

Deep Learning Techniques for Music Generation Autoencoder (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



Deep Learning for Music Generation – Technical Challenges

1. *Ex Nihilo* Generation

» vs Accompaniment (Need for Input)

2. Length Variability

» vs Fixed Length

3. Content Variability

» vs Determinism

4. Control

» ex: Tonality conformance, Maximum number of repeated notes...

5. Structure

6. Originality

» vs Conformance

7. Incrementality

» vs Single-step or Iterative Generation

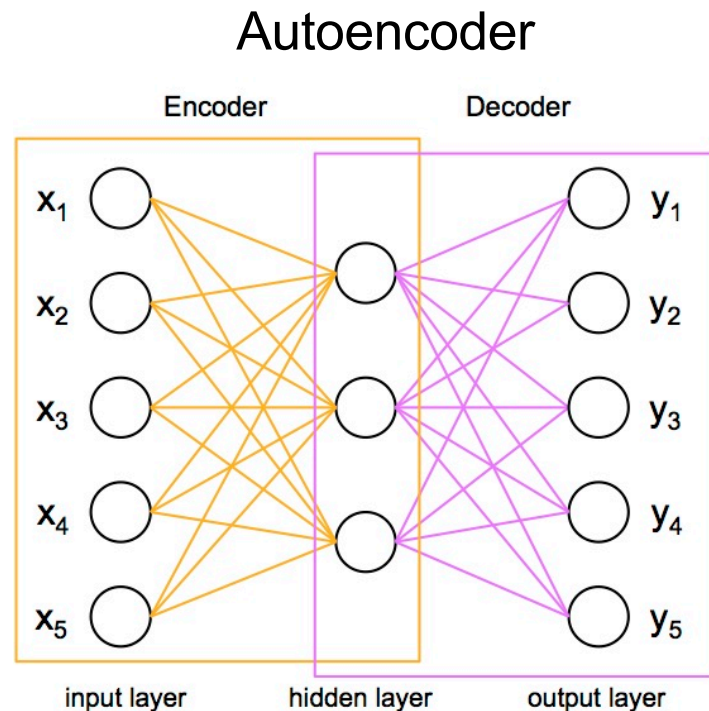
8. Interactivity

» vs (Autistic) Automation

9. Explainability

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

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



Neural Network with Input Layer = Output Layer

- Diabolo shape

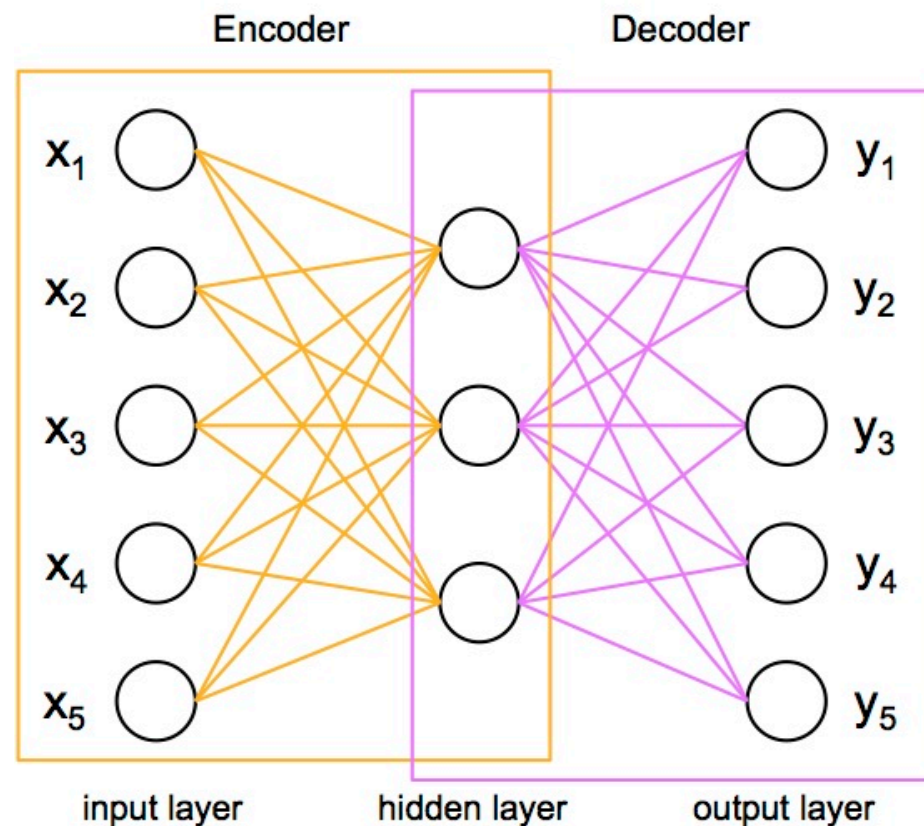


- Self-Supervised Training

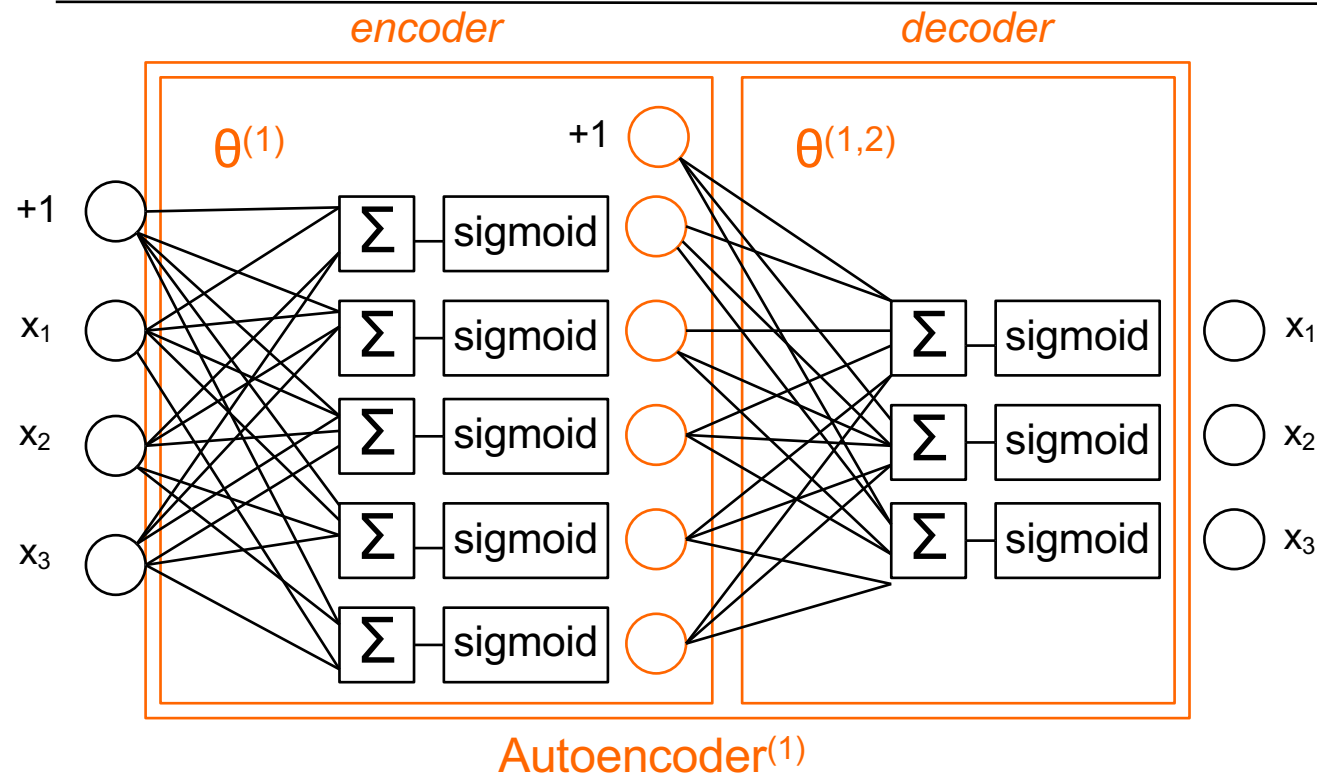
- Output = Input
- Learning Identity
- Learns to compress and reconstruct data
- Extracts significant/discriminating features

Autoencoder

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



Autoencoder Self-Supervised Training



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

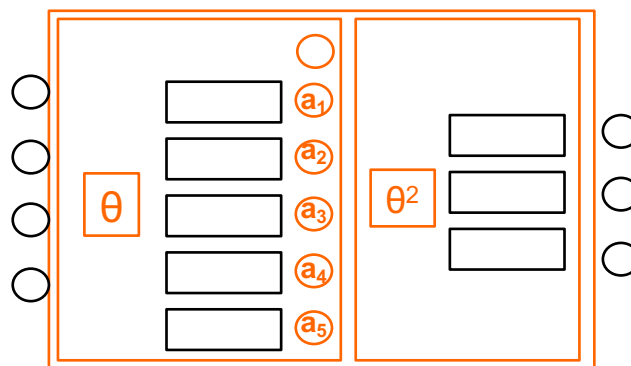
Sparse Autoencoder Learning Features

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

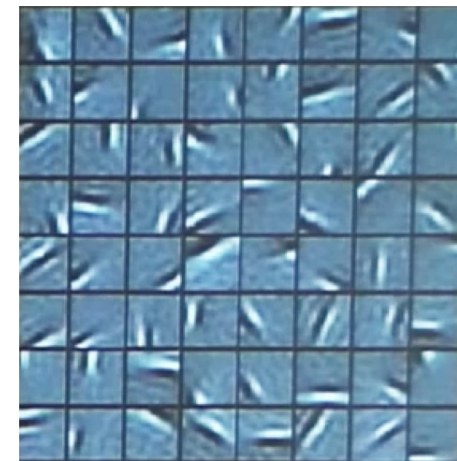


Images (Examples)

