Deep Learning Techniques for Music Generation (4)

Jean-Pierre Briot

Jean-Pierre.Briot@lip6.fr

Laboratoire d'Informatique de Paris 6 (LIP6)
Sorbonne Université – CNRS

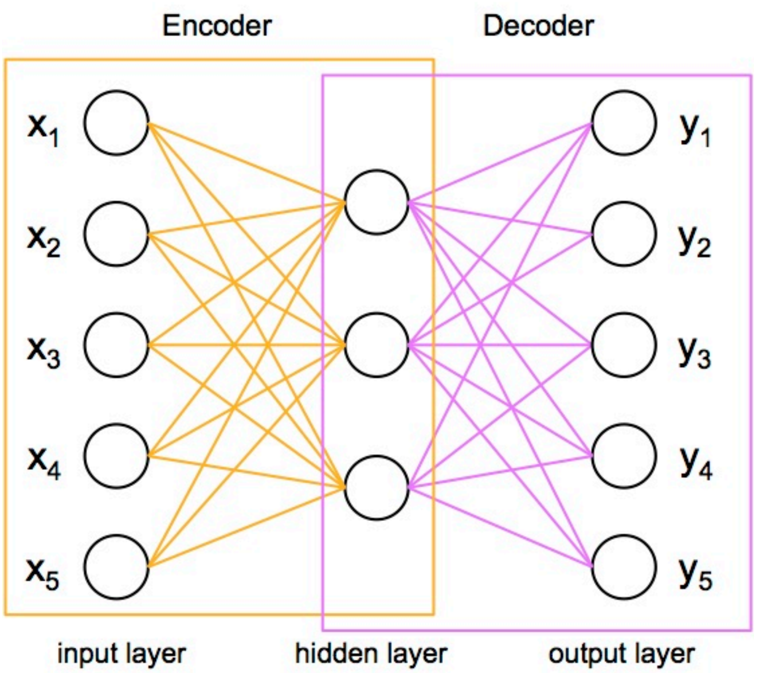
Programa de Pós-Graduação em Informática (PPGI)
UNIRIO

Autoencoder

#1 Limitation – Generation (Without or With Minimal Input)

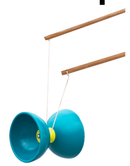
- #1 Partial Solution
- Via Decoding (Autoencoder)
  - » Ex: DeepHear [Sun, 201X]

Autoencoder



Neural Network with Input Layer = Output Layer

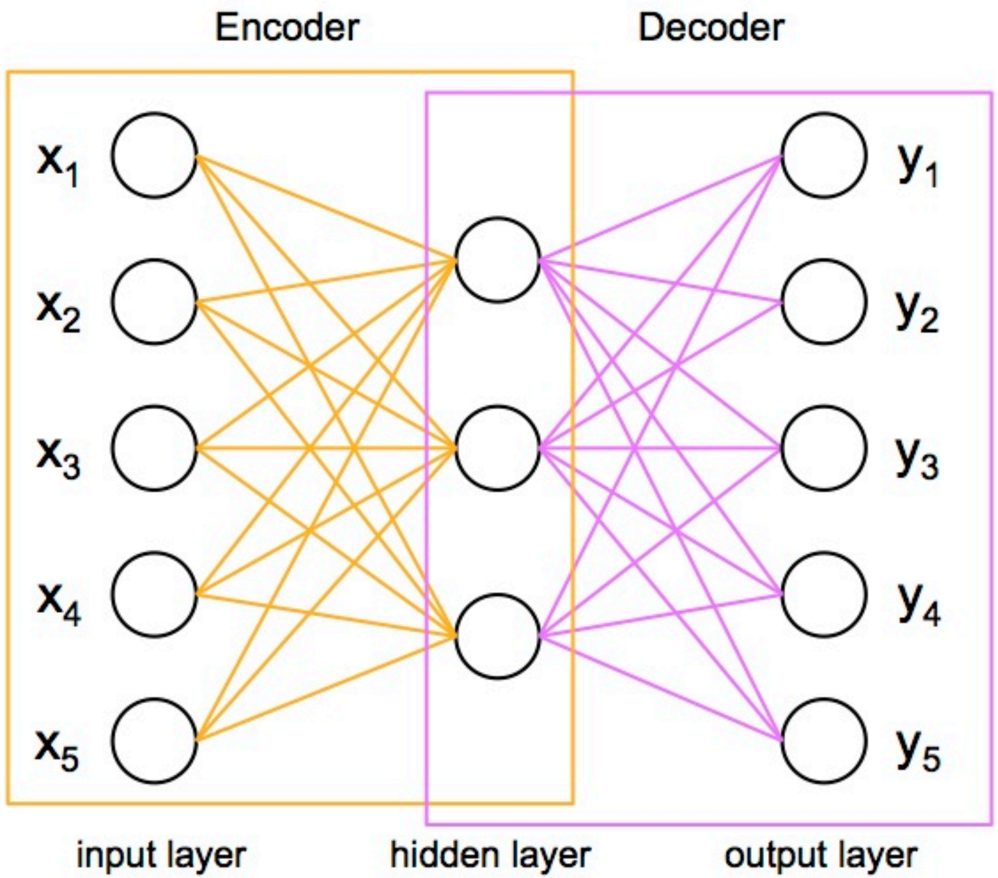
- Diabolo shape



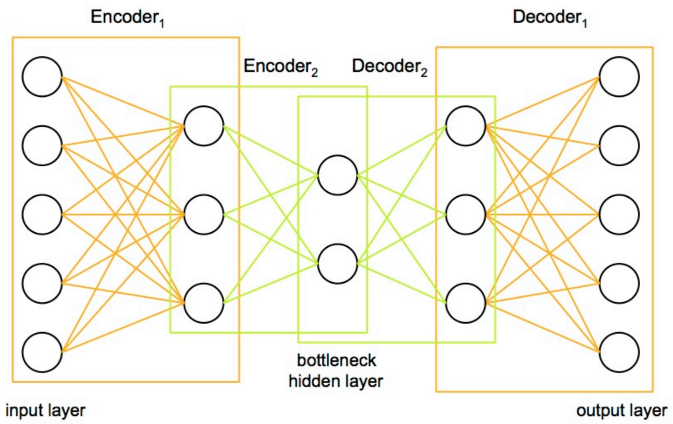
- Self-Supervised Training
  - Output = Input
  - Learning Identity
  - Learns to compress and reconstruct data
  - Extracts significant/discriminating features

Autoencoder

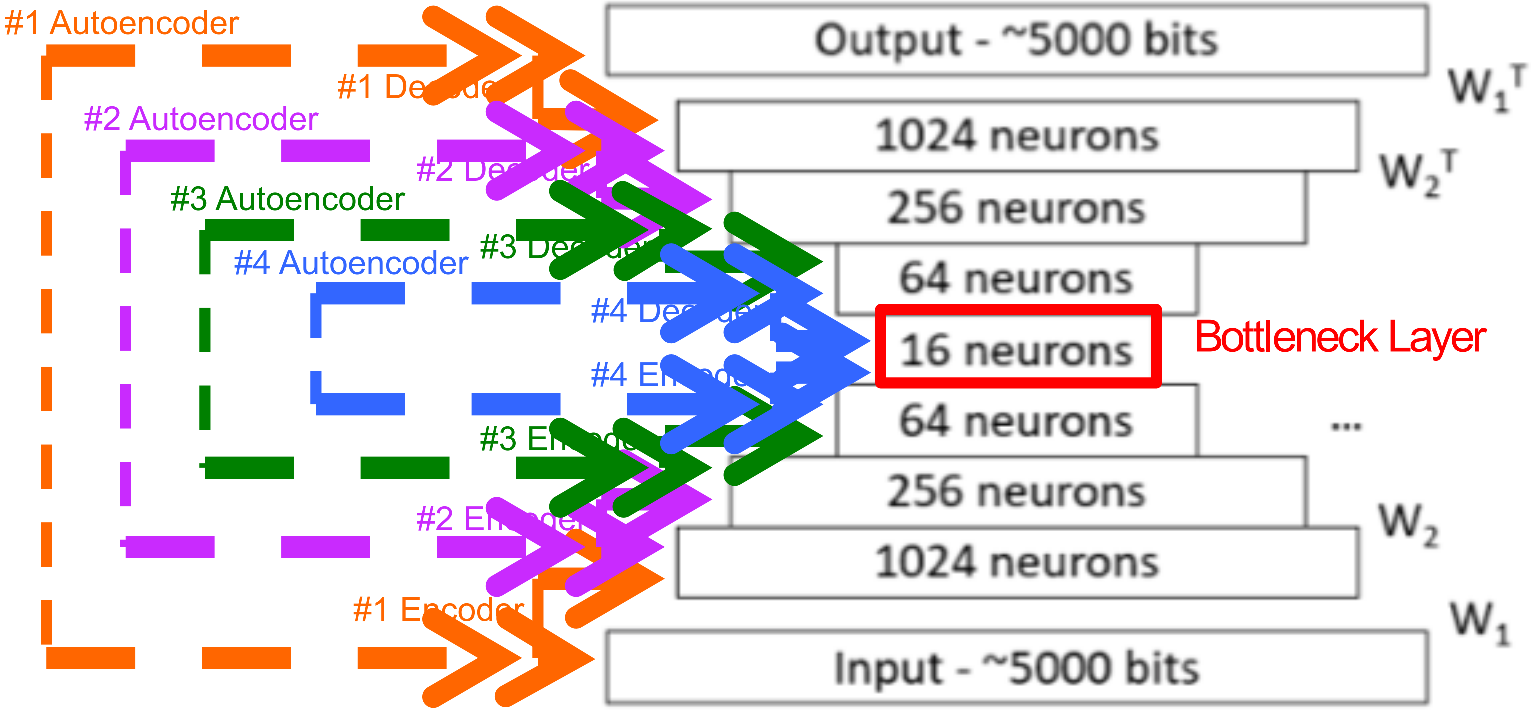
- Symmetric Neural Network
- Trained with examples as input and output
- Hidden Layer will Learn a Compressed Representation at the Hidden Layer (Latent Variables)



#1 Limitation – Generation –
#1 Partial Solution – Stacked Autoencoders

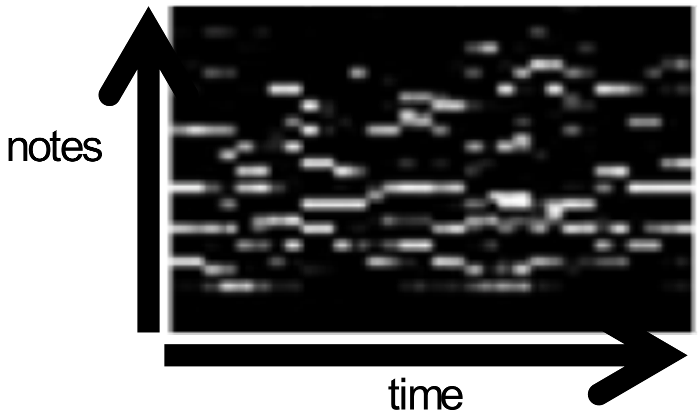


DeepHear Architecture [Sun, 2016]

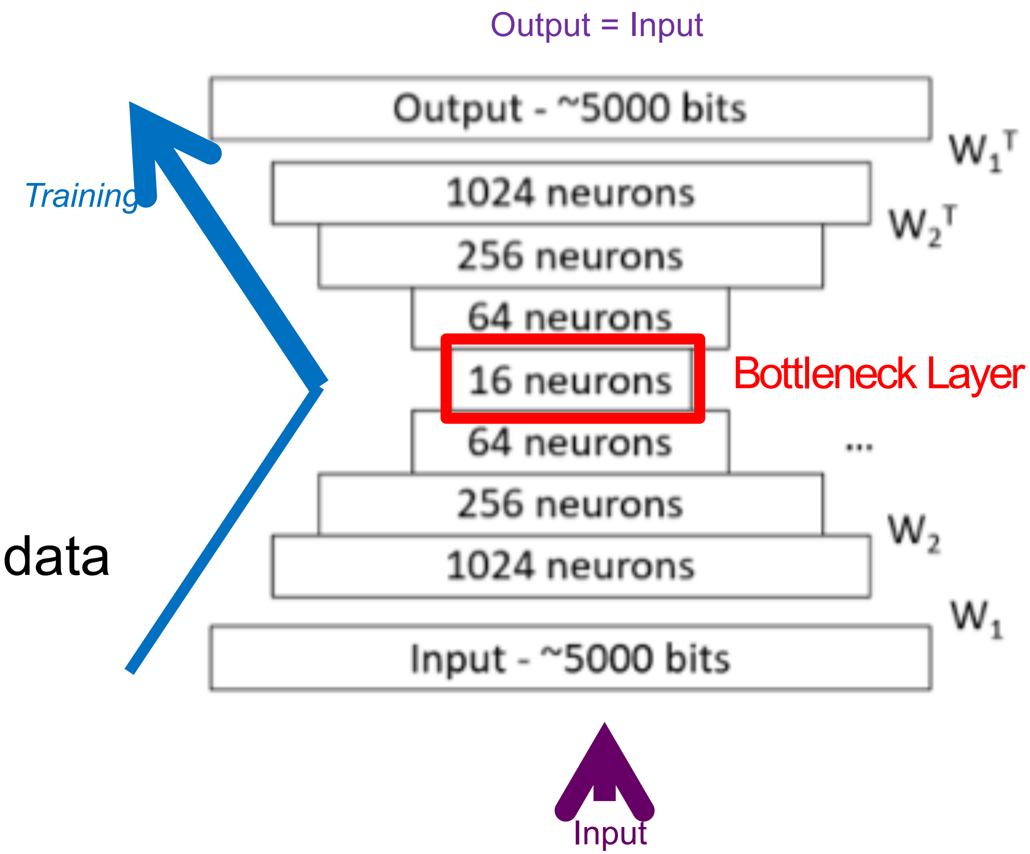


Ex1: DeepHear [Sun, 2016]

1. Pre-Training in Cascade (Layer by Layer)
   - Dataset: 600 Ragtimes (Scott Joplin)
   - Representation: Pianoroll

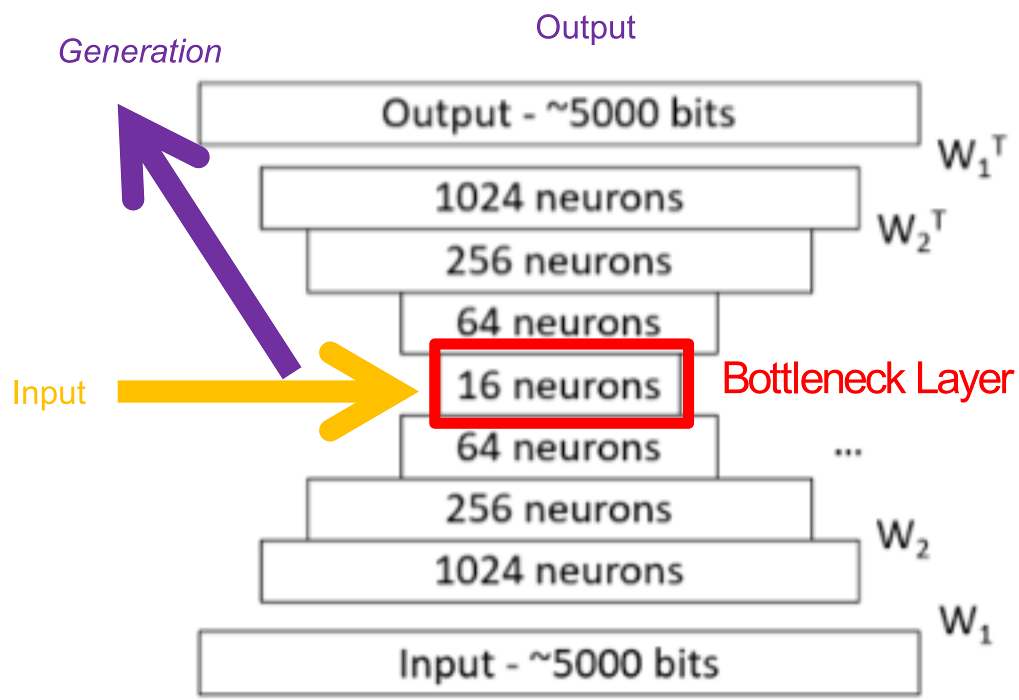


2. Self-Supervised Training
   - Learns to compress and reconstruct data
   - Extracts Features
   - Bottleneck Layer = 16 neurons



3. Generation

- Input Random Data into 16 Neurons Middle Layer
- Melody: Output of the Higher Layer Decoder

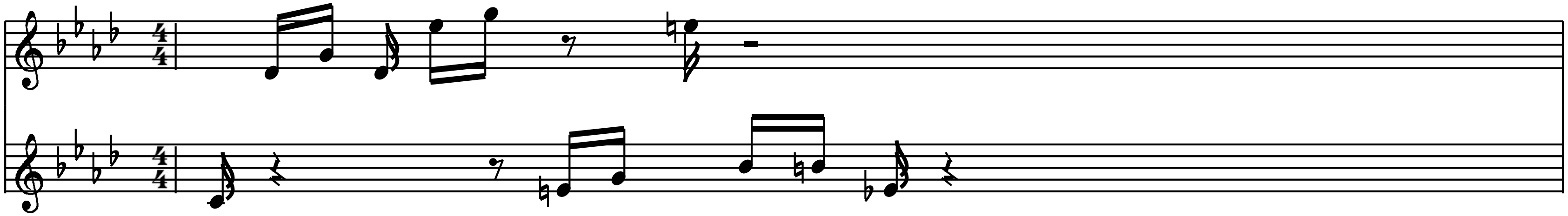


https://fephsun.github.io/2015/09/01/neural-music.html

AutoBach

random.uniform(-1, 1)

auto1

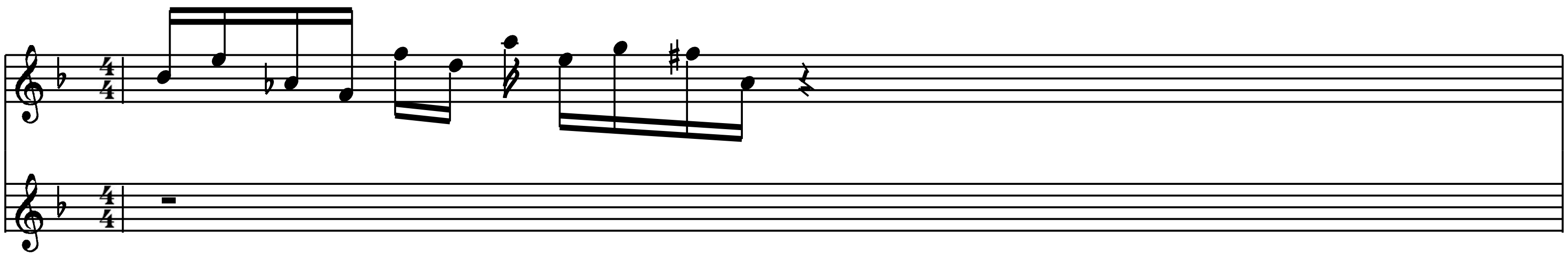


random.uniform(-10, 10)

