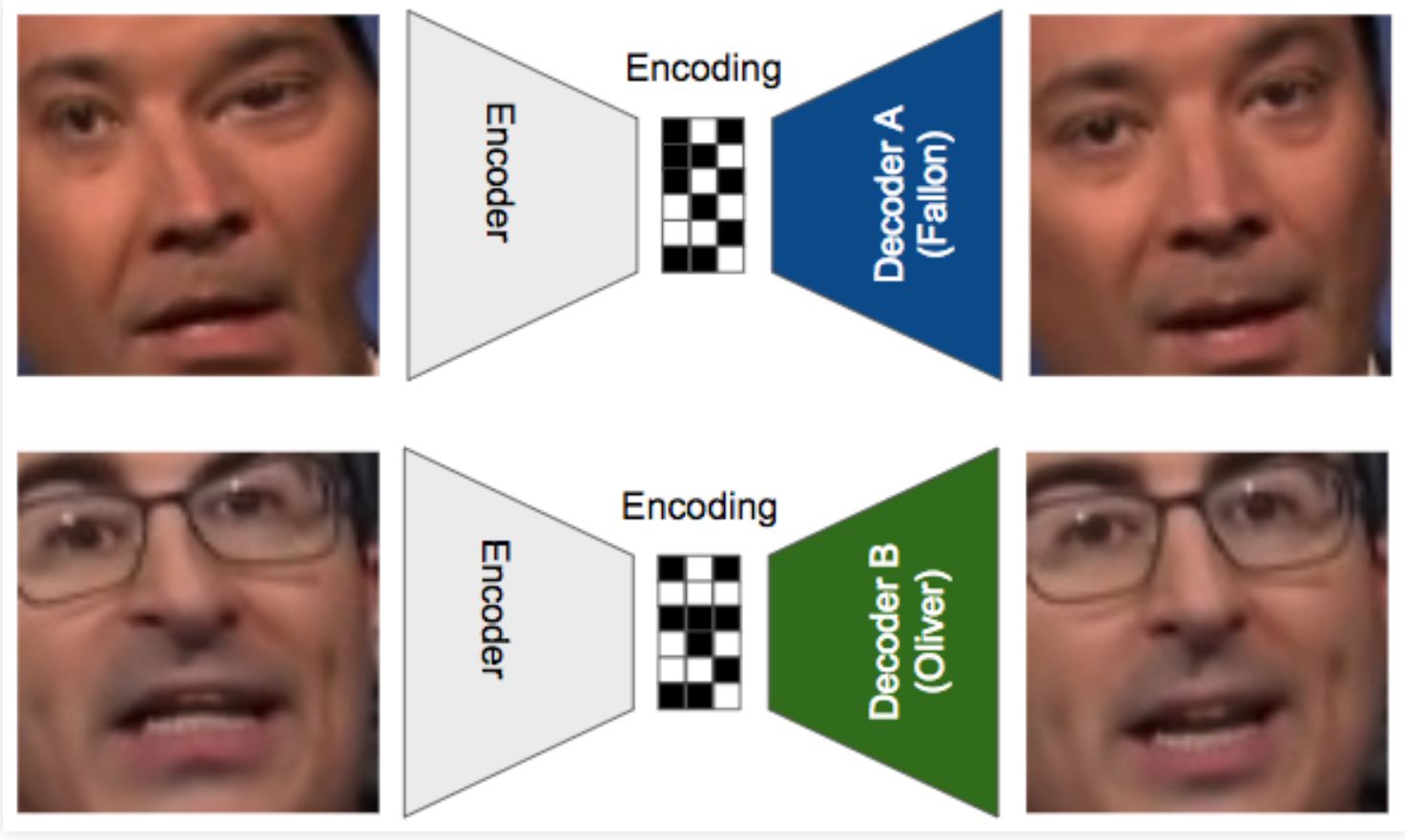
- Latent space is narrow, need to restore original dimensions



Barna Pásztor, Aligning hand-written digits with Convolutional Autoencoders

Convolutional Autoencoders

Example: deep fakes



Gaurav Oberoi, Exploring DeepFakes, https://goberoi.com/exploring-deepfakes-20c9947c22d9

Convolutional Autoencoders

Example: deep fakes

- Train with images from videos
- Process video:

- Input Fallon to encoder
- Output Oliver using Oliver decoder

Gaurav Oberoi, Exploring DeepFakes, https://goberoi.com/exploring-deepfakes-20c9947c22d9

Summary

Deep Learning

Summary

- The importance of nonlinear transformations
- Deep learning: stacking nonlinear transformations
- Recent sucess: data and tecnhiques
- Many applications:

• Classification, reinforcement learning, clustering, ...

- Autoencoders

• (because they are my favourite)

Further reading:

- LeCun, Bengio & Hinton, Deep learning, Nature 2015