x_i



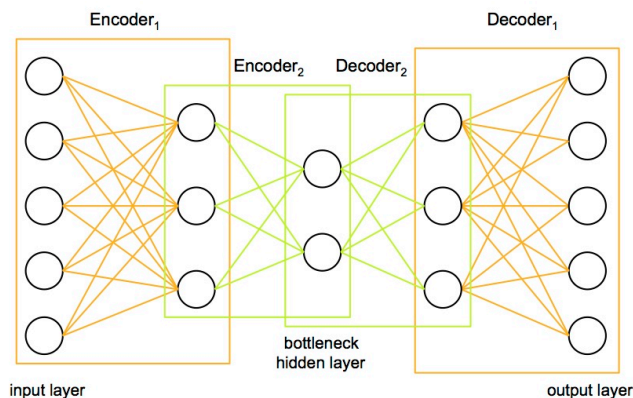
Autoencoder



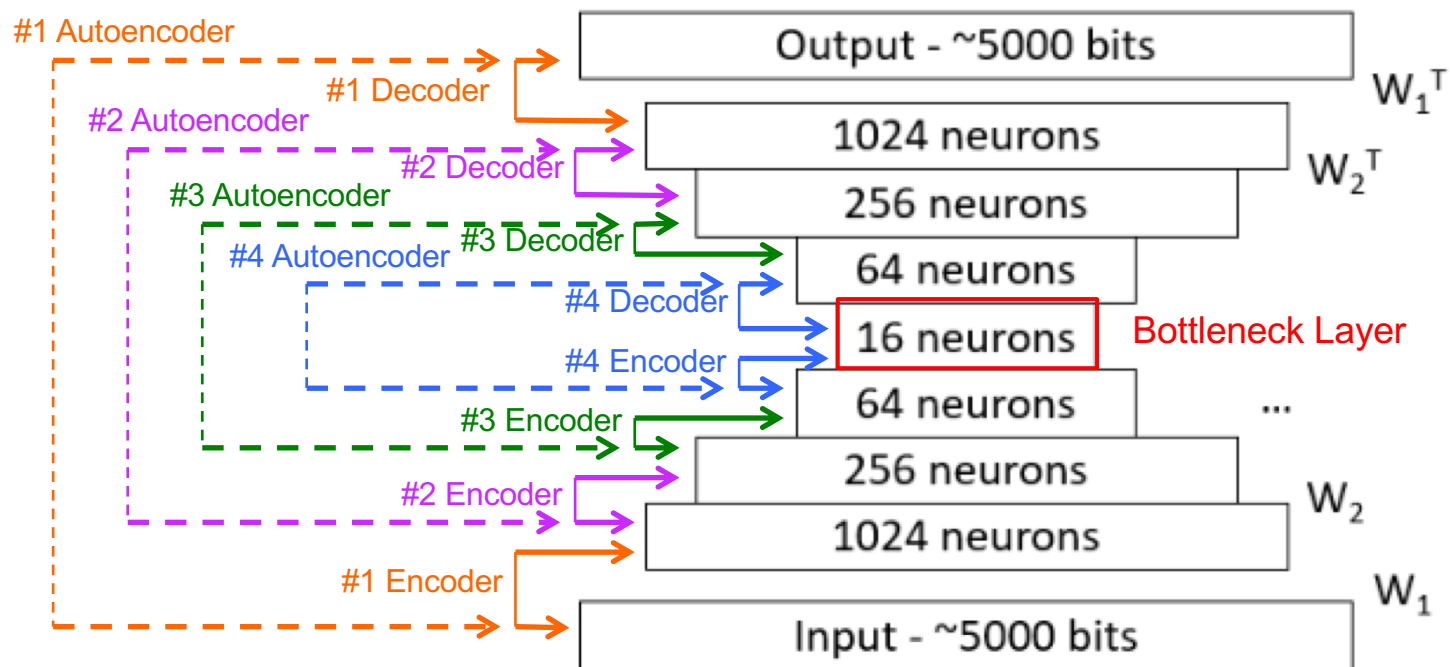
Features: $\Phi_1, \dots, \Phi_k$

- "Invents" (learns) higher level features (e.g., edges)
- Sparsity forces **specialization** (**feature detector**) of each unit [Ng, 2013]
- Alternative Non supervised learning architectures, e.g., Restricted Boltzman Machines (RBM) [Smolensky 1986] [Hinton et al. 2006]

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



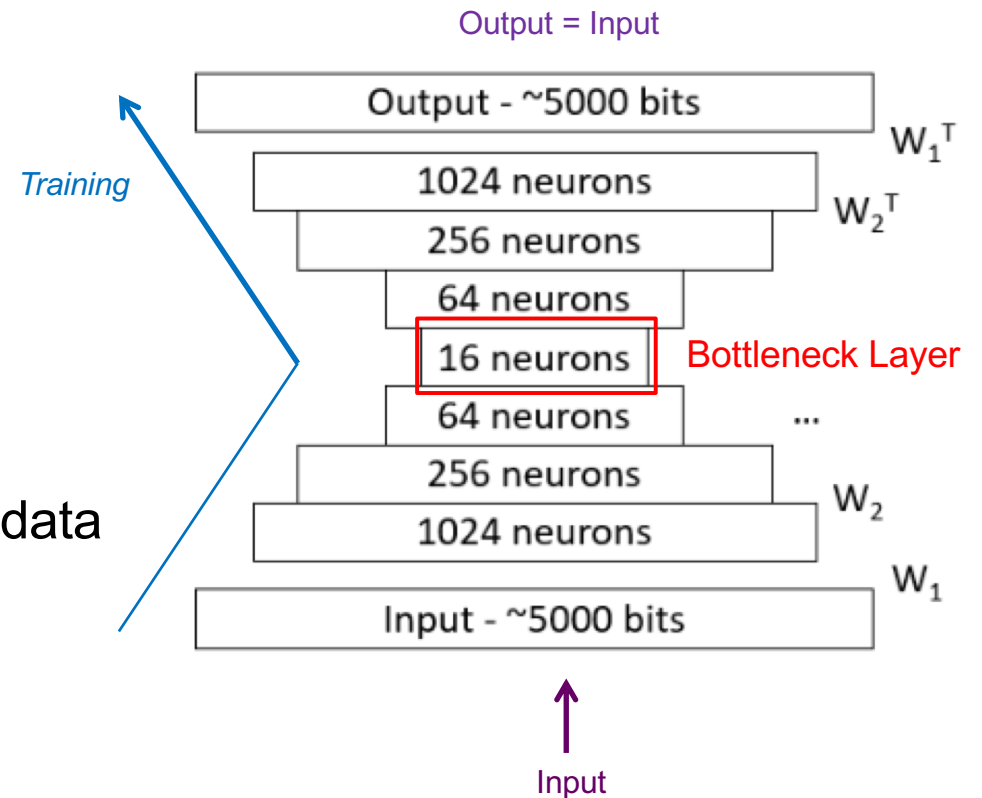
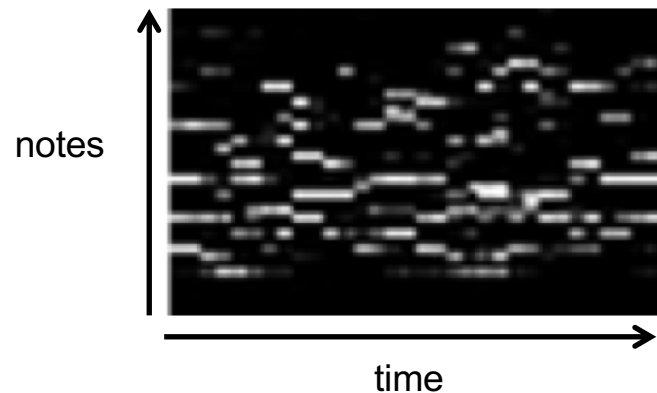
DeepHear Architecture [Sun, 2016]



Ex1: DeepHear [Sun, 2016]

1. Pre-Training in Cascade (Layer by Layer)

- Dataset: 600 Ragtimes (Scott Joplin)
- Representation: Pianoroll



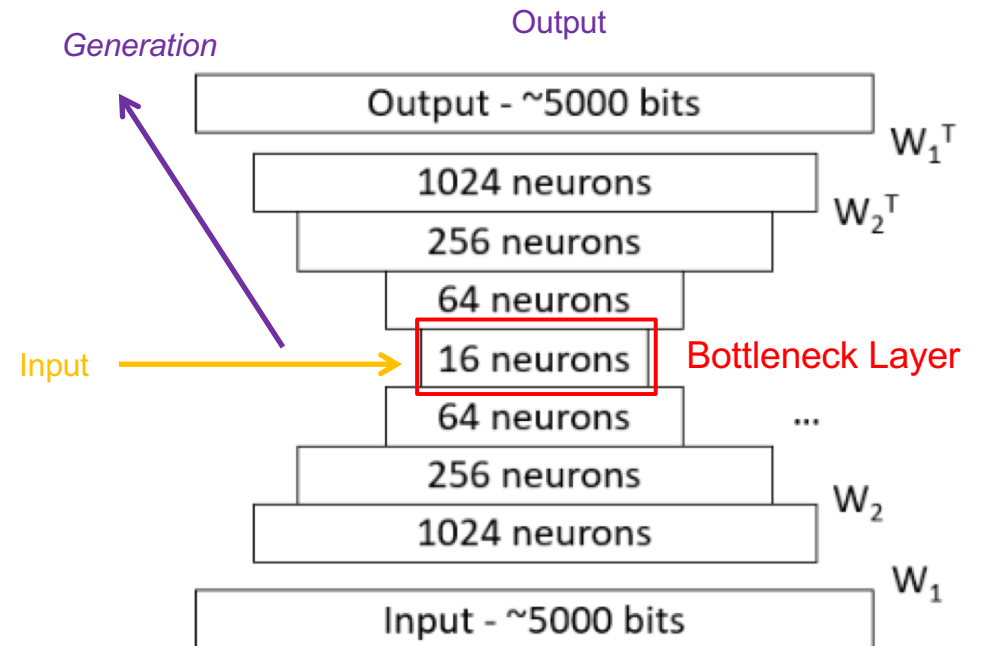
2. Self-Supervised Training

- Learns to compress and reconstruct data
- Extracts Features
- Bottleneck Layer = 16 neurons

DeepHear

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>

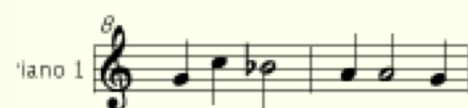
Auto(Encoder)Bach

Label elements all 0

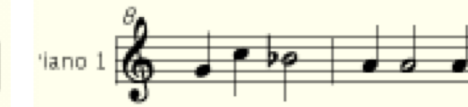
Corpus:
Soprano parts of Bach Chorales



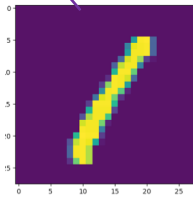
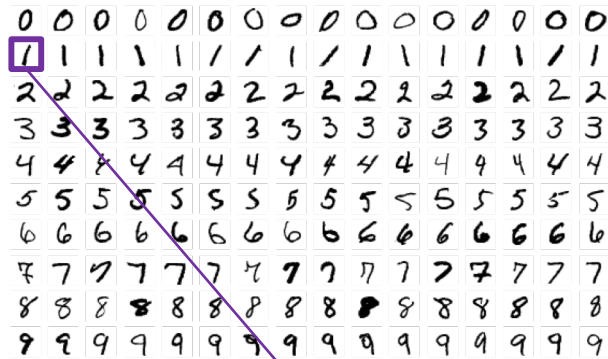
Label elements all 1



Label elements random [0, 1]