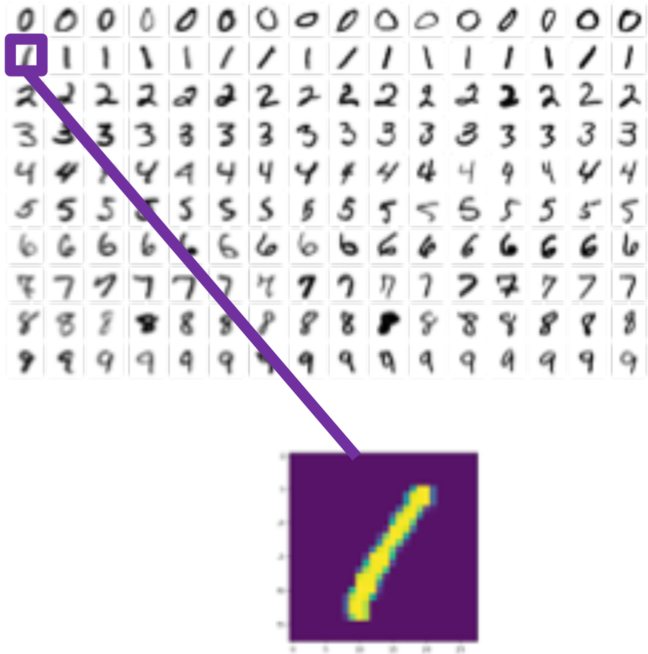
auto2

random.uniform(-1, 1)

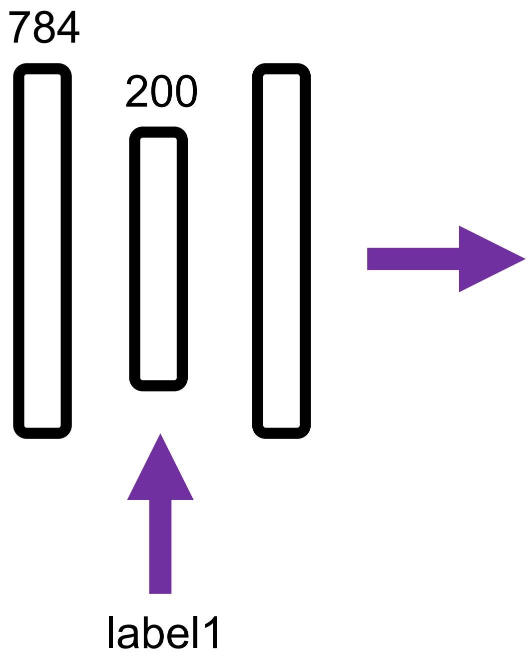
auto3



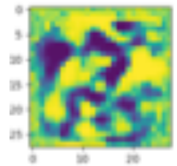
Autoencoder MNIST (Handwritten Digits)



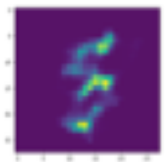
```
label1 = []
for i in range(hidden_layer_size):
            label1.append(random.uniform(-1, 1))

digit1 = decoder.predict(np.array([label1]))[0]
digit1 = digit1.reshape(28, 28)

fig, ax = plt.subplots()
im = ax.imshow(digit1)
plt.show()
```



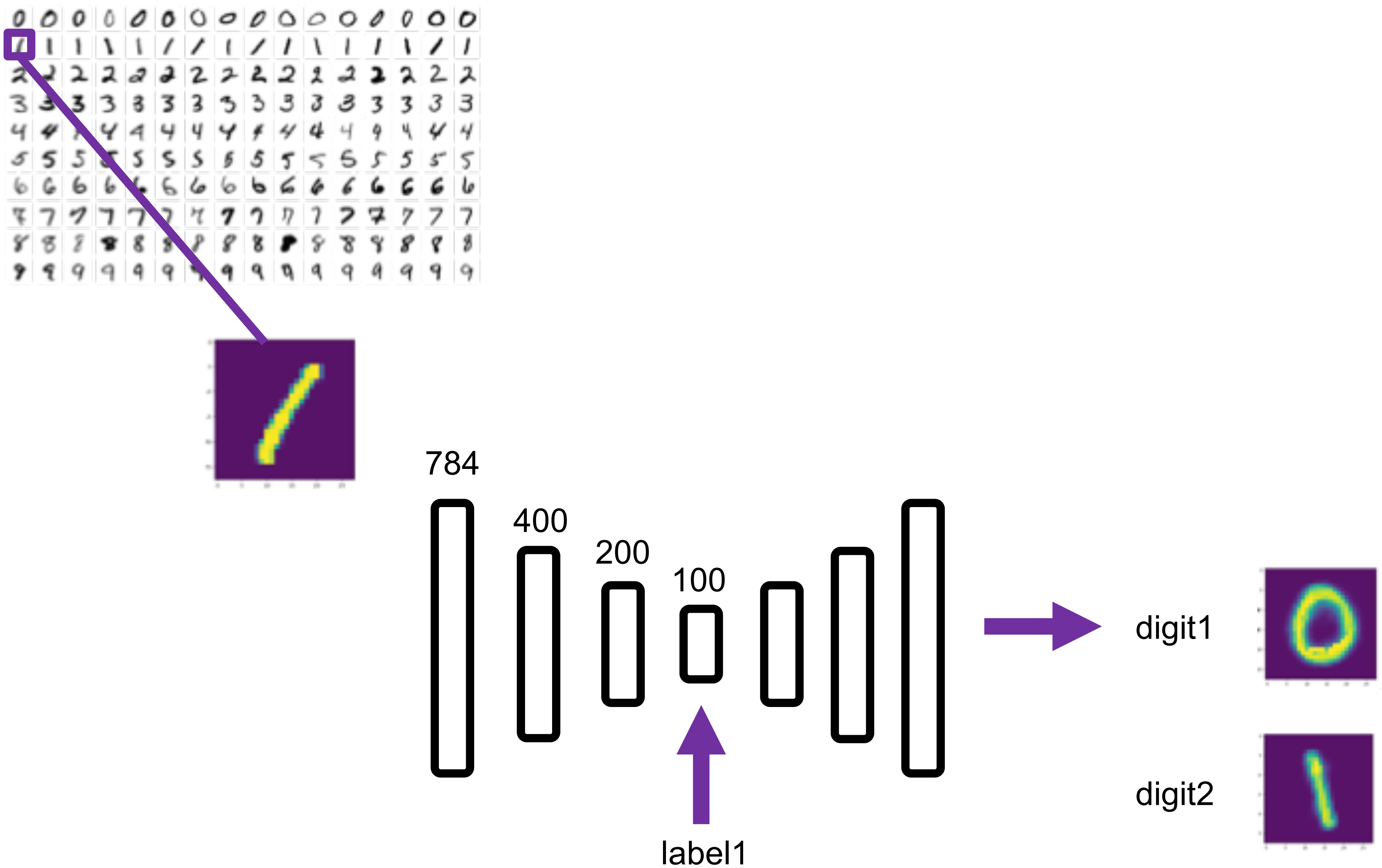
digit1



digit2



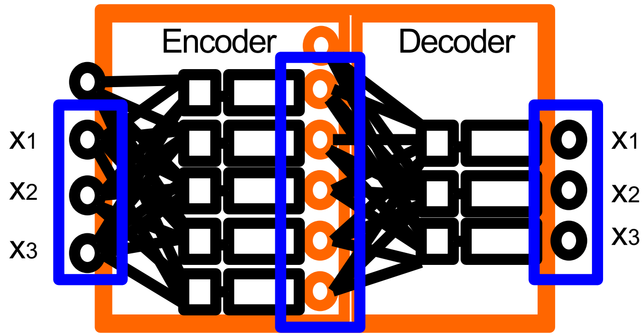
Autoencoder MNIST (Handwritten Digits)



Other Example – Creating Non-Digits

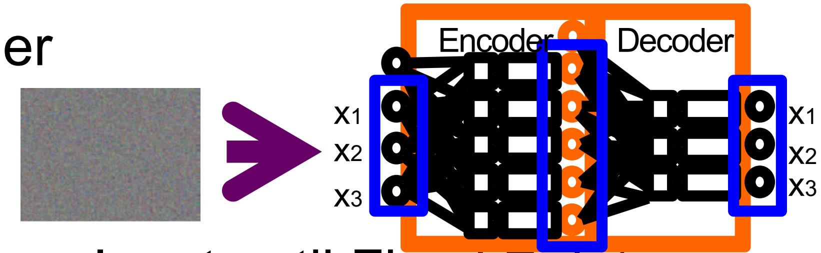
New Types Generation [Kazakçi et al., 2016]

1. Create a Sparse Autoencoder
   – Convolutional Autoencoder
     (3 Encoding Layers; 1 Decoding Layer)

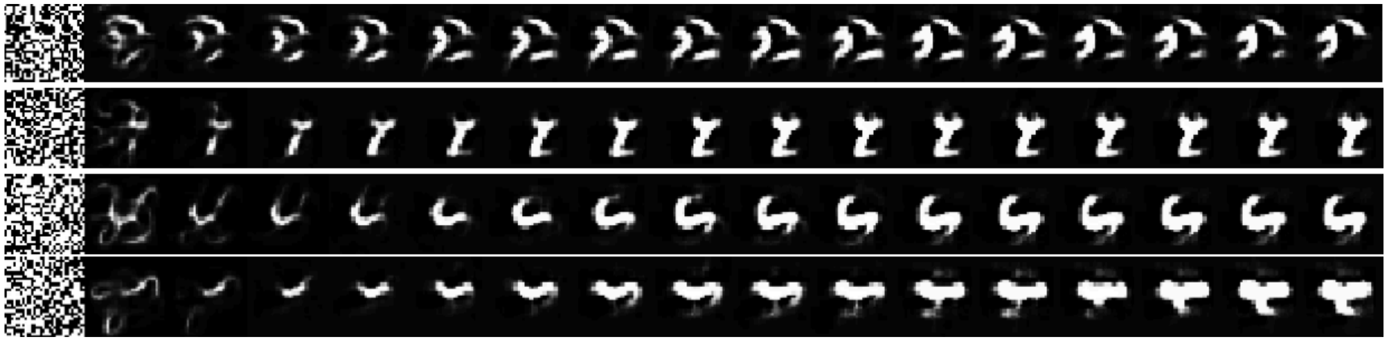


2. Train it on Objects Dataset
   – Hand Written Digits: MNIST dataset [Lecun & Cortes, 2012]

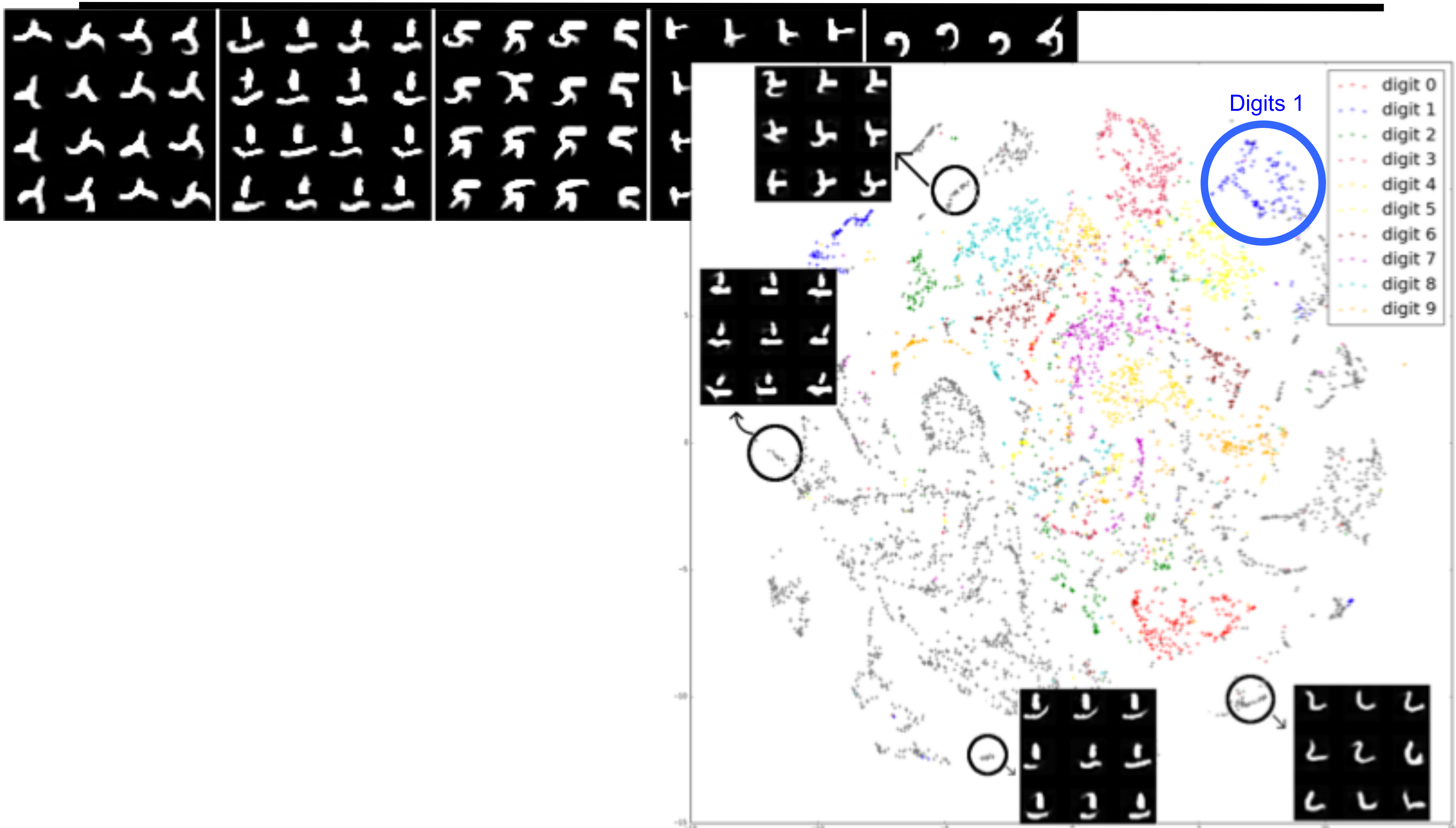
3. Feed Forward Random Image into Autoencoder
   – Output Image is Degraded but Features Emerge



4. Reiterate Feeding in Autoencoder with Output as Input until Fixed Point



New Types Generation

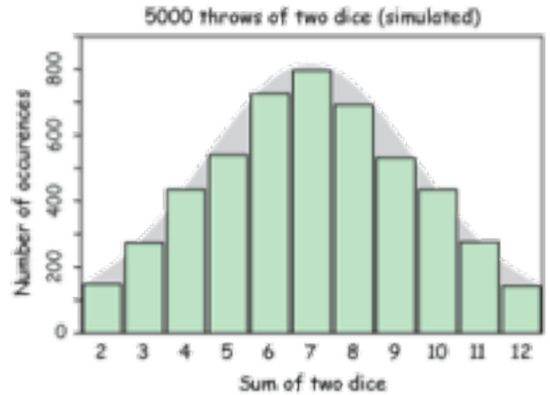


Clusters Distance

Variational Autoencoder

Variational Autoencoder (VAE) [Kingman & Welling, 2014]

- *Constraint:*
- Encoded representation (latent variables z) follow some prior probabilitydistribution $p(z)$, usually, a Gaussian distribution (normal law)

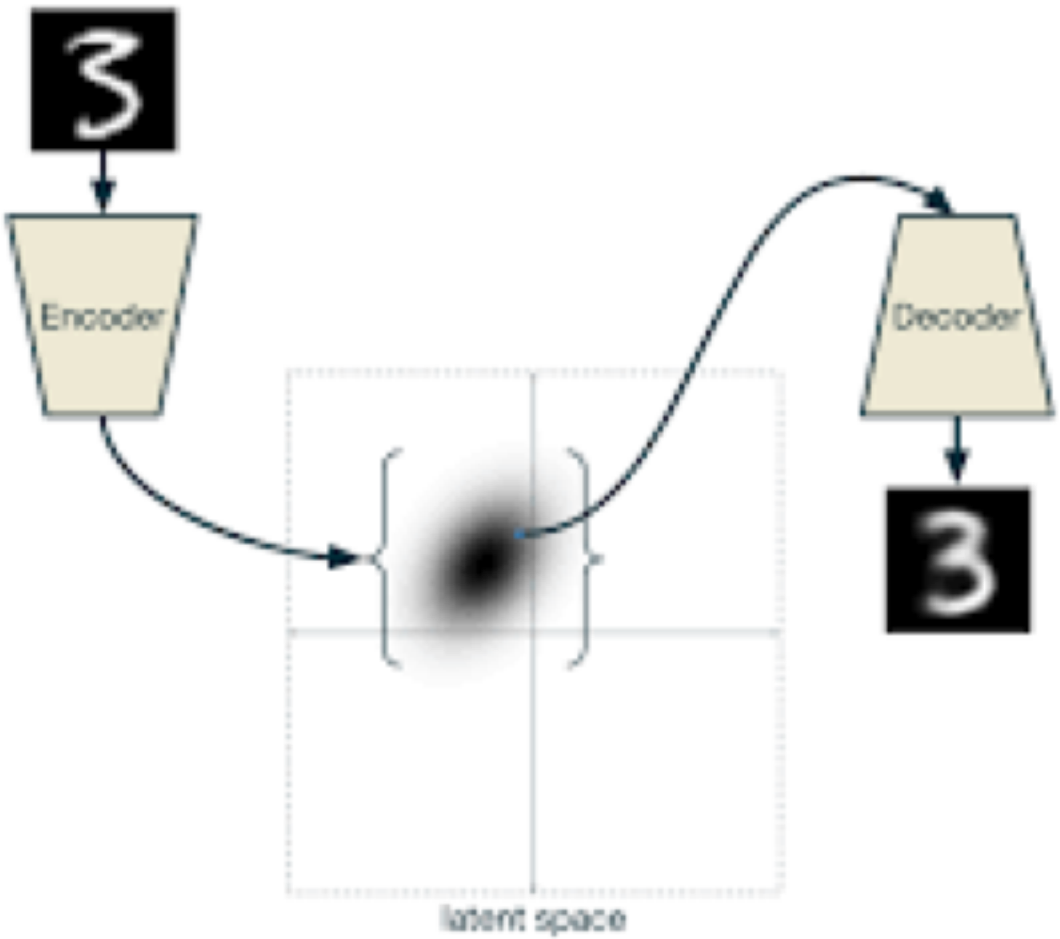


- *Non optimized implementation:*
- Adding a specific term to the cost function, by computing the cross-entropy between the values of the latent variables and the prior distribution

- The VAE decoder part will learn the relation between a Gaussian distribution of the latent variables and the learnt examples

- A VAE is able to learn a *smooth* latent space *mapping* to realistic examples

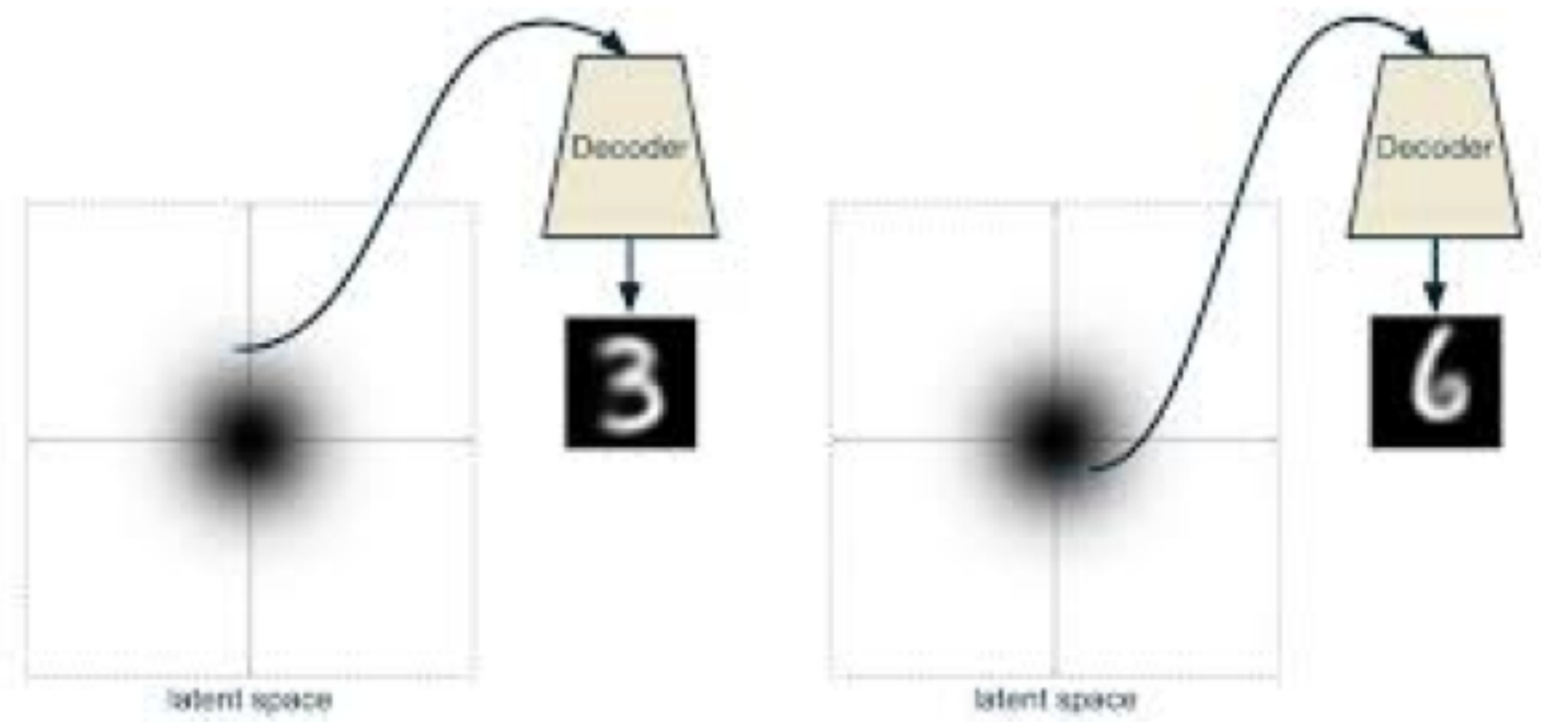
Variational Autoencoder



[Dykeman, 2016]

Variational Autoencoder

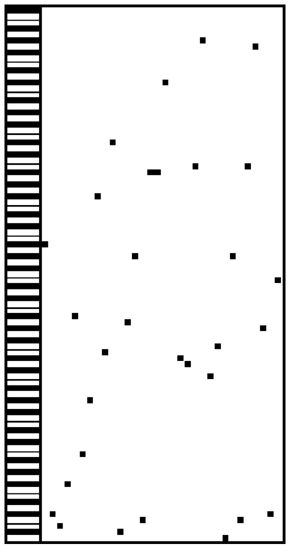
Generation
by Exploring the Latent Space
and Decoding



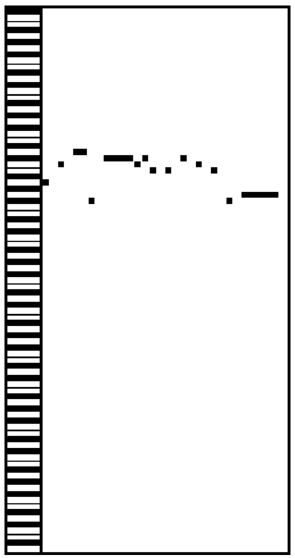
[Dykeman, 2016]

VAE vs AE Generation (MusicVAE [Roberts et al., 2018])

Non Variational



Variational



https://magenta.tensorflow.org/music-vae

Variational Generation

Exploration of the latent space with various operations to control/vary the generation of content

Ex:

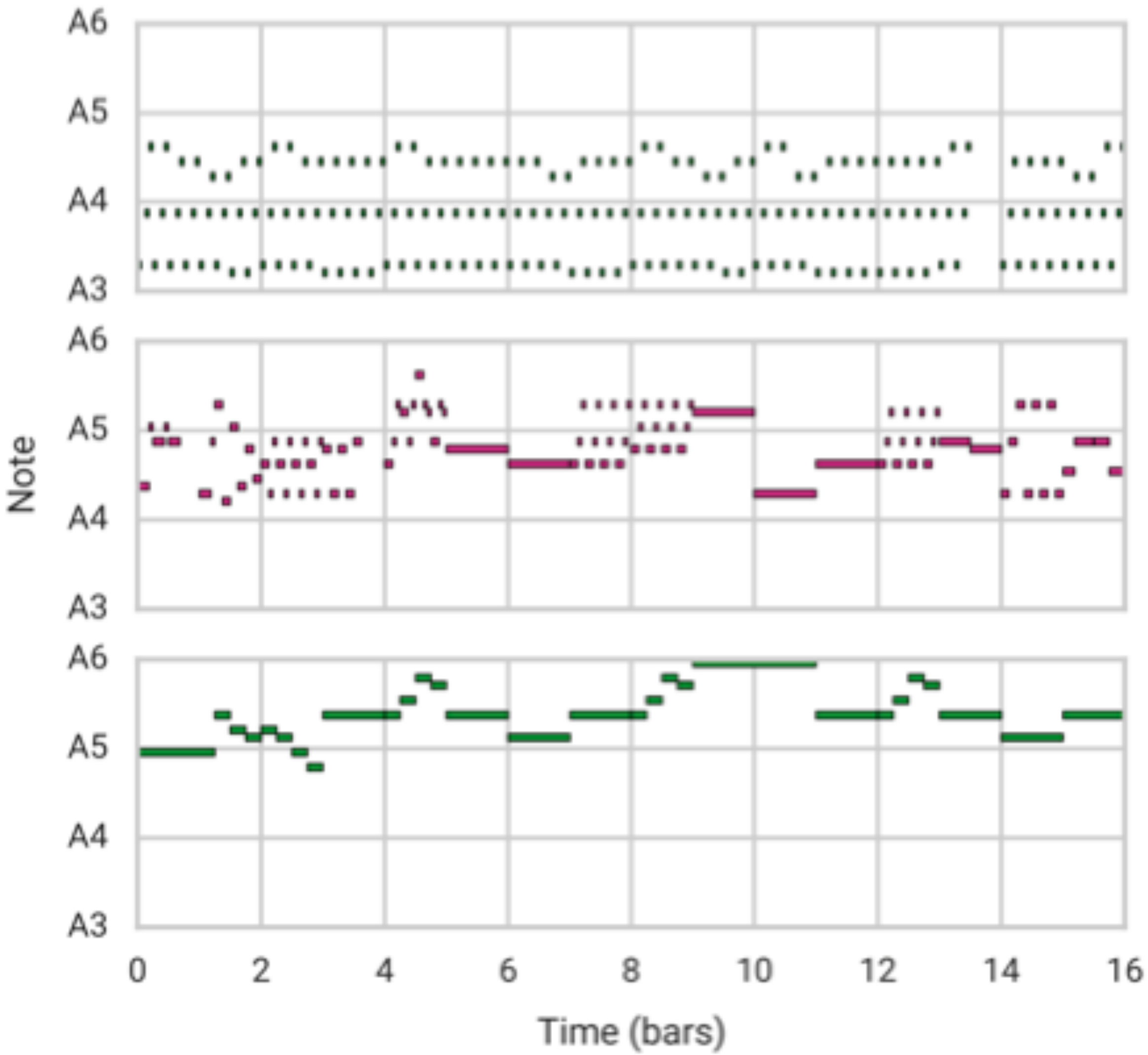
- translation
- Interpolation (morphing)
- averaging of some points
- addition or subtraction of an attribute vector capturing a given characteristic
  - This attribute vector is computed as the average latent vector for a collection of examples sharing that attribute (characteristic)

Variational Autoencoder – Ex. of Use

[Li et al., 2016]

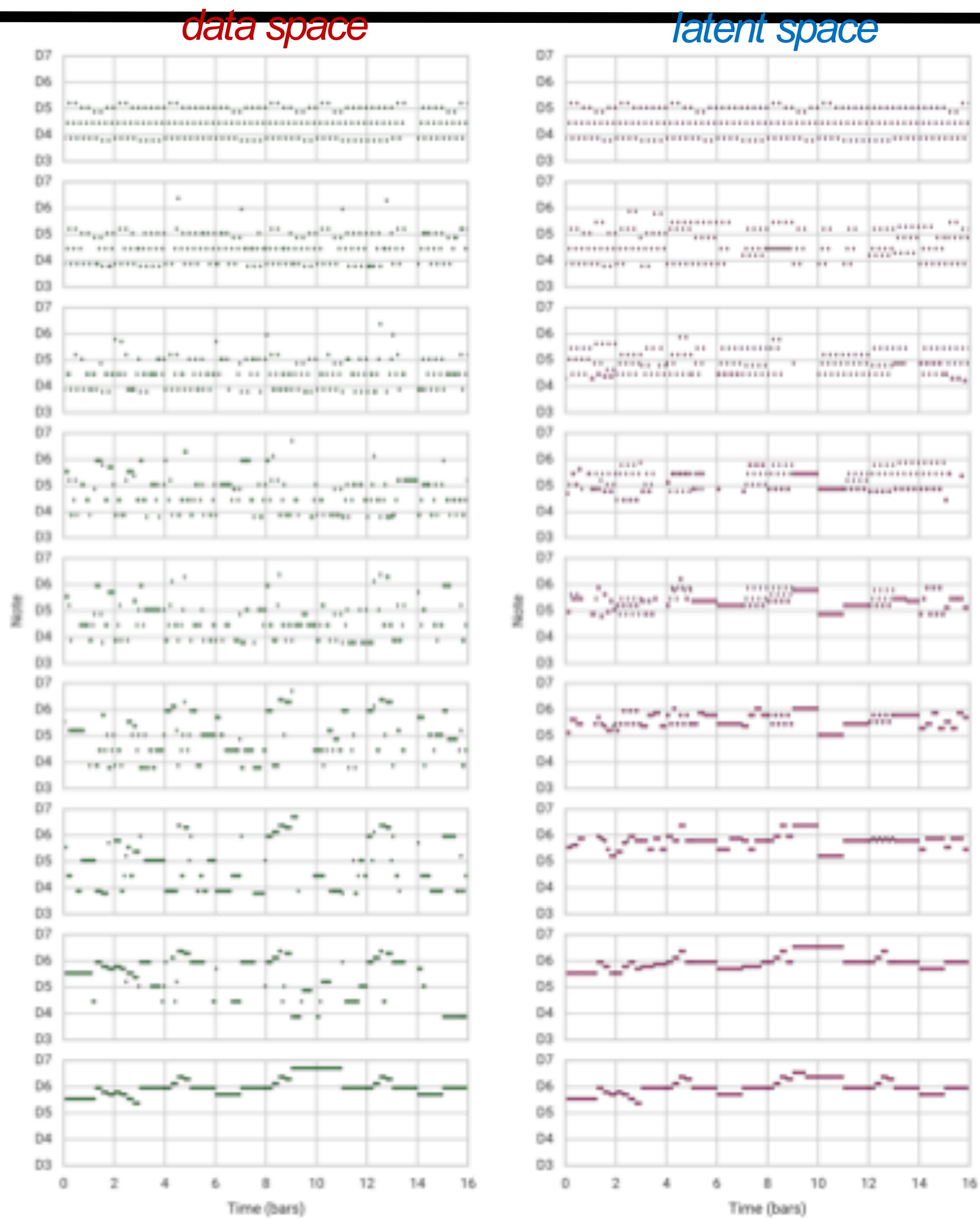
MusicVAE [Roberts et al., 2018]

- Averaging the latent space



MusicVAE [Roberts et al., 2018]

- Comparing Interpolation
  - In the data space (melodies)
  - In the latent space

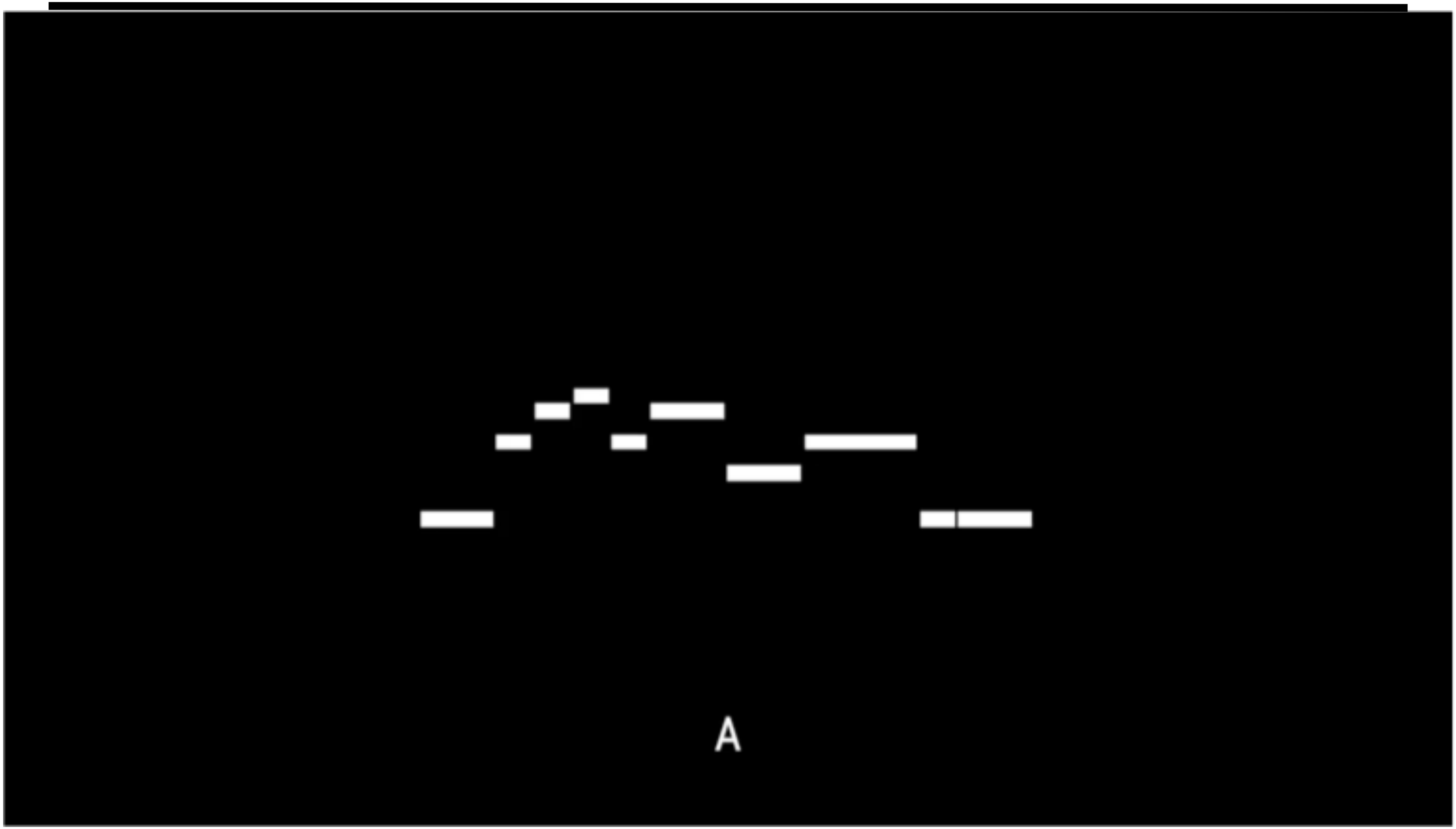


MusicVAE [Roberts et al., 2018]

- Comparing Interpolation
  - In the data space (melodies)

  - In the latent space

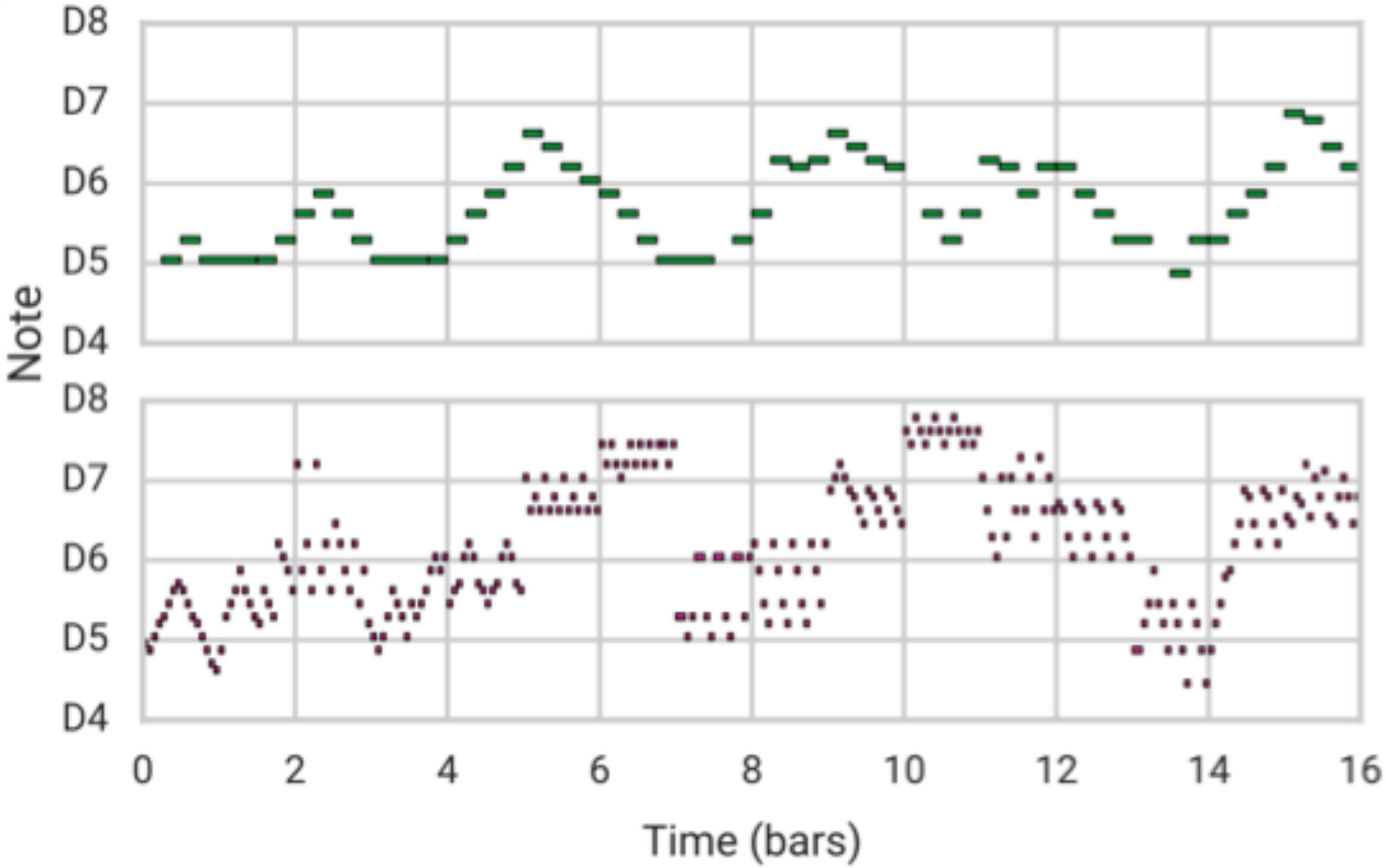
MusicVAE [Roberts et al., 2018]