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()
```

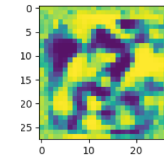
784

200

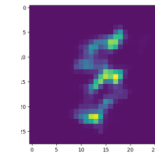
label1



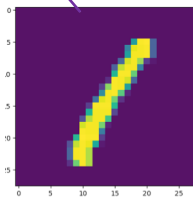
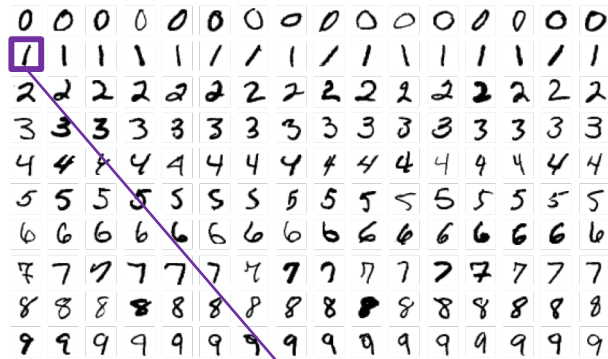
digit1



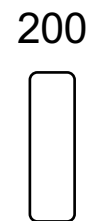
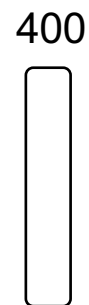
digit2



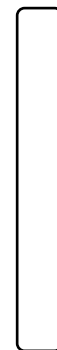
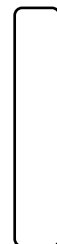
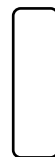
Autoencoder MNIST (Handwritten Digits)



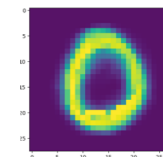
784



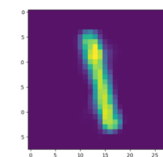
label1



digit1



digit2

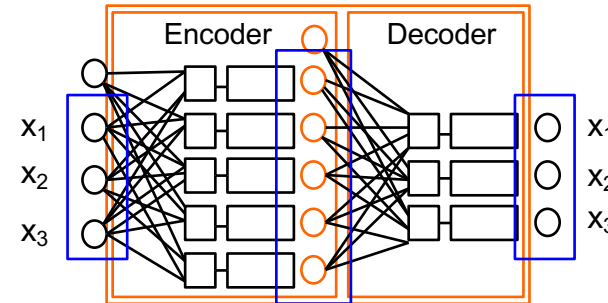


Other Example – Creating Non-Digits

New Types Generation [Kazakçi et al., 2016]

1. Create a Sparse Autoencoder

- Convolutional Autoencoder
(3 Encoding Layers; 1 Decoding Layer)



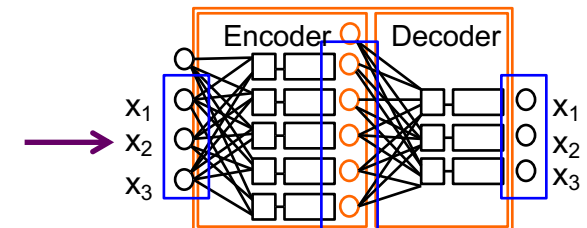
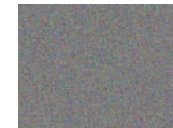
2. Train it on Objects Dataset

- Hand Written Digits: MNIST dataset [Lecun & Cortes, 2012]

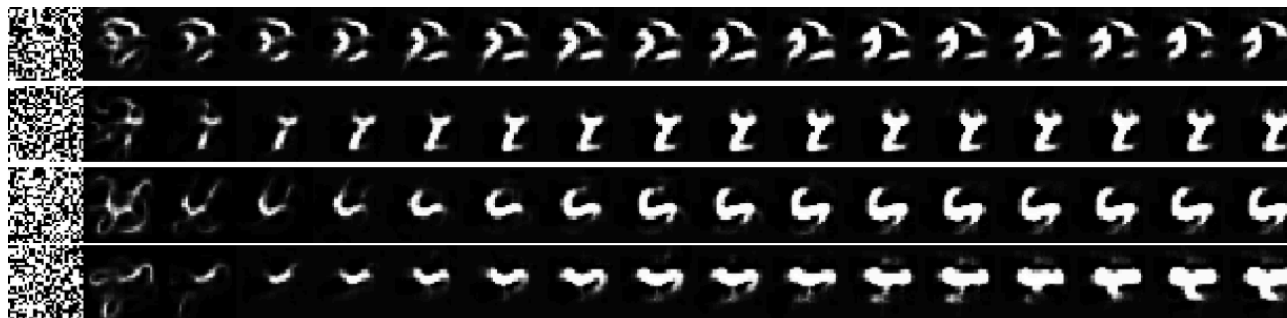


3. Feed Forward Random Image into Autoencoder

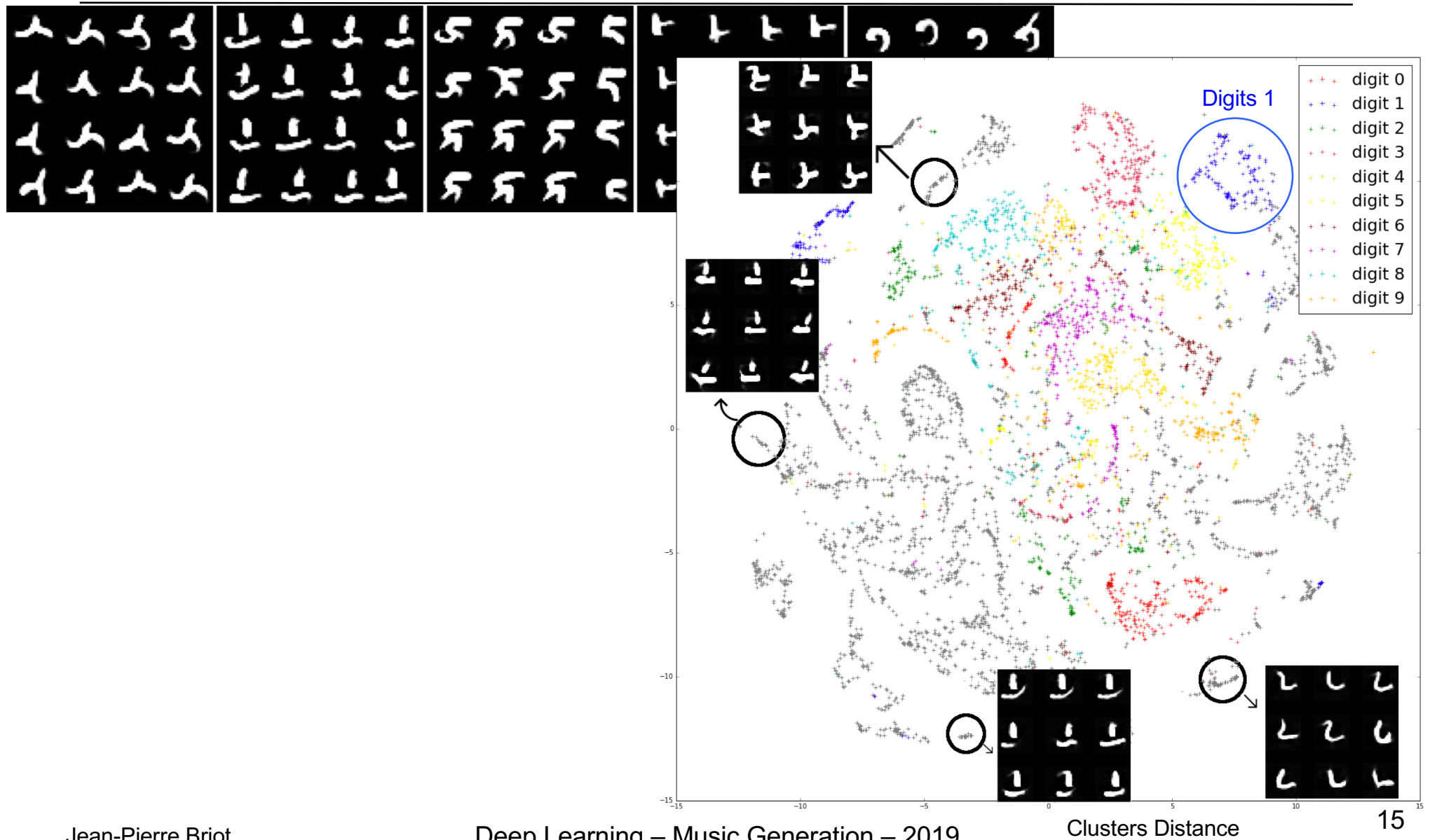
- Output Image is Degraded but Features Emerge



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



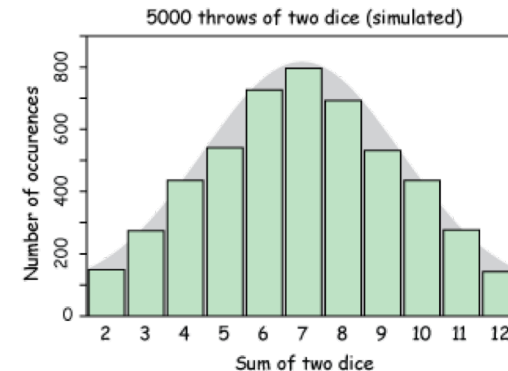
New Types Generation



Variational Autoencoder

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

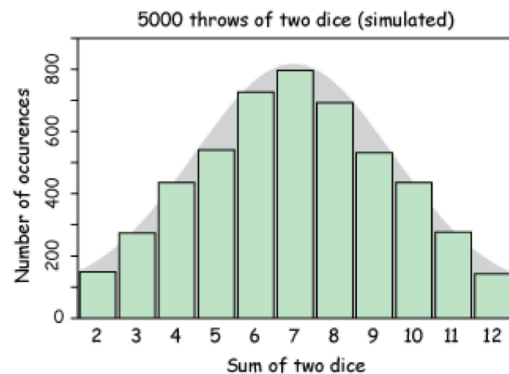
- *Constraint:*