https://www.youtube.com/watch?v=G5JT16flZwM

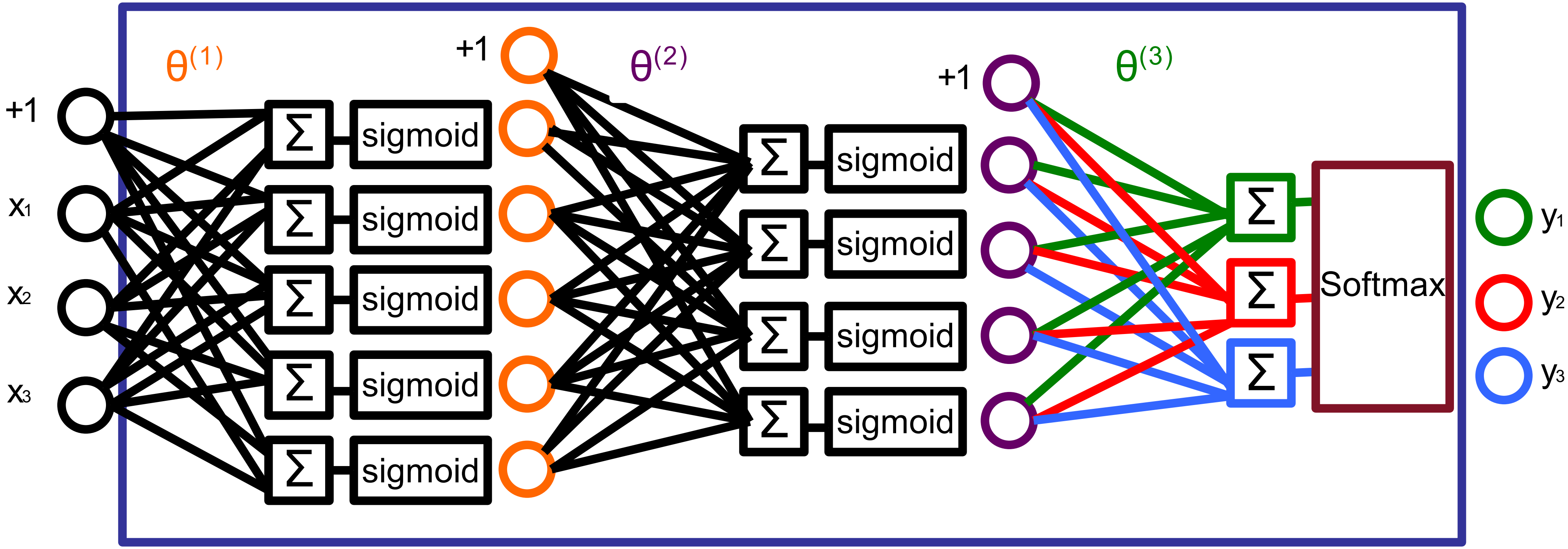
MusicVAE [Roberts et al., 2018]

- Adding a high note density attribute vector

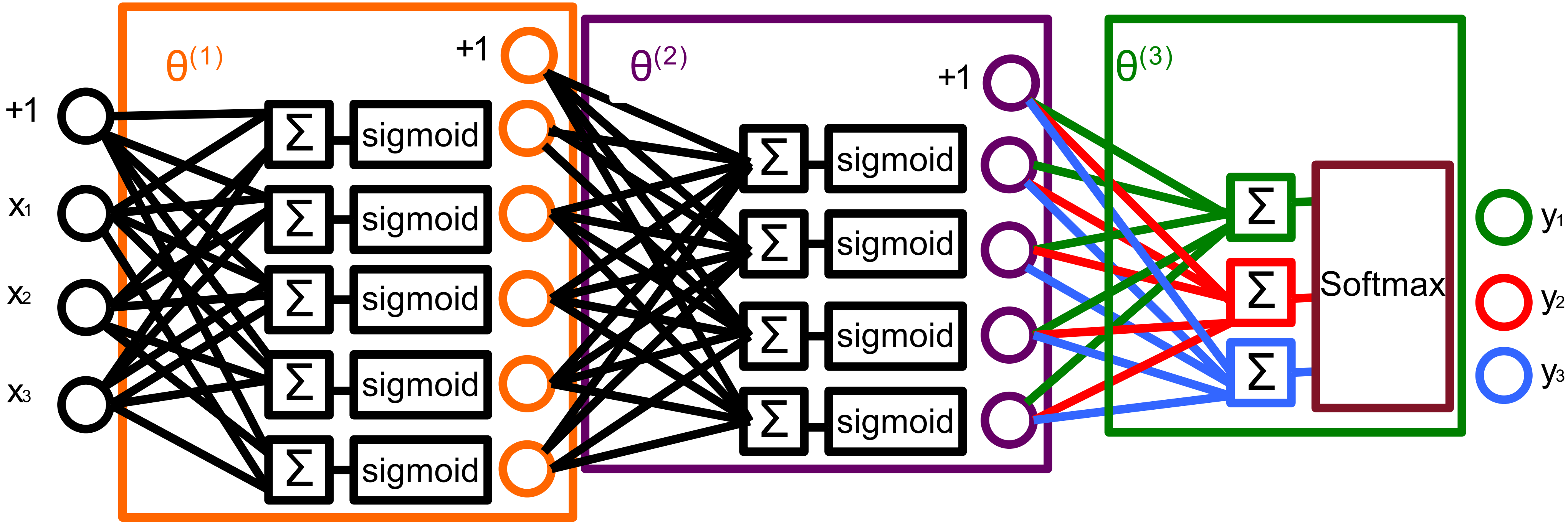


Pre-Training Deep Networks

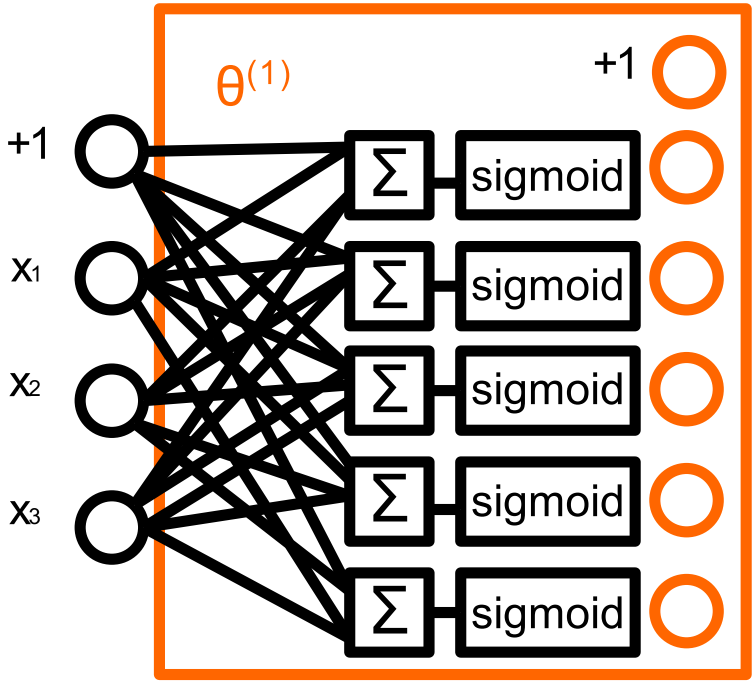
From Multilayer/Deep Networks…



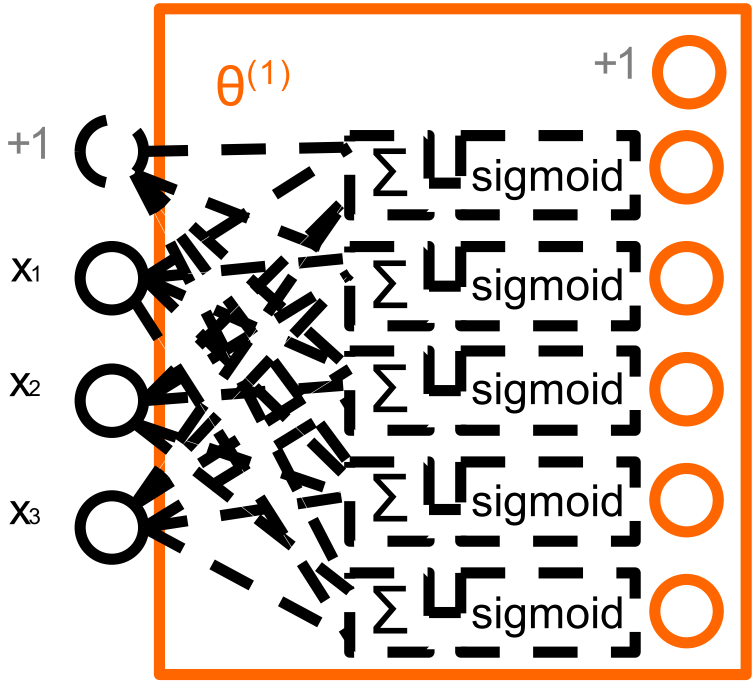
… to Pre-Trained Deep Networks [Hinton 2006]