- Encoded representation (latent variables z) follow some prior probability distribution $p(z)$, usually, a Gaussian distribution (normal law)



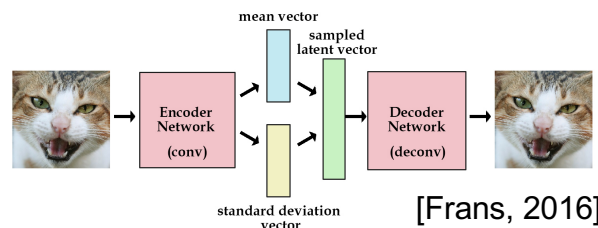
- *Non optimized implementation:*
- Adding a specific term to the cost function, by computing the cross-entropy between the values of the latent variables and the prior distribution
- The VAE decoder part will learn the relation between a Gaussian distribution of the latent variables and the learnt examples
- A VAE is able to learn a *smooth* latent space *mapping* to realistic examples

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

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



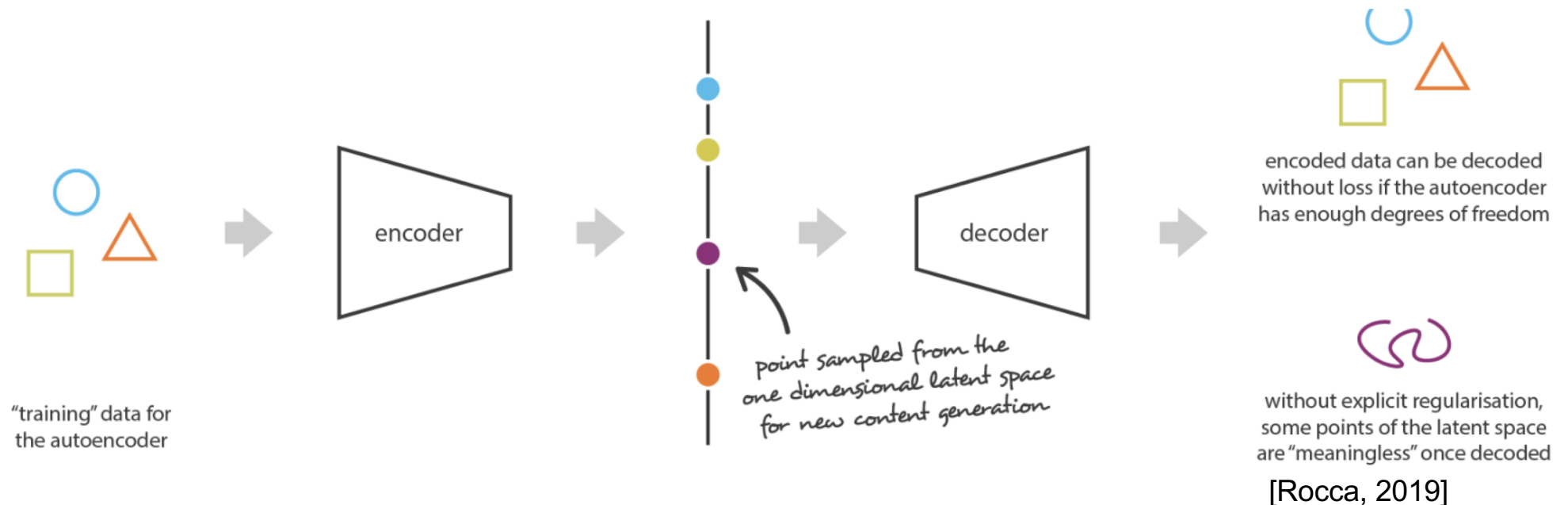
- *Reparameterization Trick*



- 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*

AE vs VAE (1/3)

- Inspired from
- Joseph Rocca, Understanding Variational Autoencoders (VAEs) – Building, step by step, the reasoning that leads to VAEs, Towards Data Science Blog, 23 September 2019
- <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>
- **Irregular latent space** prevents us from using autoencoder for new content generation



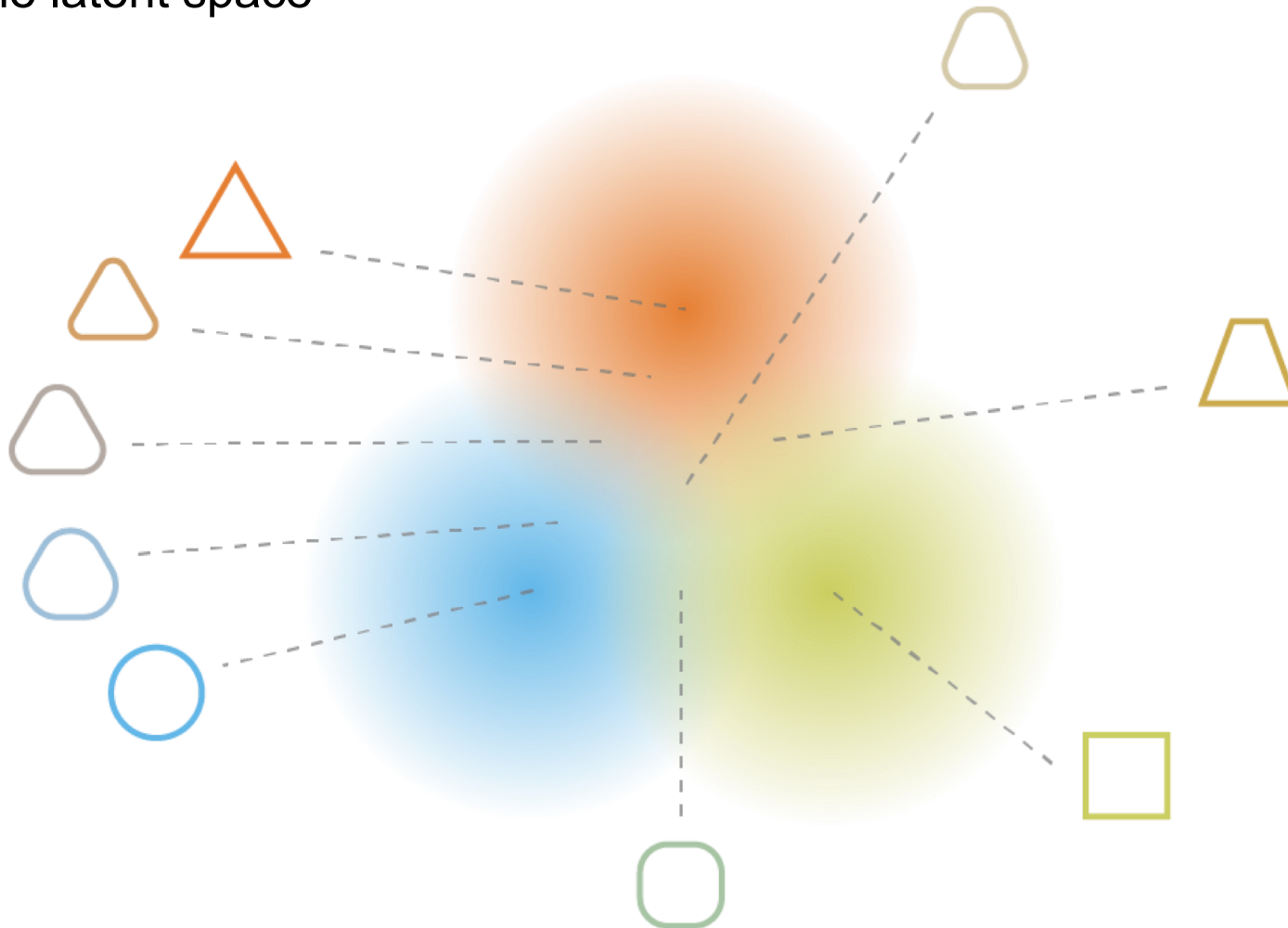
AE vs VAE (2/3)

- Regularization brings:
 - Continuity
 - Completeness
- By enforcing distribution produced by the encoder to be close to a standard normal distribution:
 - Covariance matrices to be close to the identity, preventing punctual distributions