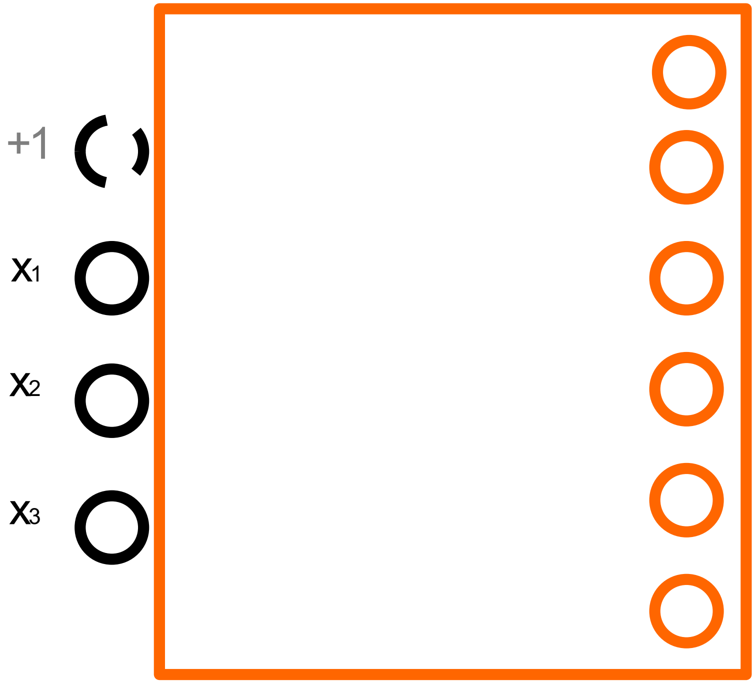
1^{st} Hidden Layer



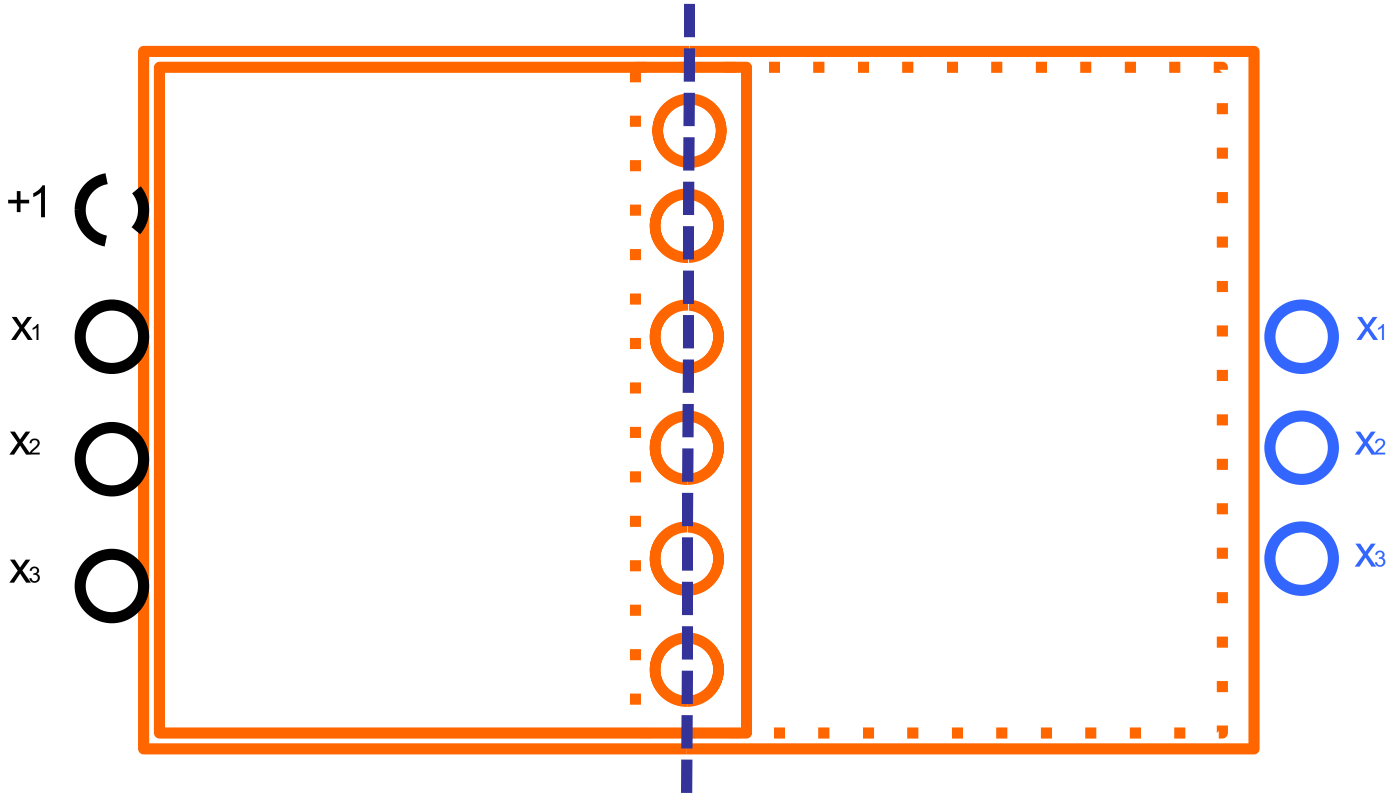
1^{st} Hidden Layer



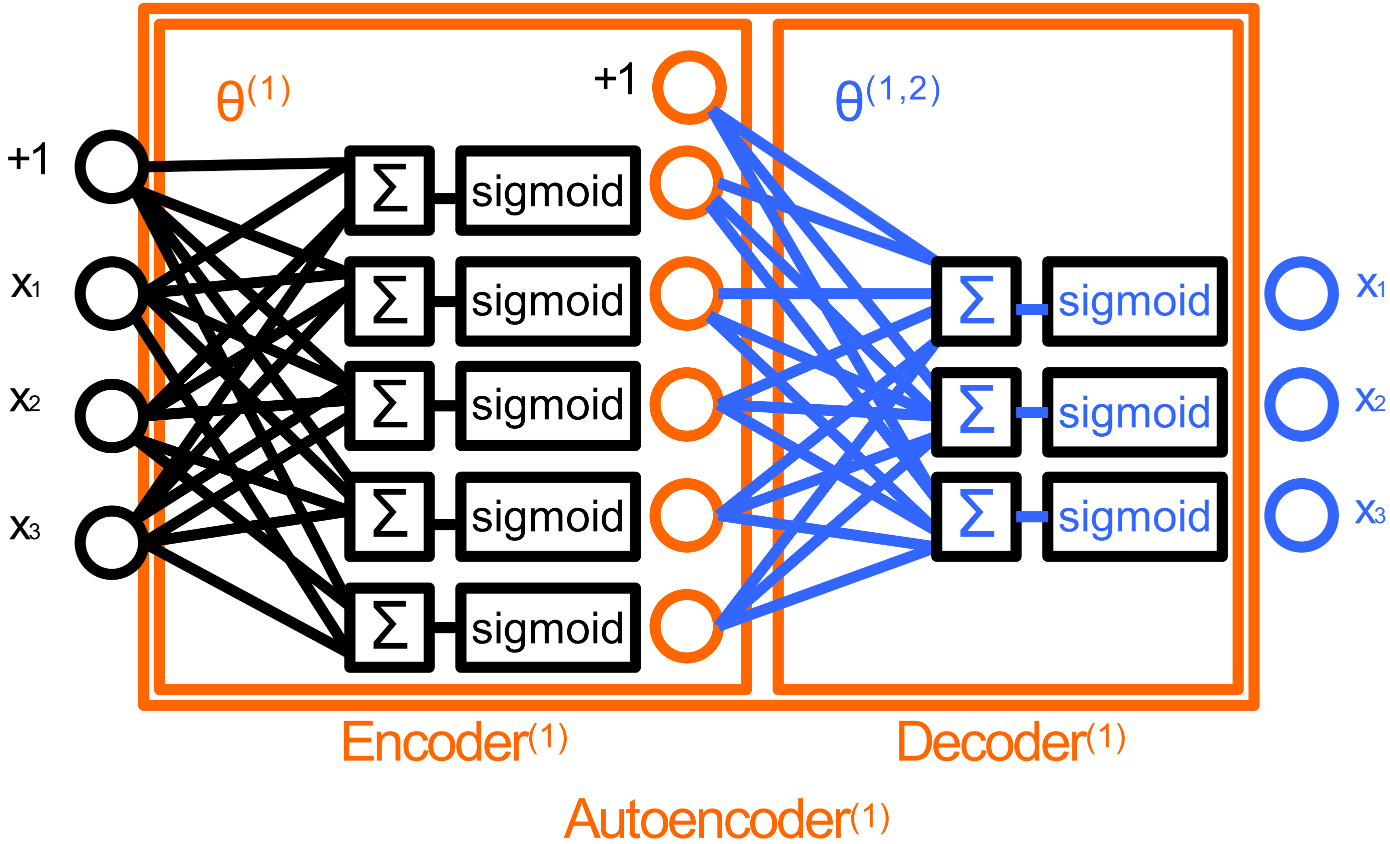
1^{st} Hidden Layer



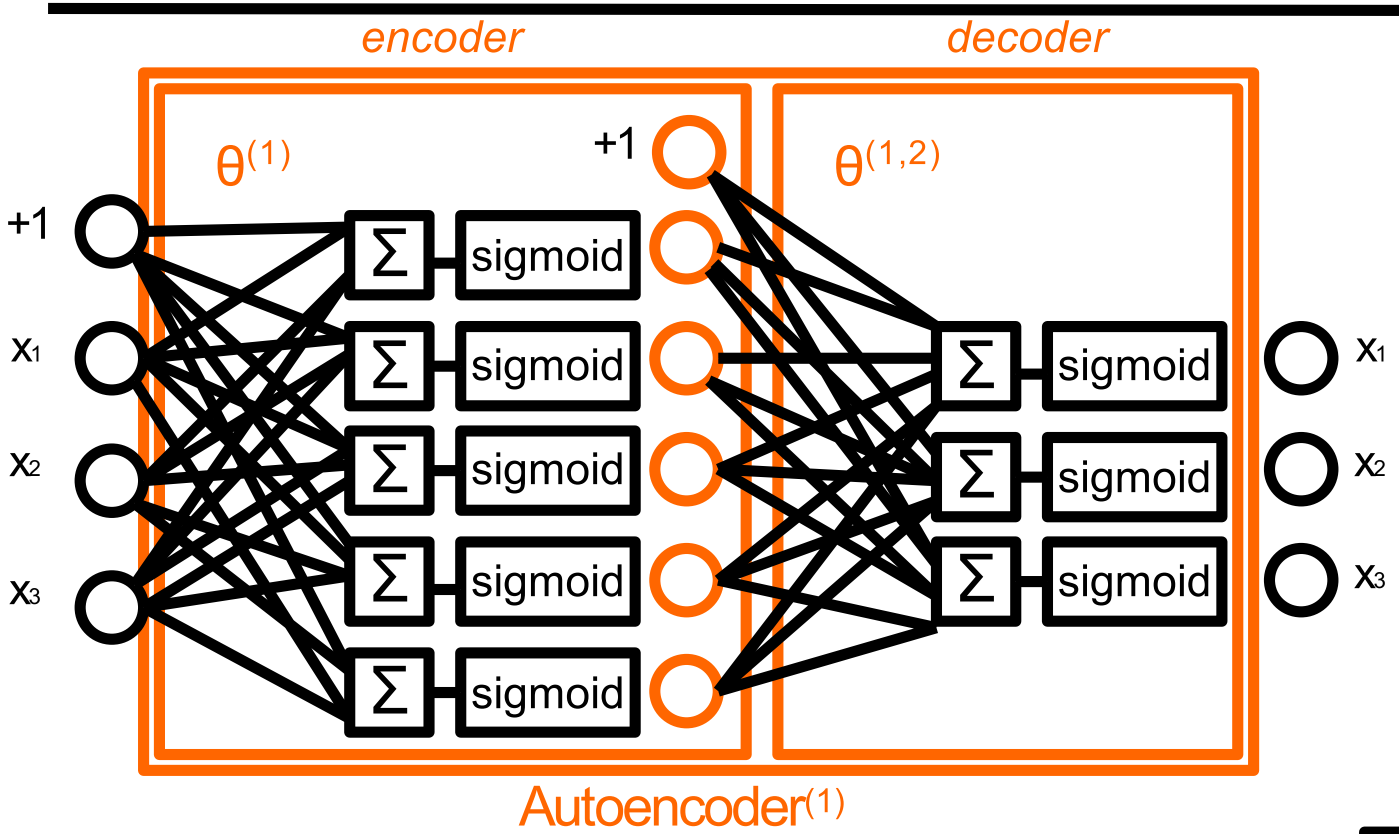
Mirroring Inputs into Outputs



Autoencoder



Autoencoder Self-Supervised Training



- Training (finding $\theta^{(1)}$ and $\theta^{(1,2)}$) on Input Dataset : X : $\begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \end{bmatrix} \ldots \begin{bmatrix} x_1^m \\ x_2^m \\ x_3^m \end{bmatrix}$
- **Self-Supervised Training implemented through Supervised Training** with Output = Input : X : **Learn Identity with Sparsity Constraint** [Ng 2012]

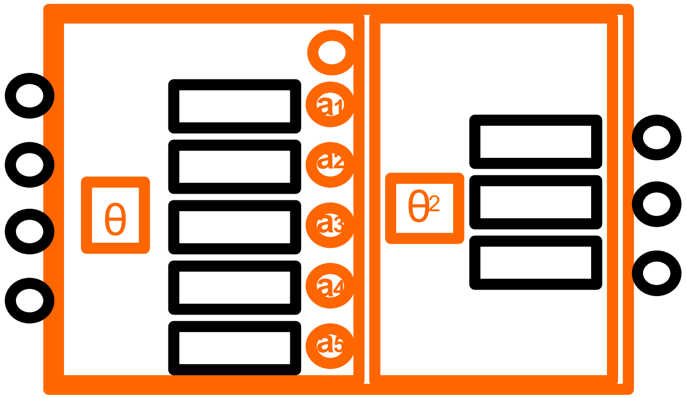
Sparse Autoencoder Learning Features

- Sparse Autoencoding [Olshausen & Field, 1996] [Ng, 2012]
- Originally developed to explain early visual processing in the brain (edge detection)
- Learns a Dictionary of bases $\Phi_1, \ldots \Phi_k$ so that each input can be approximately decomposed (recomposed) as: $x \approx {}_{j=1}{}^{k}\Sigma\, a_j\, \Phi_j$

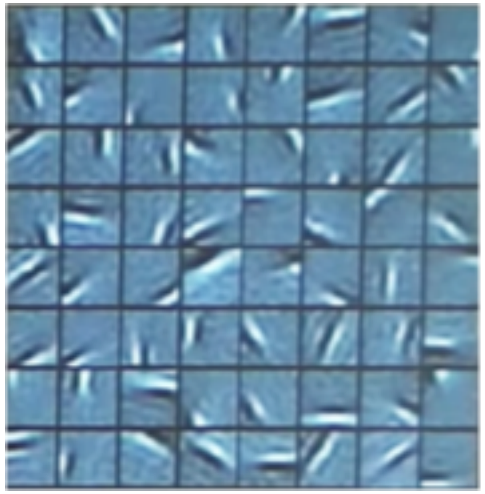
  such as a_j are mosty zero (sparsity constraint)

Images (Examples)

x_i



Autoencoder

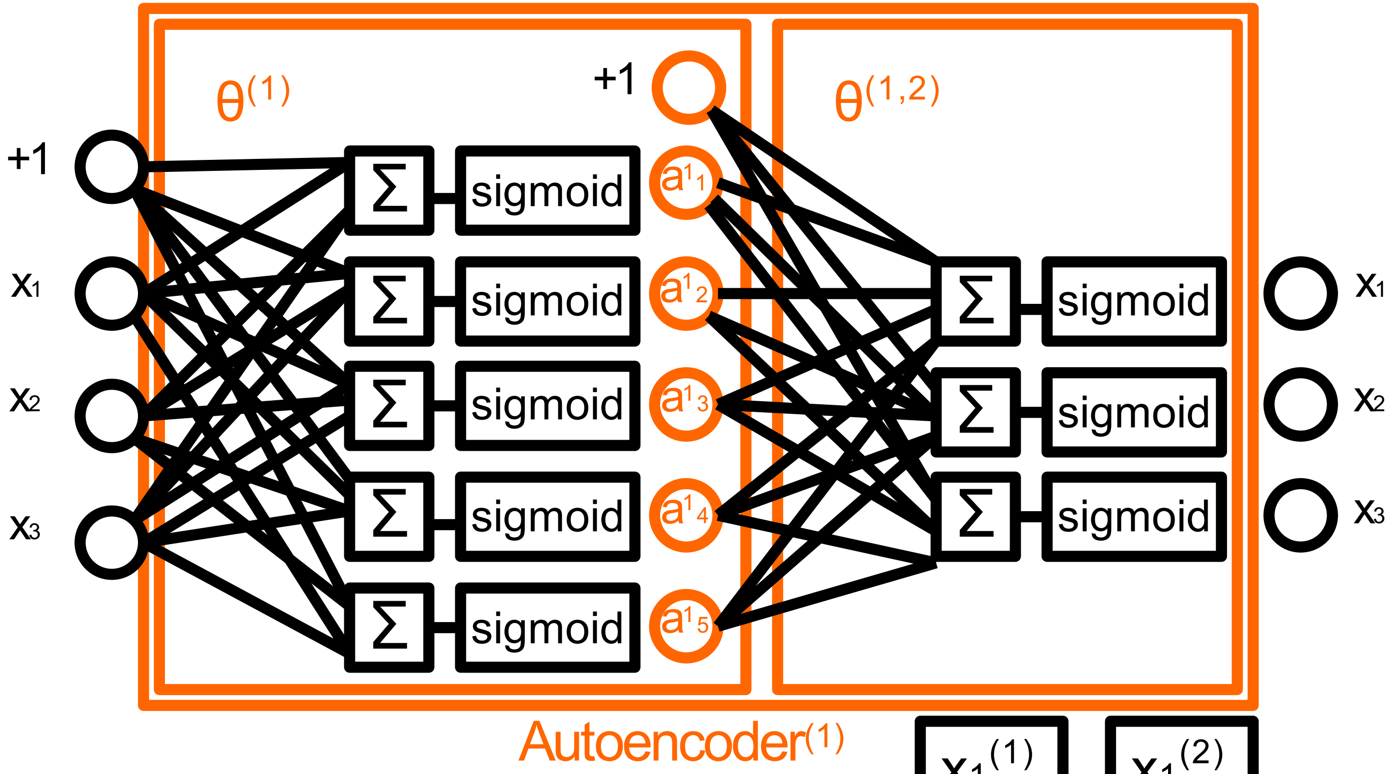


Features: $\Phi_1, \ldots \Phi_k$

[Ng, 2013]

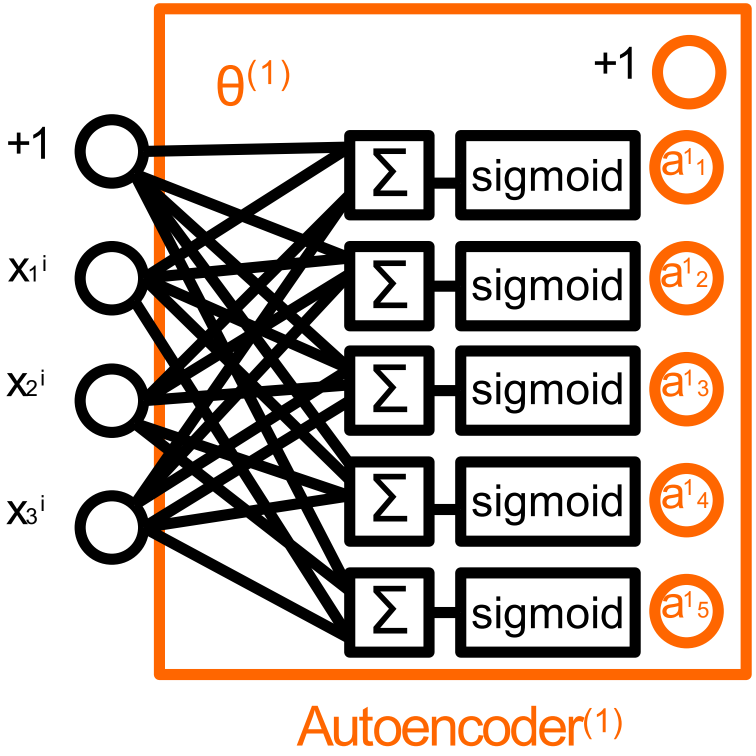
- "Invents" (learns) higher level features (e.g., edges)
- Sparsity forces specialization (feature detector) of each unit
- Alternative Non supervised learning architectures, e.g., Restricted Boltzman Machines (RBM) [Smolensky 1986] [Hinton et al. 2006]

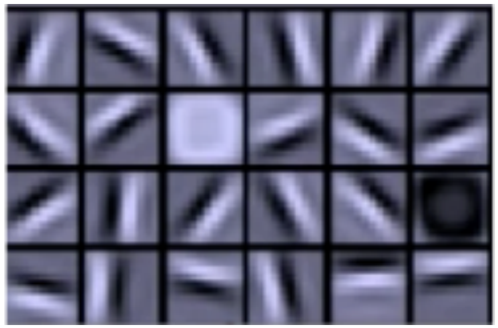
1^{st} Sparse Autoencoder Production



- Feedforward on Set of Inputs X : $\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} \ldots \begin{bmatrix} x_1^{(m)} \\ x_2^{(m)} \\ x_3^{(m)} \end{bmatrix}$
- Produces a Set of Internal States $A^1(X)$: $\begin{bmatrix} a^{(1)}{}_1{}^{(1)} \\ \ldots \\ a^{(1)}{}_5{}^{(1)} \end{bmatrix} \begin{bmatrix} a^{(1)}{}_1{}^{(2)} \\ \ldots \\ a^{(1)}{}_5{}^{(2)} \end{bmatrix} \ldots \begin{bmatrix} a^{(1)}{}_1{}^{(m)} \\ \ldots \\ a^{(1)}{}_5{}^{(m)} \end{bmatrix}$

1st Sparse Autoencoder Finalization





Learnt features(1)

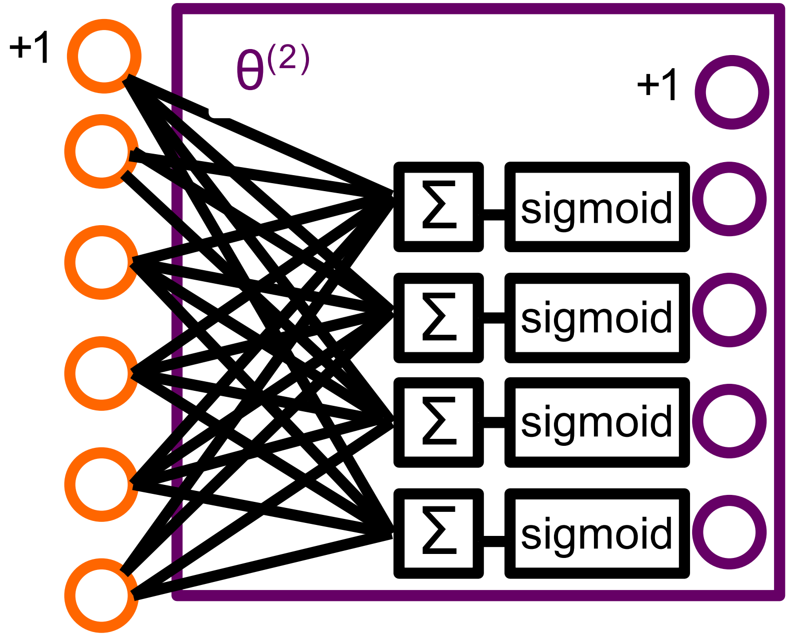
Note:

- Outputs and weights matrix $\theta^{(1,2)}$ *(decoder)* are discarded – They will not be used

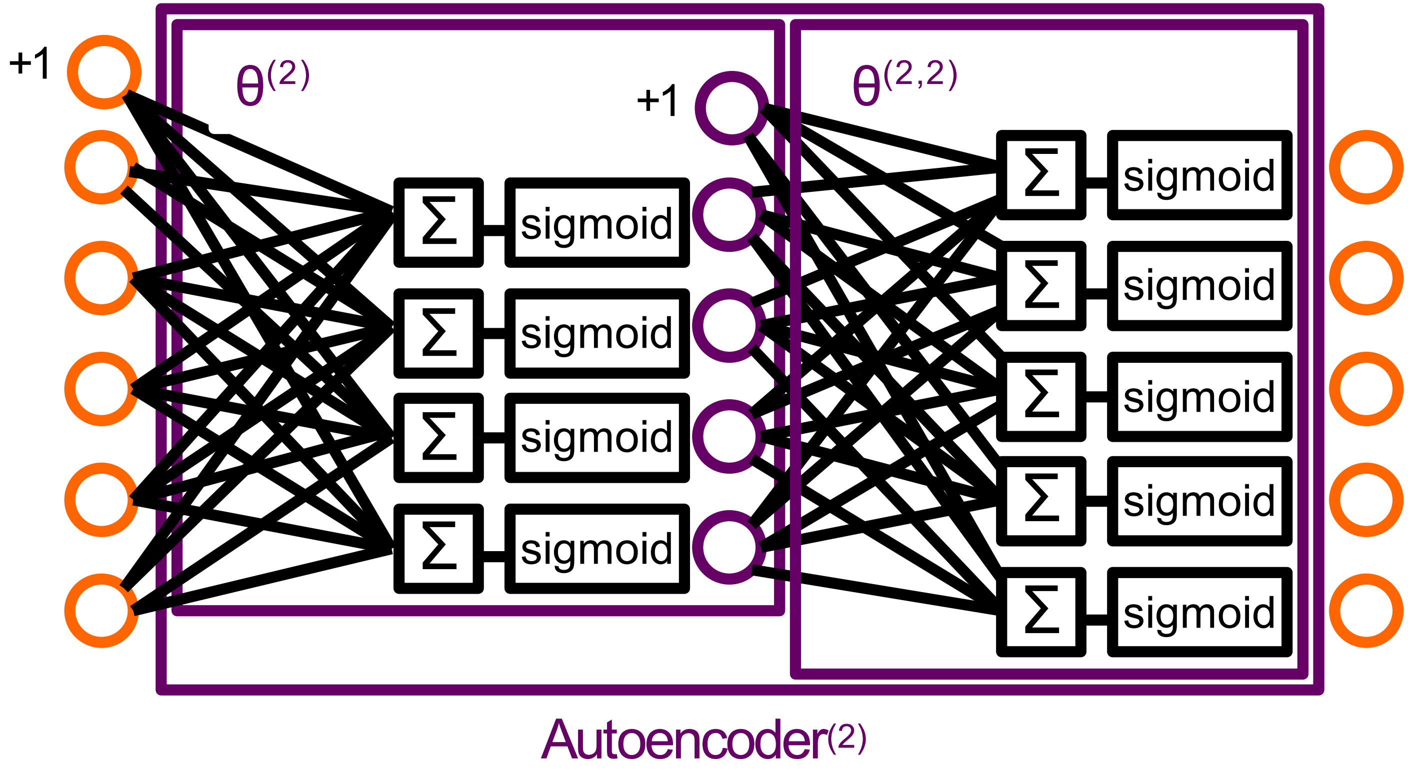
- **The weights matrix $\theta^{(1)}$ is saved for the final stage (final global fine tuning – see later)**
- **It provides accurate initialization of this layer's weight matrix**

- **The set of Internal States $a^{(1)}$ is kept for the next stage**
- **It provides examples inputs for the next layer autoencoder (see next slide)**