 - Mean to be close to 0, preventing encoded distributions to be too far apart from each others

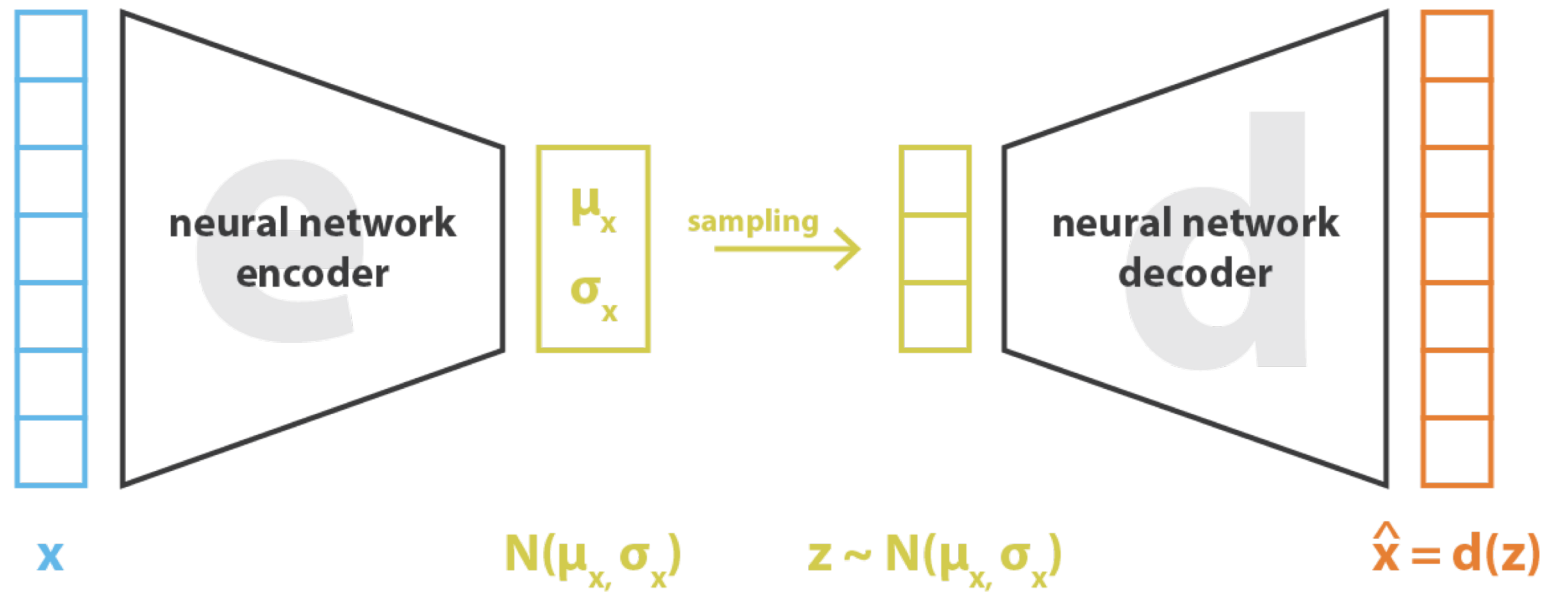


AE vs VAE (3/3)

- Regularisation tends to create a “gradient” over the information encoded in the latent space



AE vs VAE



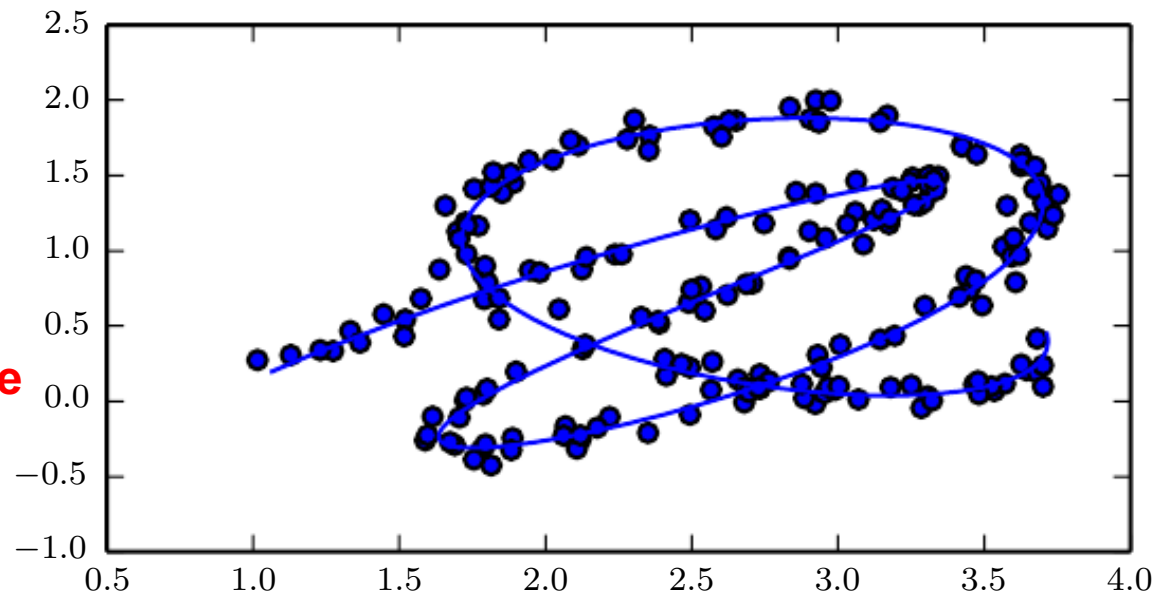
$$\text{loss} = ||x - \hat{x}||^2 - \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 - \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

[Rocca, 2019]

Representation/Manifold Learning

Manifold :
Set of connected points

In a **high-dimensional space**

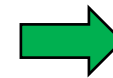


But can be approximated by
a **smaller number of dimensions**,
each dimension corresponding
to a **local variation**

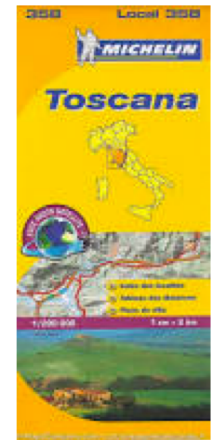
Analogy:



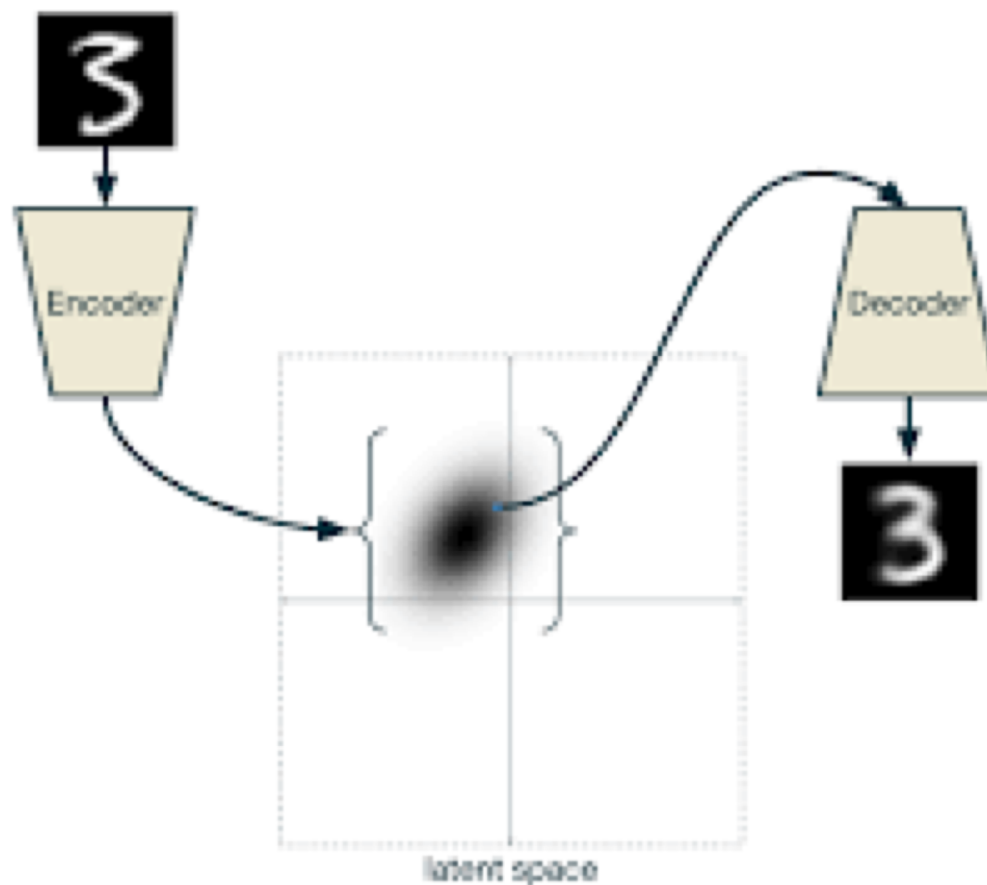
3D Earth



2D Map



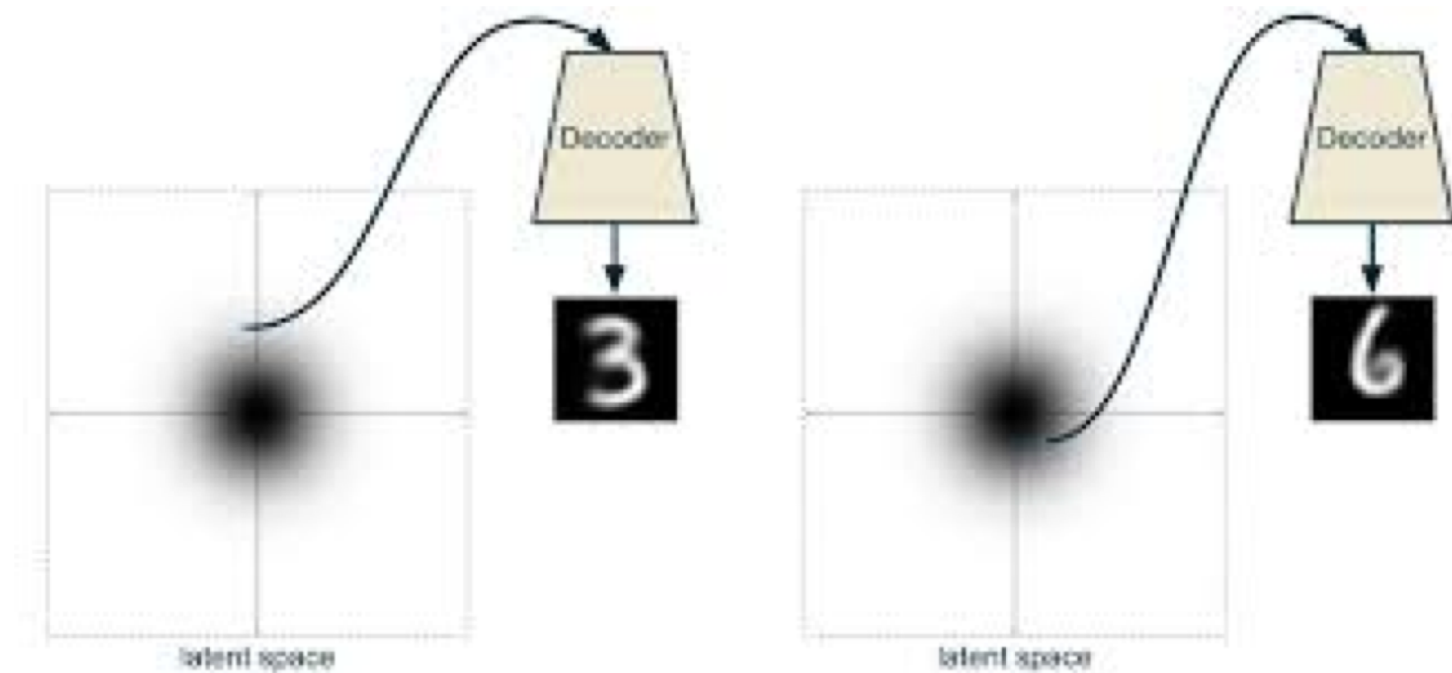
Variational Autoencoder



[Dykeman, 2016]

Variational Autoencoder

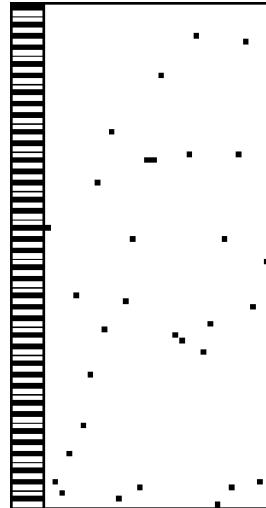
Generation
by Exploring the Latent Space
and Decoding



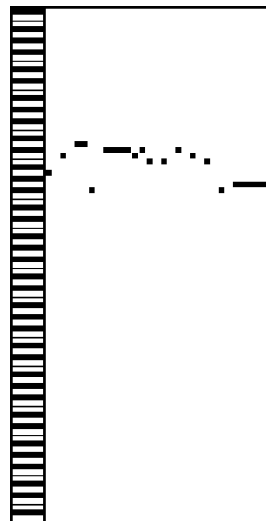
[Dykeman, 2016]

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:

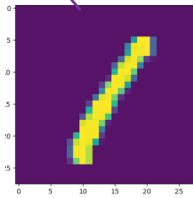
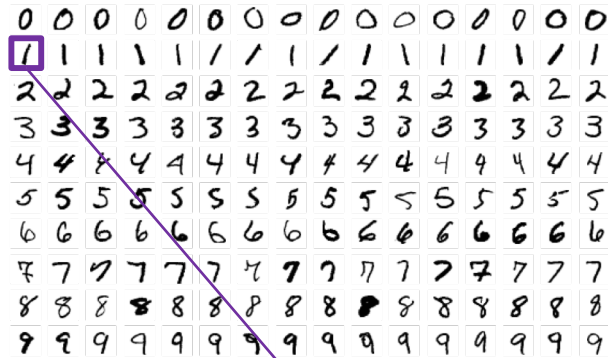
- Translation
- Arbitrary path
- Interpolation (morphing) (between points)
- Averaging (of some points)
- Attribute arithmetic
 - Addition or subtraction of an attribute vector capturing a given characteristic
 - This attribute vector is computed as the average latent vector for a collection of examples sharing that attribute (characteristic)