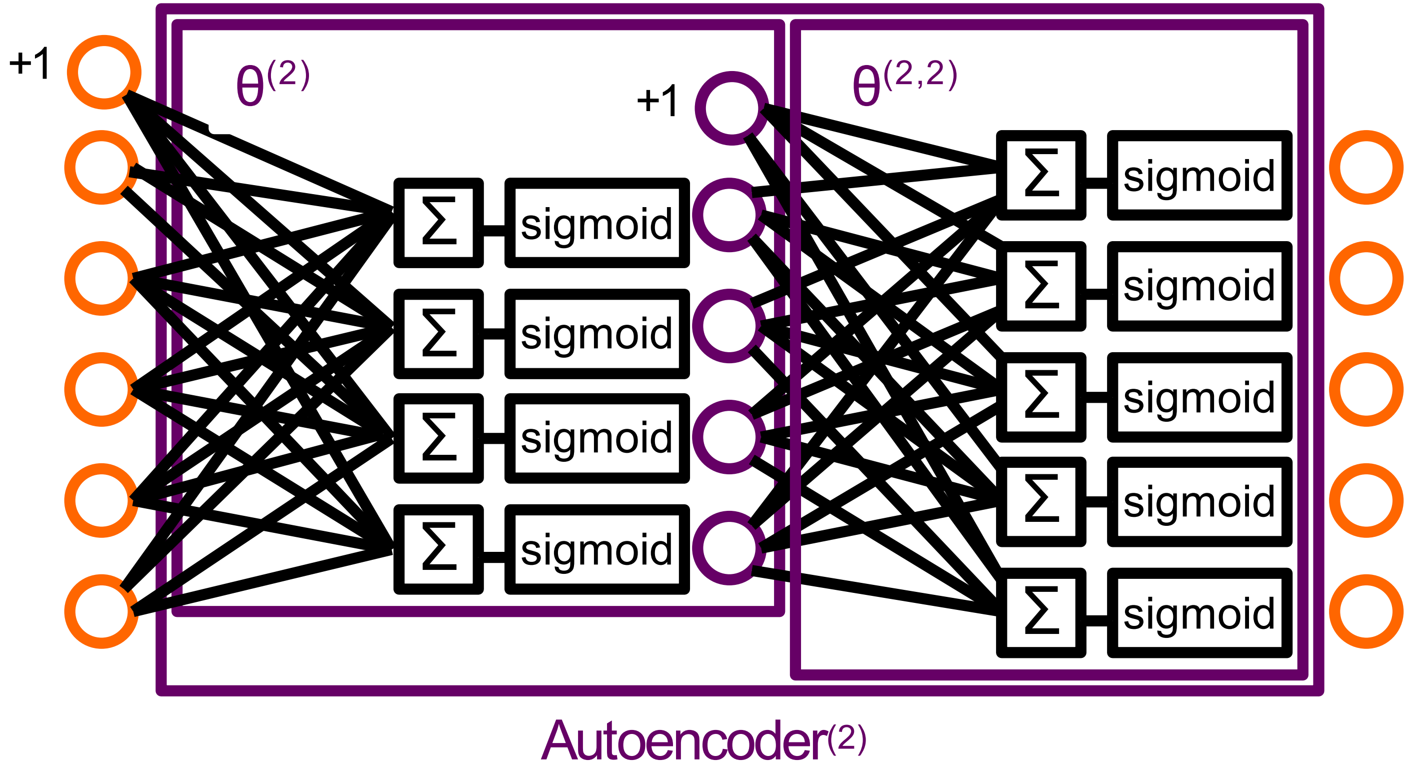
2nd Hidden Layer



2nd Sparse Autoencoder

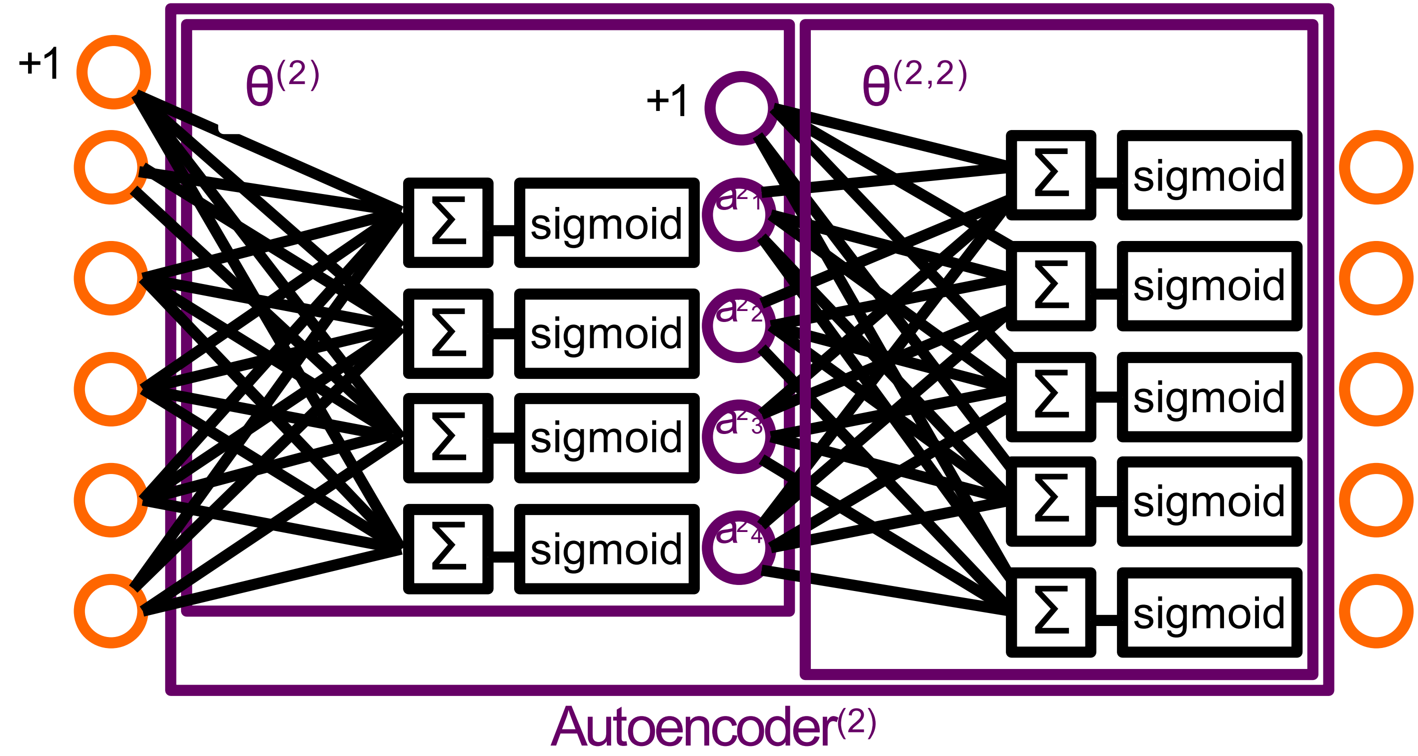


2^{nd} Sparse Autoencoder Training



- Training ($\theta^{(2)}$ and $\theta^{(2,2)}$) on Input Dataset : $A^1(X)$: $\begin{bmatrix} a^{(1)}{}_1{}^{(1)} \\ \ldots \\ a^{(1)}{}_5{}^{(1)} \end{bmatrix}$ $\begin{bmatrix} a^{(1)}{}_1{}^{(2)} \\ \ldots \\ a^{(1)}{}_5{}^{(2)} \end{bmatrix}$ … $\begin{bmatrix} a^{(1)}{}_1{}^{(m)} \\ \ldots \\ a^{(1)}{}_5{}^{(m)} \end{bmatrix}$

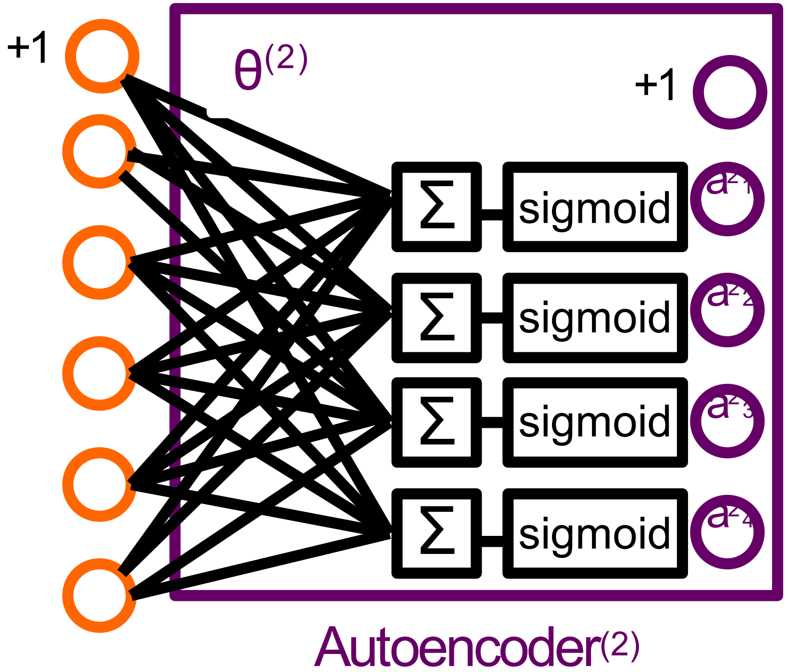
2^{nd} Sparse Autoencoder Production

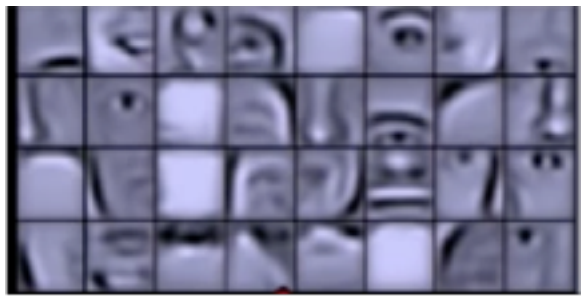


- Feedforward on Input Dataset : $A^1(X)$: $\begin{bmatrix} a^{(1)}{}_1{}^{(1)} \\ \dots \\ a^{(1)}{}_5{}^{(1)} \end{bmatrix}$ $\begin{bmatrix} a^{(1)}{}_1{}^{(2)} \\ \dots \\ a^{(1)}{}_5{}^{(2)} \end{bmatrix}$ … $\begin{bmatrix} a^{(1)}{}_1{}^{(m)} \\ \dots \\ a^{(1)}{}_5{}^{(m)} \end{bmatrix}$

- Produces a Set of Internal States $A^2(A^1(X))$: $\begin{bmatrix} a^{(2)}{}_1{}^{(1)} \\ \dots \\ a^{(2)}{}_4{}^{(1)} \end{bmatrix}$ $\begin{bmatrix} a^{(2)}{}_1{}^{(2)} \\ \dots \\ a^{(2)}{}_4{}^{(2)} \end{bmatrix}$ … $\begin{bmatrix} a^{(2)}{}_1{}^{(m)} \\ \dots \\ a^{(2)}{}_4{}^{(m)} \end{bmatrix}$

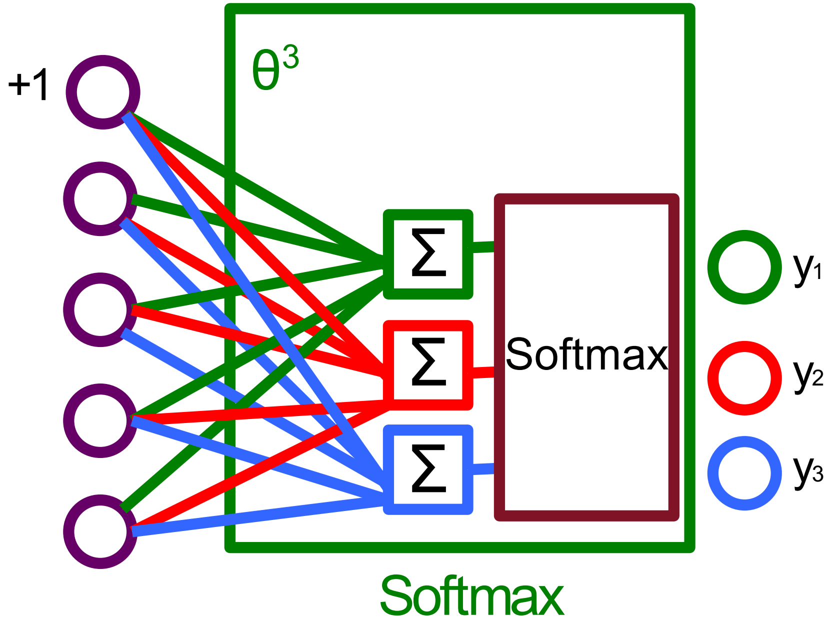
2^{nd} Sparse Autoencoder Finalization





Learnt features(2)

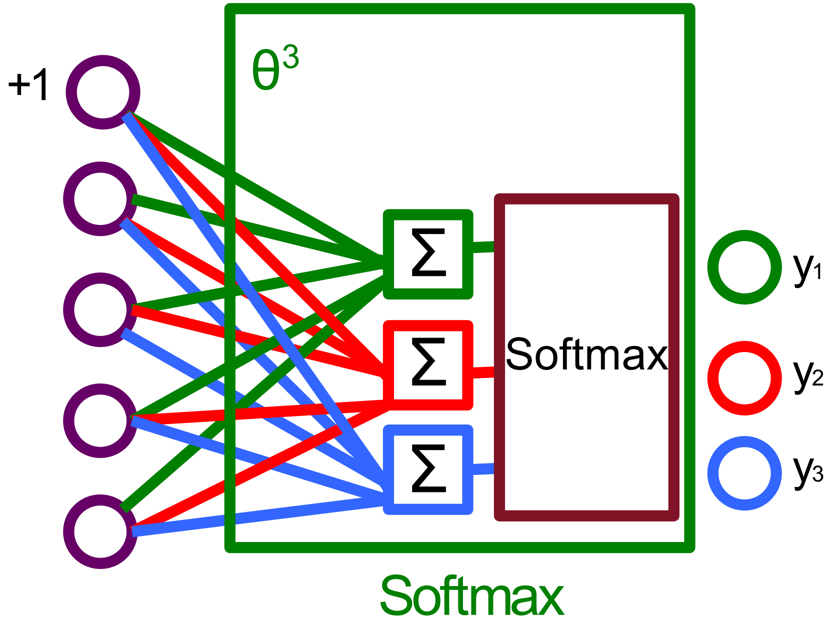
(Final) Softmax Layer



- (Supervised) Training on Input Dataset : $A^2(A^1(X))$: $\begin{bmatrix} a^{(2)}{}_1{}^{(1)} \\ \dots \\ a^{(2)}{}_4{}^{(1)} \end{bmatrix} \begin{bmatrix} a^{(2)}{}_1{}^{(2)} \\ \dots \\ a^{(2)}{}_4{}^{(2)} \end{bmatrix} \dots \begin{bmatrix} a^{(2)}{}_1{}^{(m)} \\ \dots \\ a^{(2)}{}_4{}^{(m)} \end{bmatrix}$

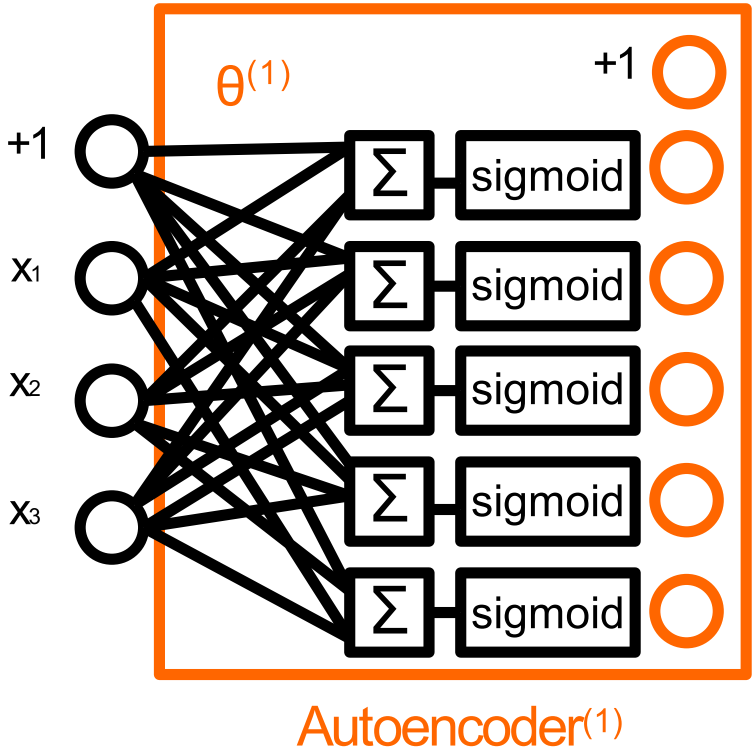
  and Output Dataset : y : $\begin{bmatrix} y_1{}^{(1)} \\ y_2{}^{(1)} \\ y_3{}^{(1)} \end{bmatrix} \begin{bmatrix} y_1{}^{(2)} \\ y_2{}^{(2)} \\ y_3{}^{(2)} \end{bmatrix} \dots \begin{bmatrix} y_1{}^{(m)} \\ y_2{}^{(m)} \\ y_3{}^{(m)} \end{bmatrix}$

(Final) Softmax Layer Finalization

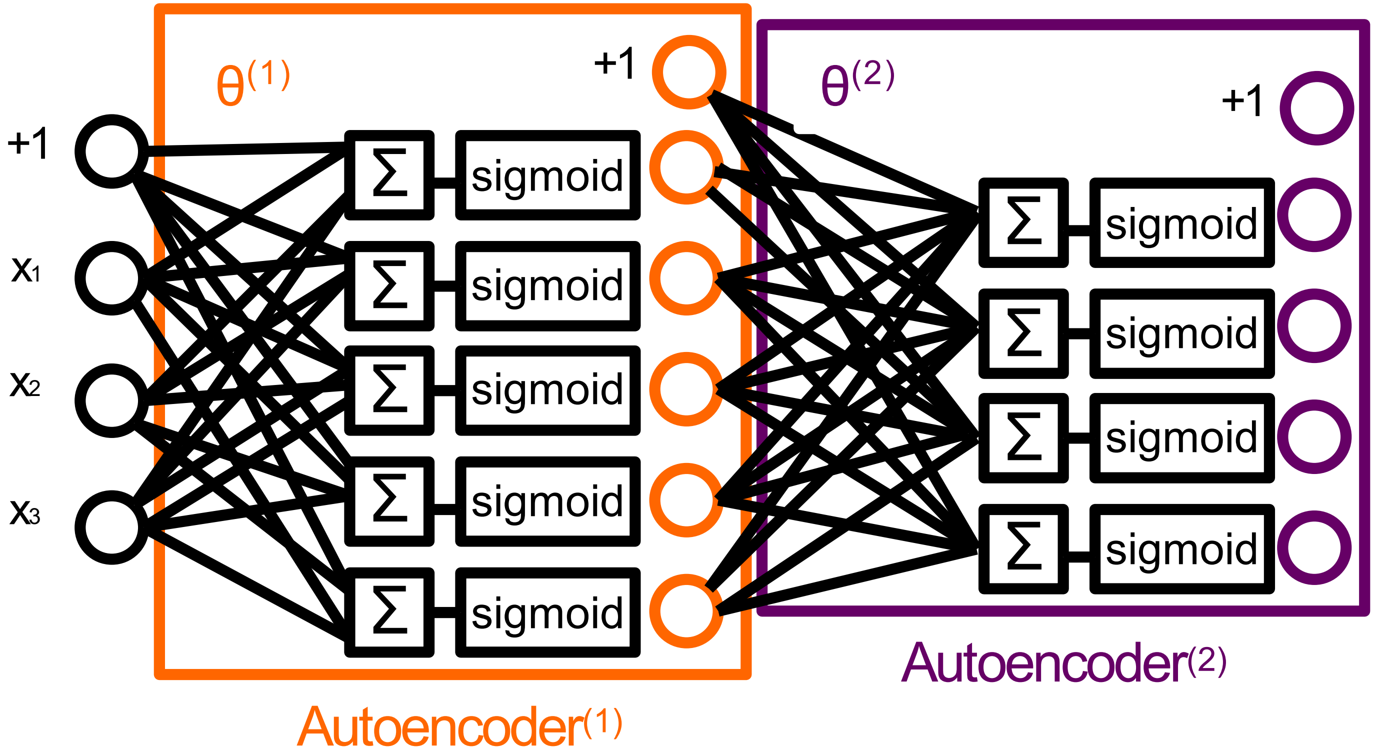


Learnt features(3)