Variational Autoencoder

Ex. of Attribute Arithmetic



VAE MNIST [Keras/Cholet, 2016]



784

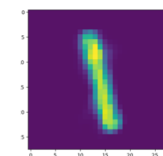
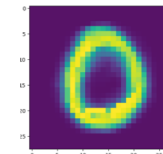
512

2

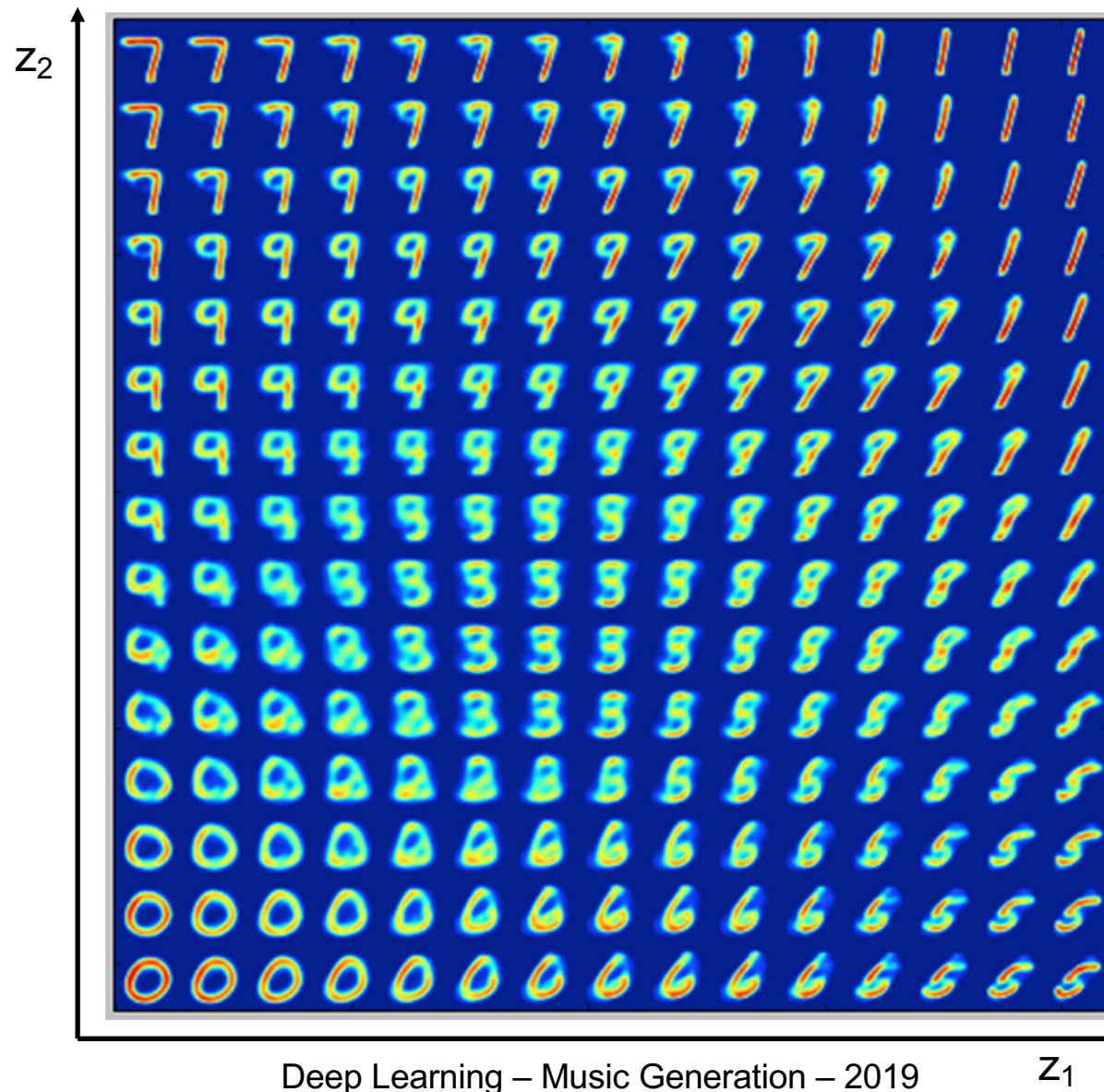
Label = (z_1, z_2)

digit1

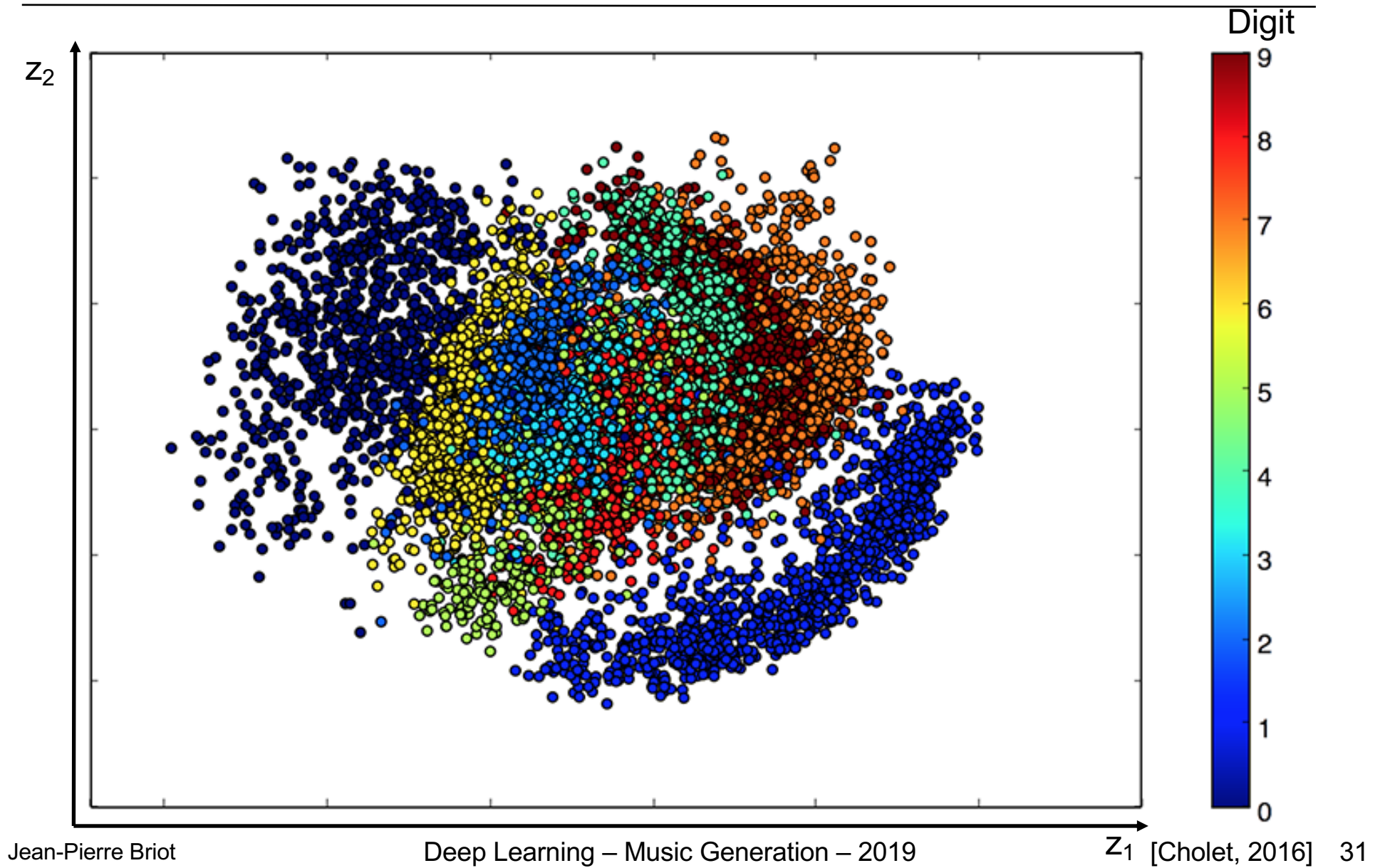
digit2



VAE MNIST [Keras/Cholet, 2016]



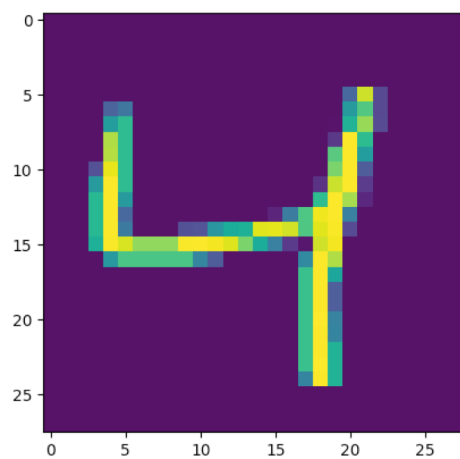
Latent Space Distribution



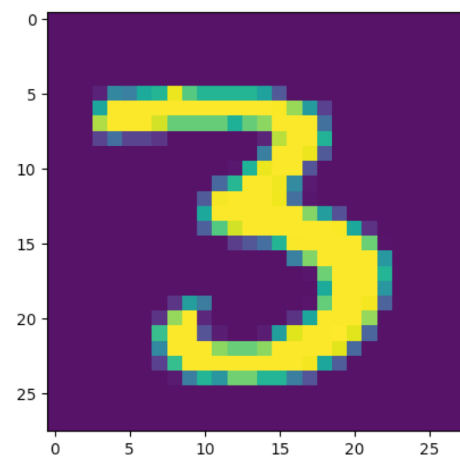
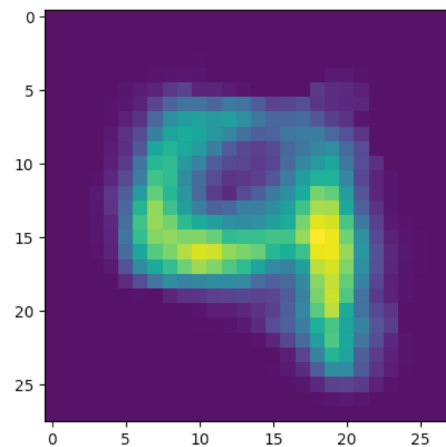
Attribute Arithmetic

- (Characteristics) Attribute Arithmetic
 - 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)
- Select a set of round and angular digits images
 - `round_numbers = [3, 6, 8, 9]`
 - `angular_numbers = [1, 4, 7]`
- Encode each one
 - `_, _, z_round_elements = encoder.predict(np.array(round_elements))`
 - `_, _, z_angular_elements = encoder.predict(np.array(angular_elements))`
- Compute the mean of the (z) corresponding latent variable values
 - `z1_mean_round_elements = mean(z1_round_elements)`
 - `z1_mean_angular_elements = mean(z1_angular_elements)`
 - ...
- Do attribute arithmetic
 - `def roundify(z):`
 - `z_rounded = [z[0] + z1_mean_round_elements, z[1] + z2_mean_round_elements]`
 - `return(decoder.predict(np.array([z_rounded]))[0])`

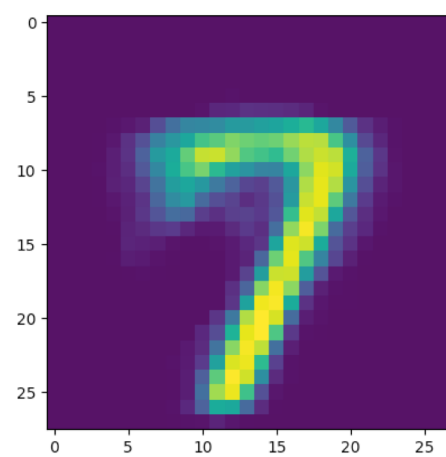
Examples



Roundify



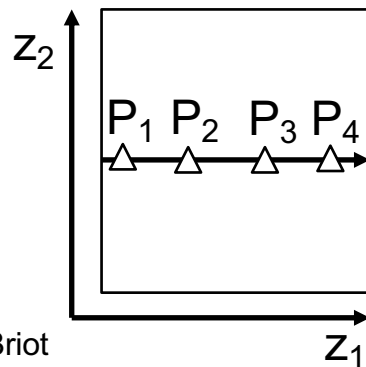
Angularify



AutoVarBach

Z_1 Step Interpolation