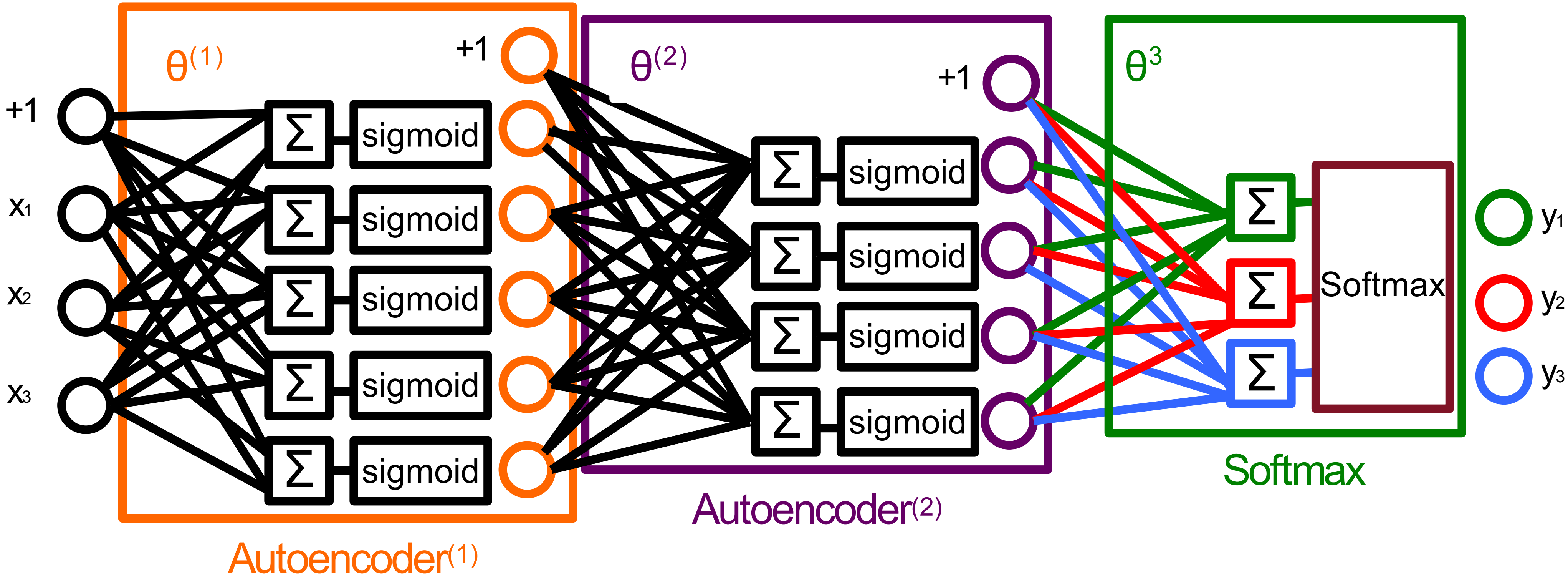
Assembling the Pre-Trained Deep Network



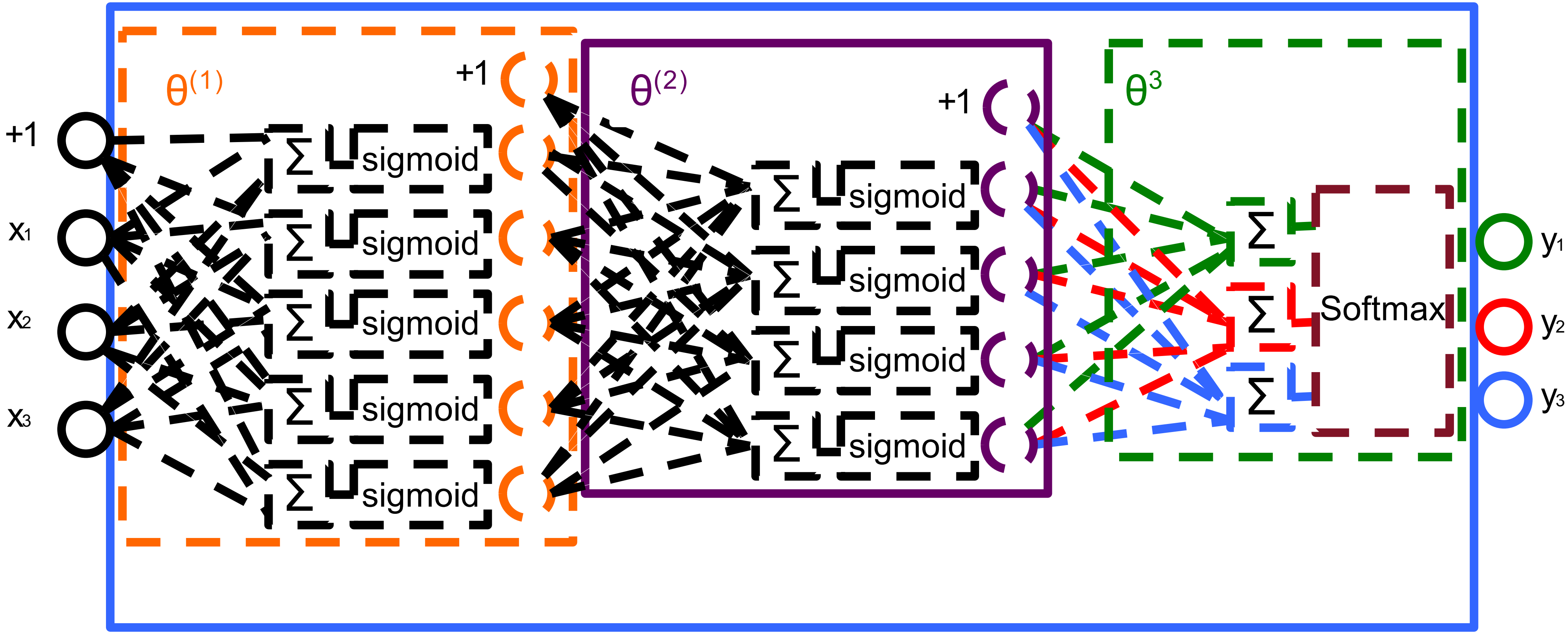
Assembling the Pre-Trained Deep Network



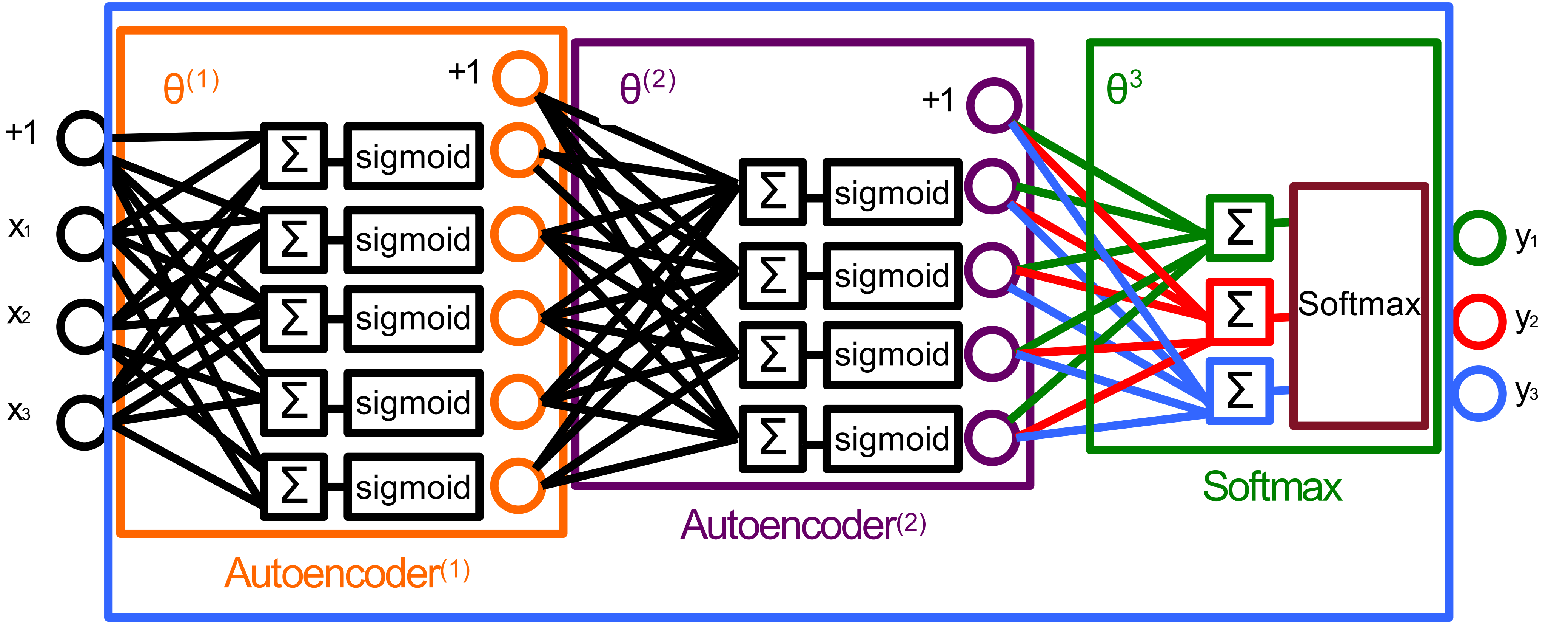
Assembling the Pre-Trained Deep Network
Functional Composition



Assembling the Pre-Trained Deep Network
Structural Composition – Composite Component



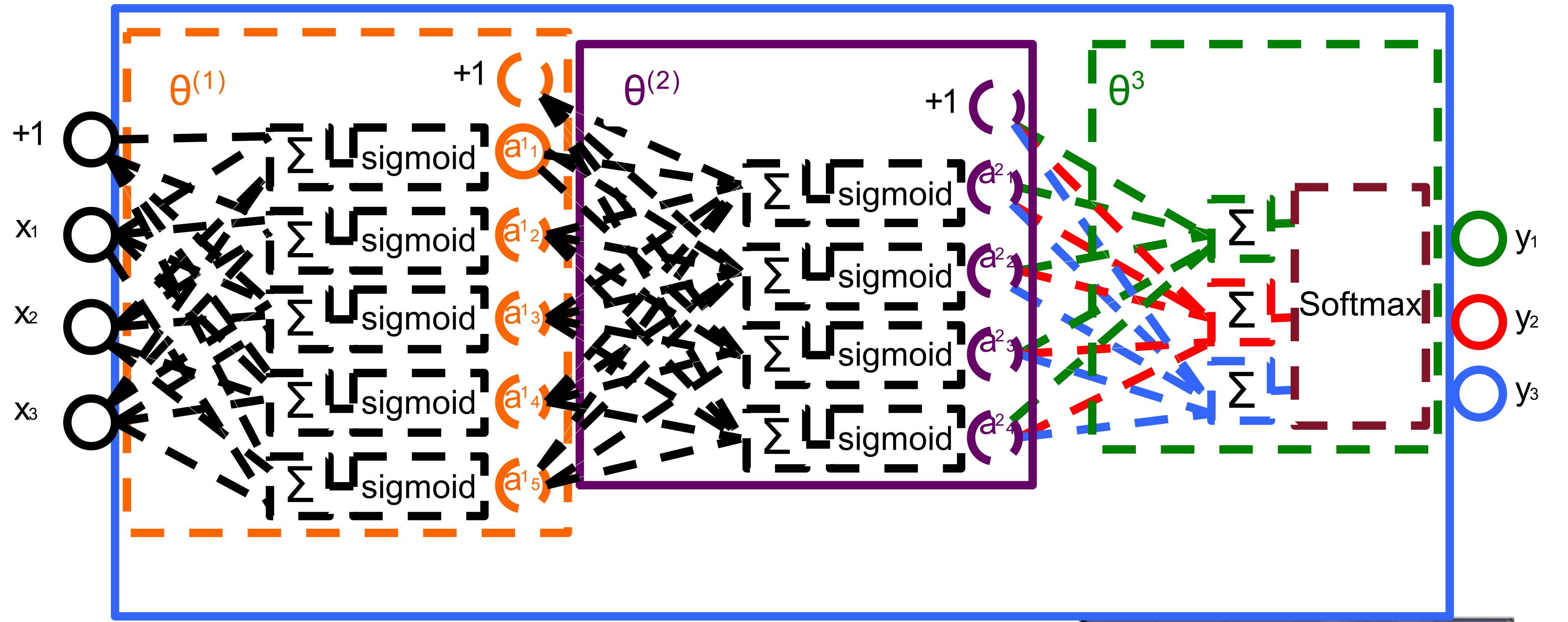
(Pre-Trained) Deep Network Final Global Training (Fine Tuning)

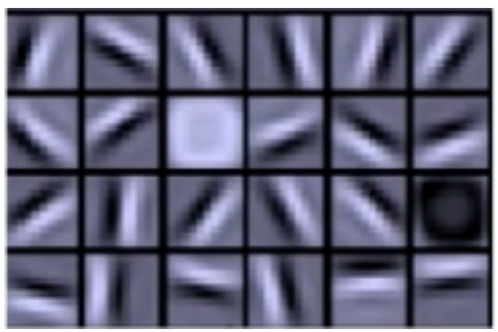


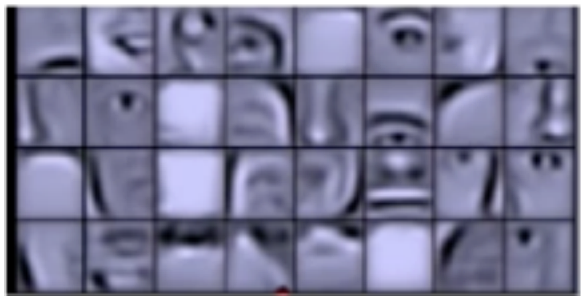
- Training (improving $\theta^{(1)}$, $\theta^{(2)}$, $\theta^{(3)}$) with Input Dataset : X : $\begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix}$ $\begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \end{bmatrix}$ … $\begin{bmatrix} x_1^m \\ x_2^m \\ x_3^m \end{bmatrix}$

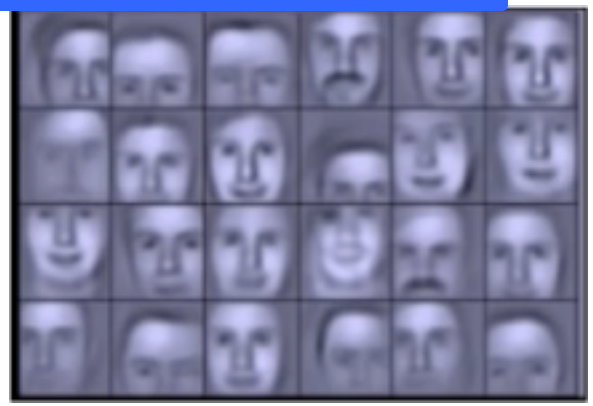
  and Output Dataset : y : $\begin{bmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \end{bmatrix}$ $\begin{bmatrix} y_1^2 \\ y_2^2 \\ y_3^2 \end{bmatrix}$ … $\begin{bmatrix} y_1^m \\ y_2^m \\ y_3^m \end{bmatrix}$

Successive Features/Abstractions Constructed



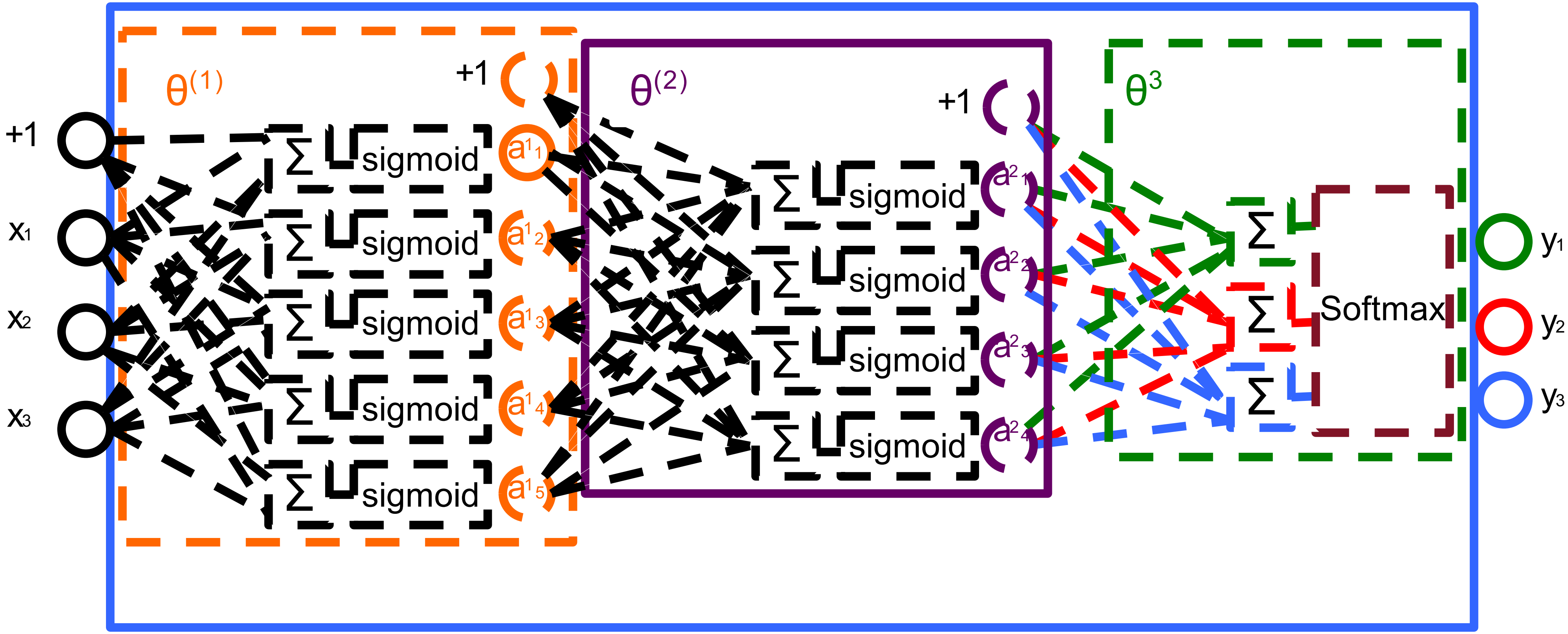


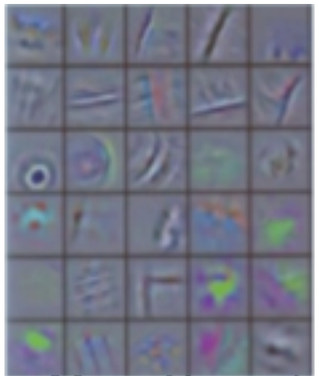


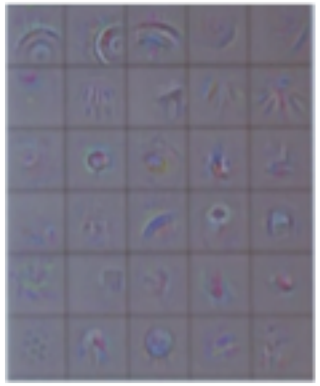


[Ng 2013]

Successive Features/Abstractions Constructed



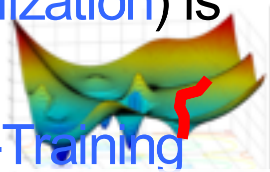




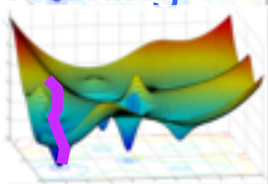
[Zeiler and Fergus 2013]

Summary – Key Ideas

1. Automatic Construction (Cascade Learning) of Hierarchical Abstractions/Features, which are Exposed and Useable (and Used, see 2.)
   – In Standard Multilayer Neural Networks, Features are Hidden (Black Box)
2. Accurate Initialization of Weights Matrixes for each Layer
   – In Standard Multilayer Neural Networks, Initialization of Weights Matrixes of each Layer is Heuristic, and is NOT Based on Information (Training Data/Examples)
   – InPre-trained Deep Networks, Initialization is based on Training Examples for the First Layer and on Successive Features extracted from Training Examples for Successive Layers
   – Therefore Initialization could be Much More Accurate
   – Remember that Initialization of Weights Matrixes is Critical because the Cost Function to be Minimized is Not Convex, therefore a Good Start (More Accurate Initialization) is a Better Promise to Avoid Falling into a Bad Local Minimum

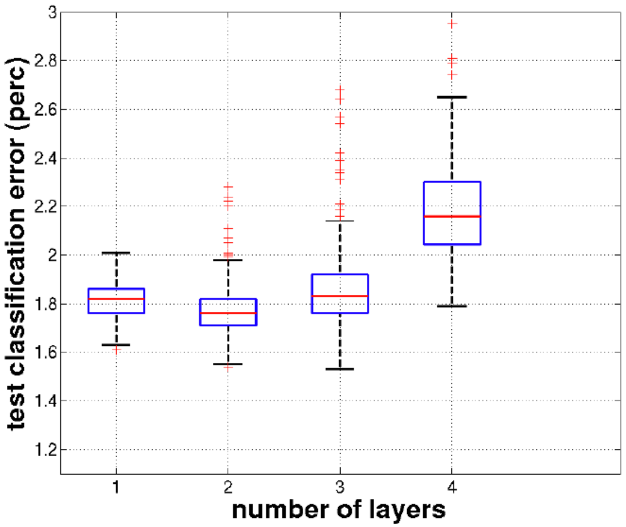


   – This Appears (at least conceptually/potentially) as a Main Advantage of Pre-Training Deep Networks [Erhan et al. 2010]

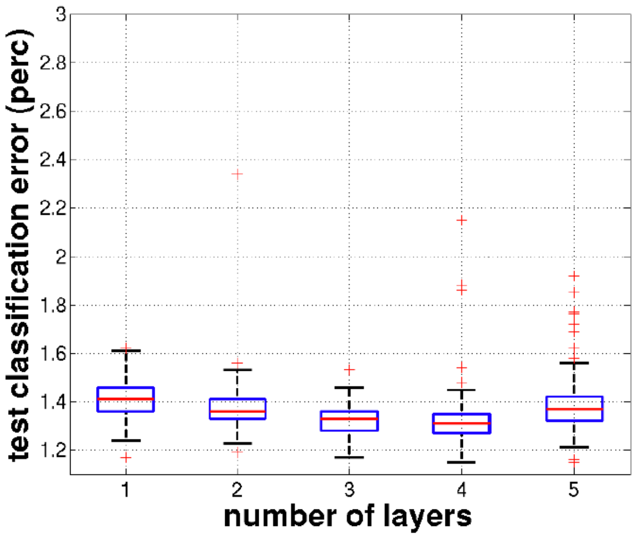


– Using Autoencoders (e.g. rather than Restricted Boltzmann Machines) is particularly Elegant/Economical, because Standard Supervised Learning Algorithm is used to implement/emulate Self-Supervised Learning

Analysis of Differences between Deep Network with and without Pre-Training [Erhan et al. 2010]

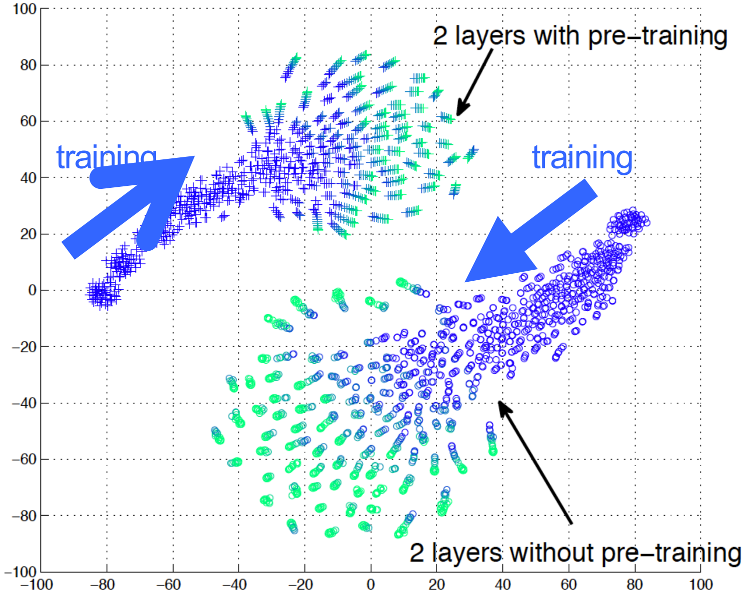


Without pre-training

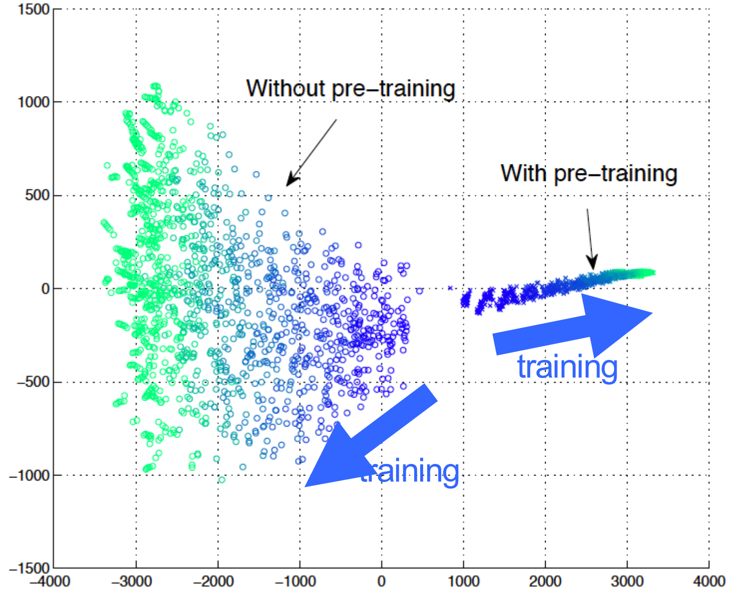


With pre-training

Training/Evolution of Hypotheses



Local structure-focused dimensity reduction (tNSE)

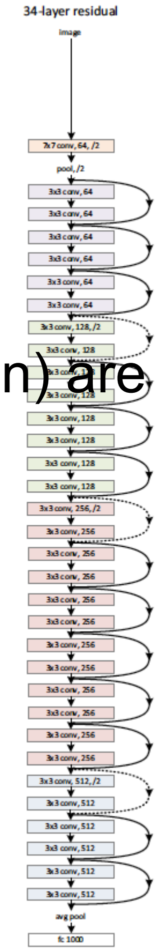


Global structure-focused dimensity reduction (ISOMAP)

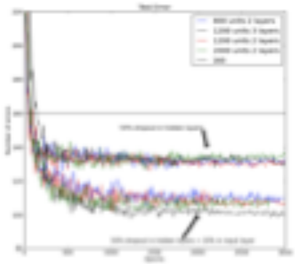
Gains

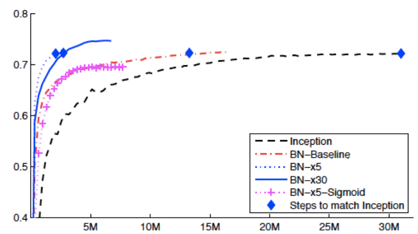
Rank	Name	Error rate	Description
1	**U. Toronto**	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

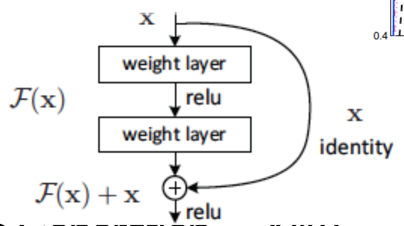
- (Significantly) Better Accuracy (upto 10% gain [Hinton 2009])
- Better Generalization (Regularization)
- Automatic Construction (Cascade Learning) of Hierarchical Abstractions/Features
- More Data Available for Training Autoencoders/Hidden Layers
  - Does not need Labels because Self-Supervised/Unsupervised
  - But Data should be from the same Domain to be learnt
  - Relation with Transfer Learning



- But Pre-Training is less Used at this Moment
- Because other Techniques Optimizing Learning (Improving Generalization) are Efficient
  - Preventing co-adaptation of feature detectors
    - » Randomly omitting half of the feature detectors
    - » Advanced Dropout
  - Batch Normalization [Ioffe & Szegedy, 2015]
    - » Normalization of each layer unit
  - Deep Residual Learning [He et al., 2015]
    - » Learning x -> x+y and not x -> y



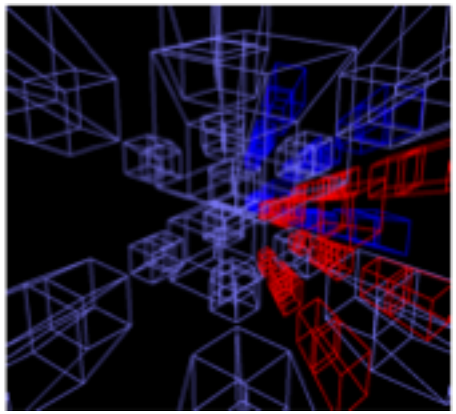




Manifold Representation

The Malediction of High Dimensionality [Mallat, 2018]

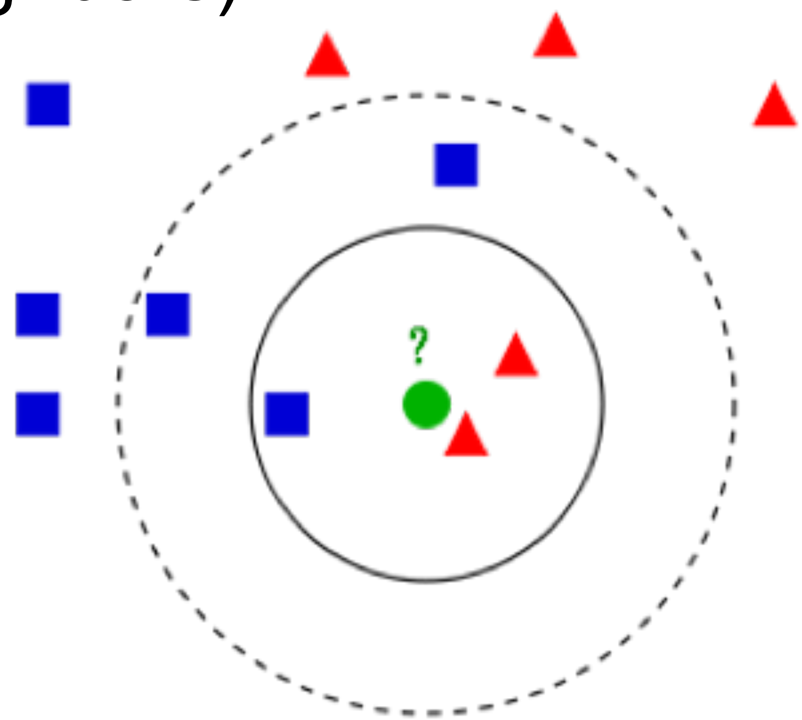
- Ex: Image with 2,000 x 1,000 pixels with color
- = 6.000.000 bits
- Space of dimension 6.000.000 !!



The Malediction of High Dimensionality [Mallat, 2018]

- Ex. of Task: Image recognition (Classification Task)

- kNN algorithm (k Nearest Neighbors)
- The class of an element =
  majority class
  of k nearest neighbors



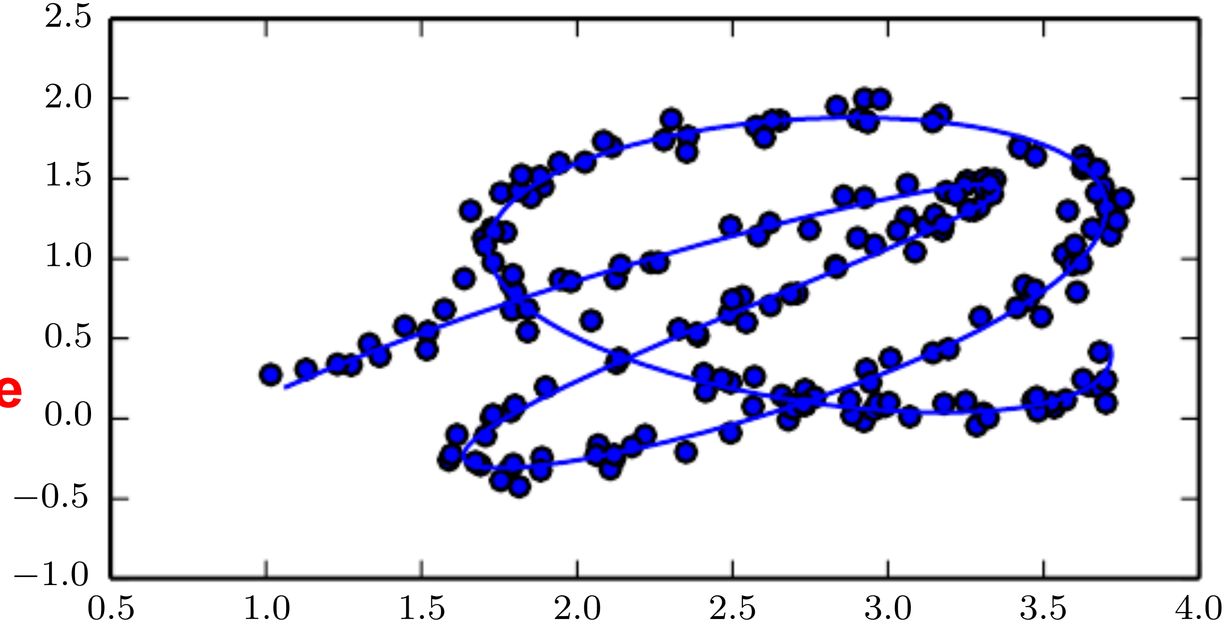
Problem:

- Euclidean distance is unhelpful in high dimensions because all vectors are almost equidistant to the query vector

Representation/Manifold Learning

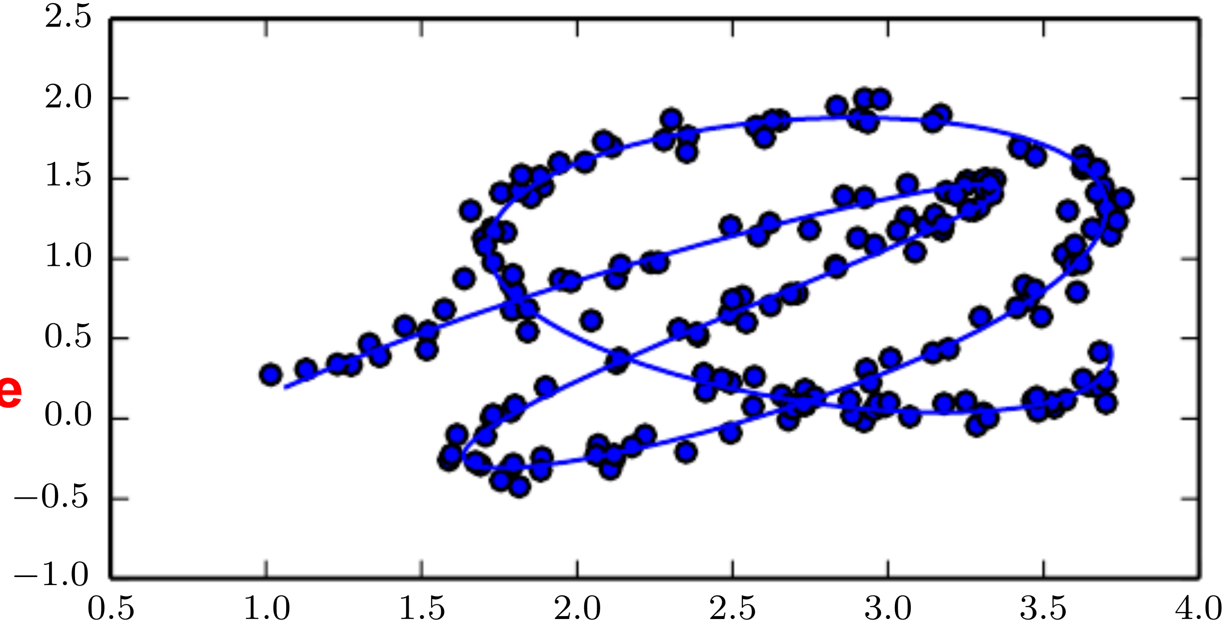
Manifold :
Set of connected points

In a high-dimensional space



[Goodfellow et al., 2016]

Representation/Manifold Learning



Manifold :
Set of connected points

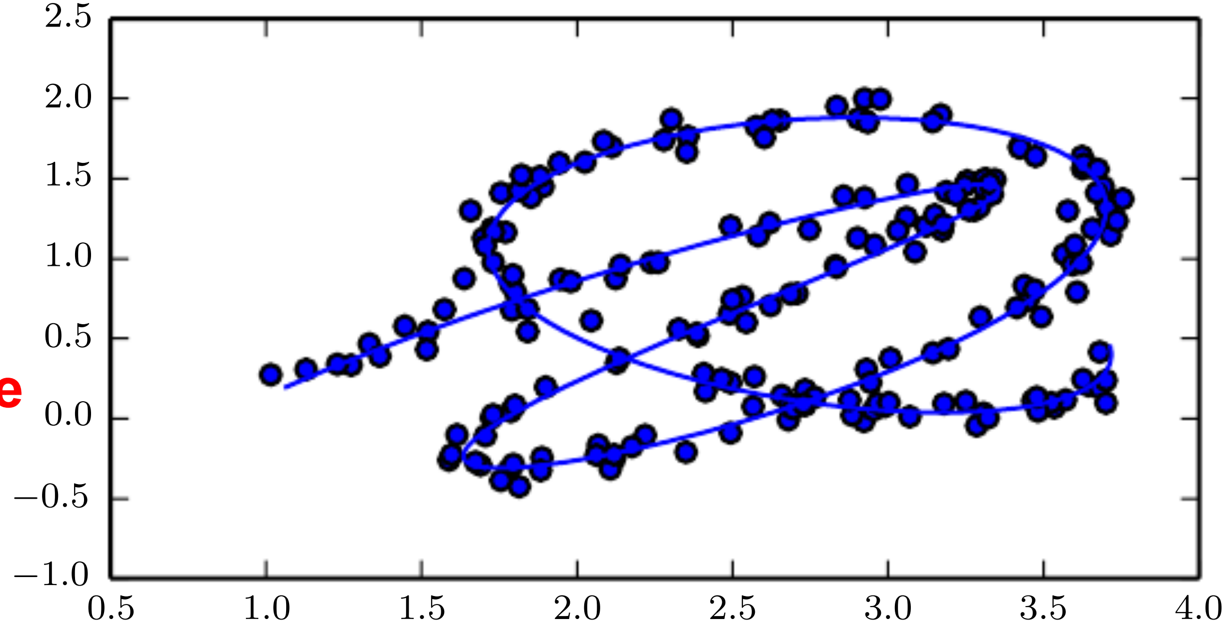
In a high-dimensional space

But can be approximated by a smaller number of dimensions, each dimension corresponding to a local variation

Representation/Manifold Learning

Manifold :
Set of connected points

In a high-dimensional space



But can be approximated by a smaller number of dimensions, each dimension corresponding to a local variation

Analogy:

3D Earth **2D Map**