Four musical staves, labeled P_1 , P_2 , P_3 , and P_4 from top to bottom, illustrating a step interpolation. Each staff contains two staves of music (treble and bass clef). The top staff (P_1) is in $\sharp\sharp$ (F# and C#) and 4/4 time. The bottom staff (P_4) is in $\flat\flat$ (Bb and Fb) and 4/4 time. The intermediate staves (P_2 and P_3) show a gradual transition in pitch and rhythm between the two extremes. To the right of each staff is a speaker icon, indicating that the music can be played.



AutoVarBach

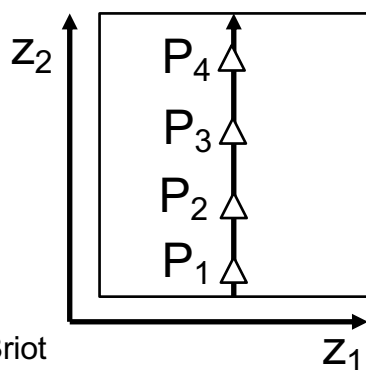
Z_2 Step Interpolation

P_1  

P_2  

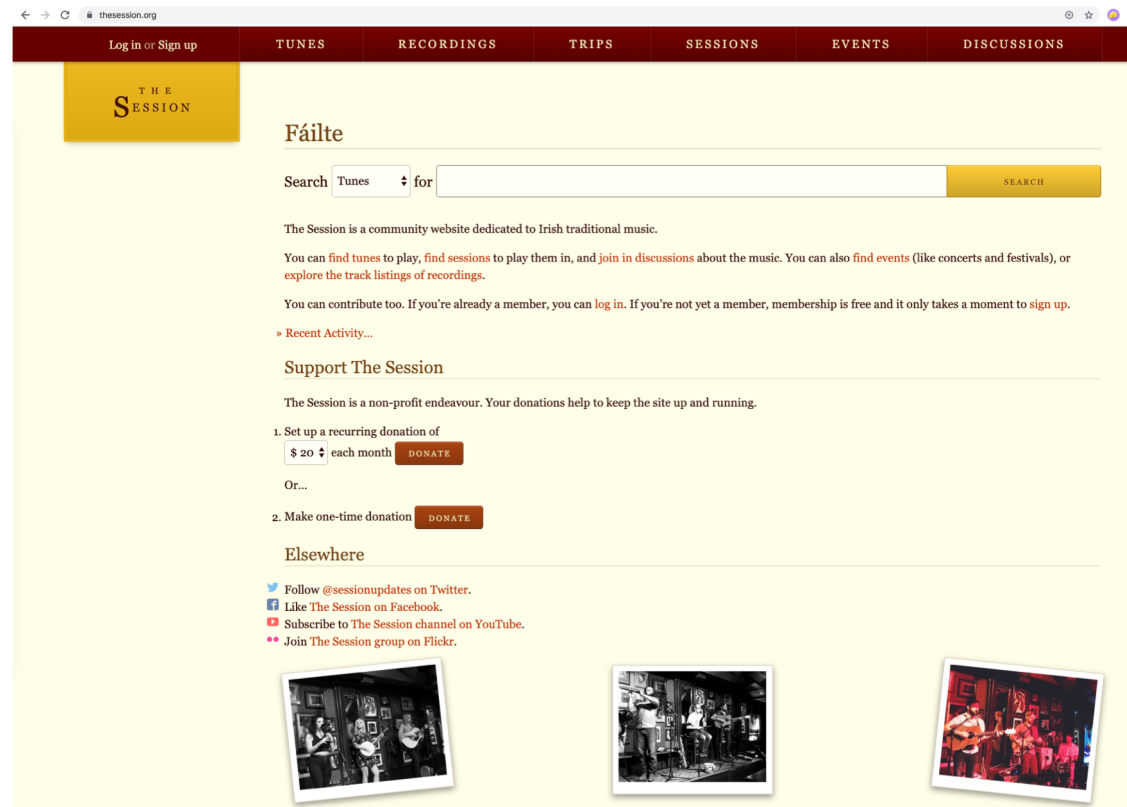
P_3  

P_4  



Celtic Music

- Training Examples/Corpus:
- In ABC format (see later) -> Music21 -> representation
- 29 songs from the Session (<https://thesession.org/>)
- In the same key (D major) and the same rhythm metric (4/4)



AutoVarCeltic

Z₁ Step Interpolation

P₀



P₁



Jean-Pierre Briot

P₂



P₃



P₄



AutoVarCeltic

Z₂ Step Interpolation

P₁



P₂



P₃



P₄



AutoVarCeltic (4 Layers: #1000; #200; #50; #20 – 200 Epochs)

Z_1 Step Interpolation

P_1



5



P_4



7



AutoVarCeltic (4 Layers: #1000; #200; #50; #20 – 200 Epochs) Z₂ Step Interpolation

P₀



5



P₄



7



Disentanglement (1/3)

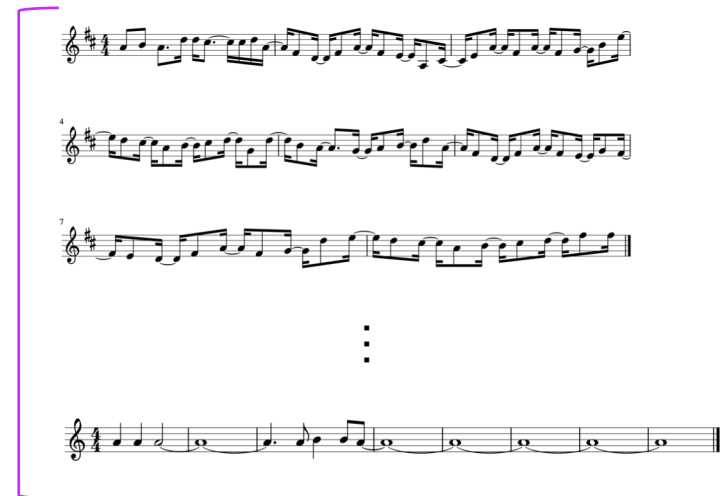
- $z_1 \perp z_2$



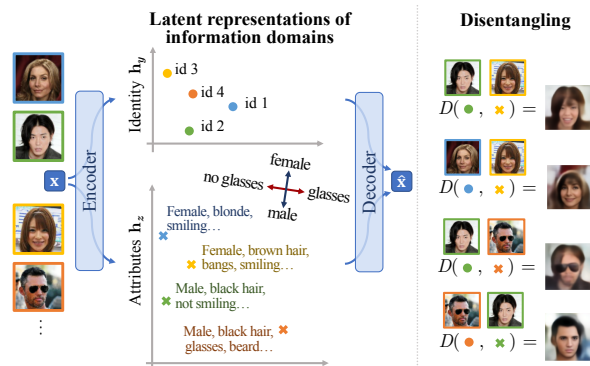
- Ex: Pitch Range $\perp$



- Duration Range



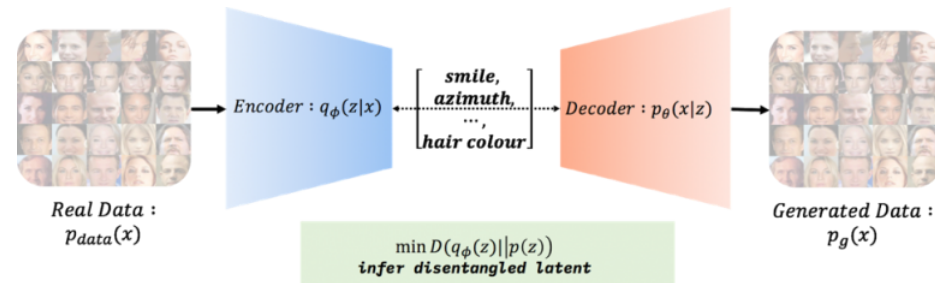
- Ex: Gender $\perp$ Glasses



[Robert et al, 2019]

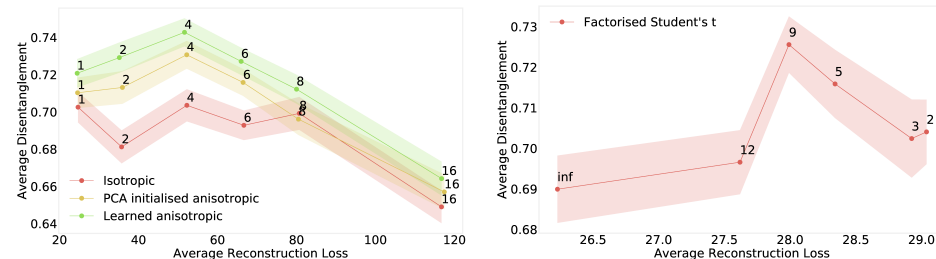
Disentanglement (2/3)

- Adding Term to the Reconstruction Loss [IBM Research, 2018]

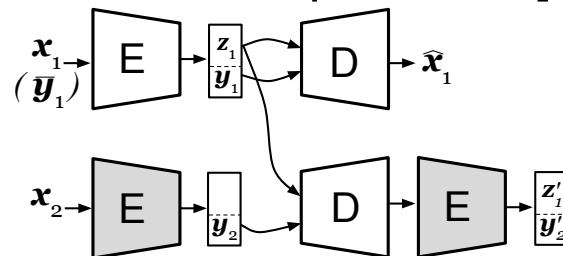


$$\max_{\theta, \phi} \mathbb{E}_{\mathbf{x}} [\mathbb{E}_{\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})] - \text{KL}(q_{\phi}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))] - \lambda D(q_{\phi}(\mathbf{z})||p(\mathbf{z}))]$$

- Deconstructing the β -VAE [Mathieu et al., 2019]

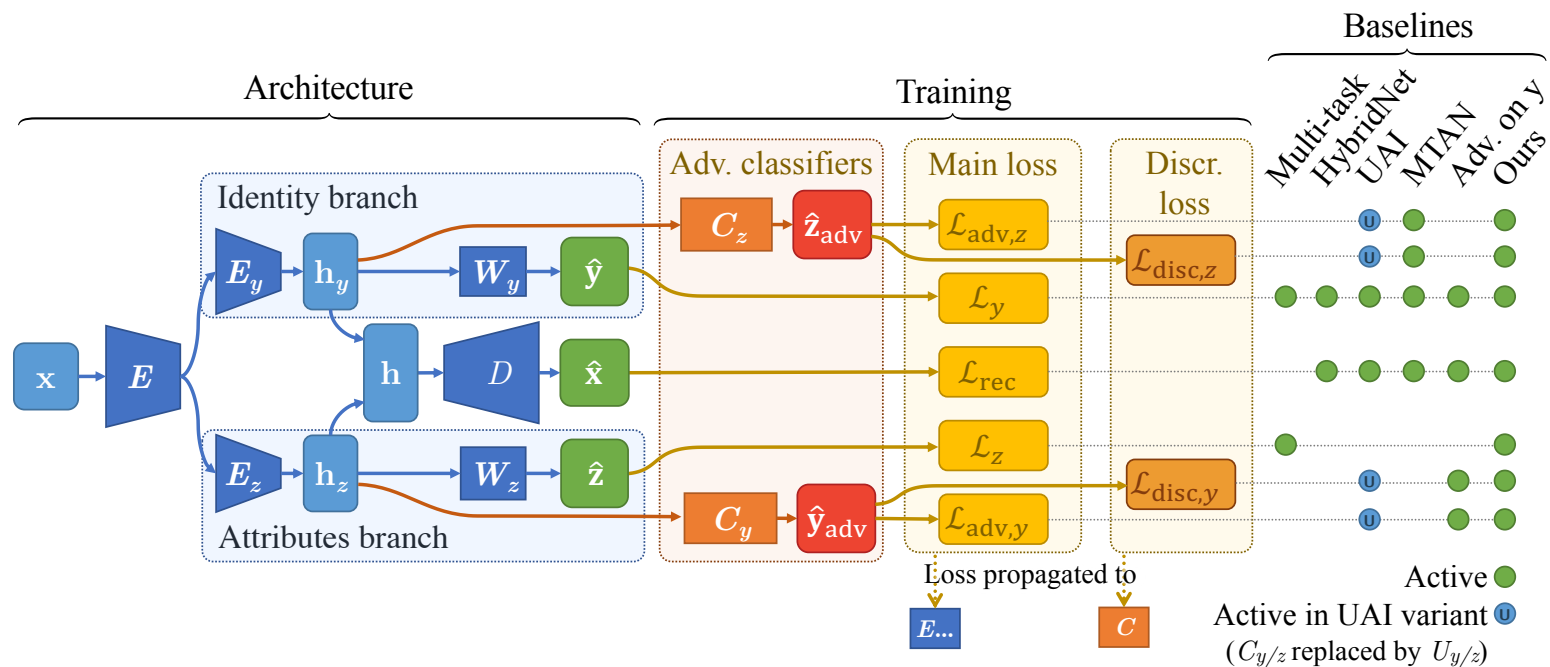


- Reconstruction Trade-off via Jacobian Supervision [Lezama, 2019]



Disentanglement (3/3)

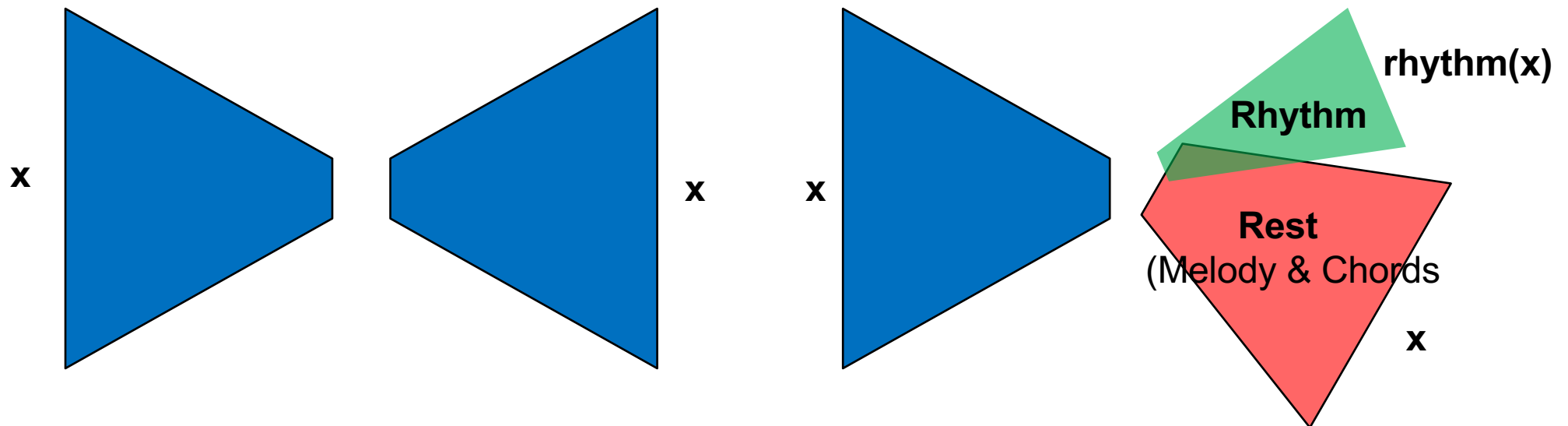
- Dual Branch Adversarial (DualDIS) [Robert et al, 2019]
- Separate Dimensions in Distinct Autoencoders (E_y and E_z)
- Measure the Presence/Absence of the Dual Dimension through a Dual (Adversarial) Classifier (C_z and C_y)
- Objective(s): **Not** Being Able to Classify Properly the Dual Dimension



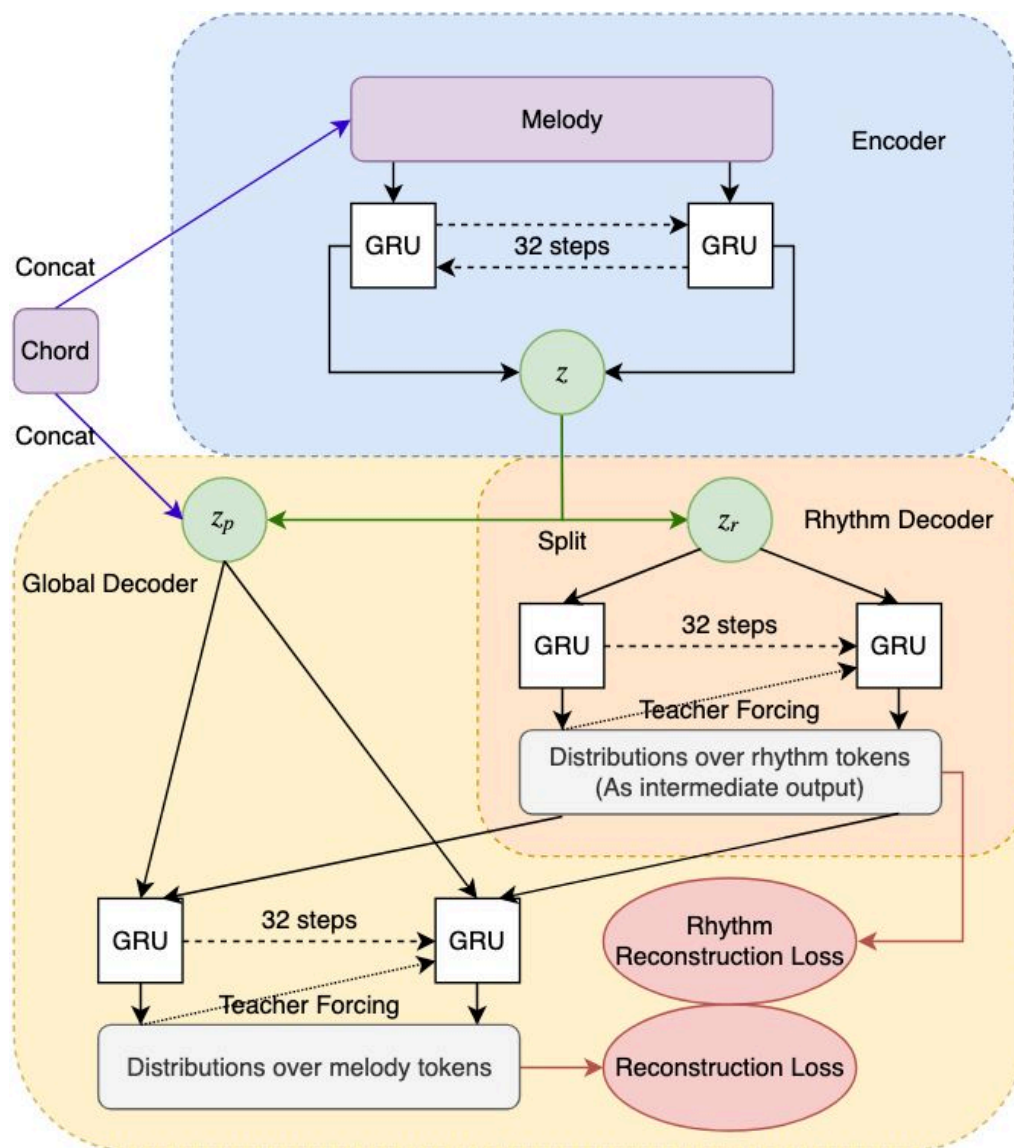
[Robert et al, 2019]

Implicit vs Explicit Dimensions (Disentanglement)

- Dimensions (ex: Pitch Range, Duration Range...) are « Chosen » by the Architecture
- But we can also Configure/Train the Architecture in order to « Force » some Dimensions



Ex: EC2-VAE [Yang et al., 2019]



Examples EC2-VAE [Yang et al., 2019]

Original Melody A



Rythm Reference B



Rest(A) + Rythm(B)



Examples EC2-VAE [Yang et al., 2019]

Original Melody B



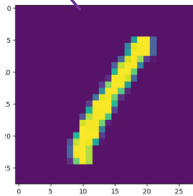
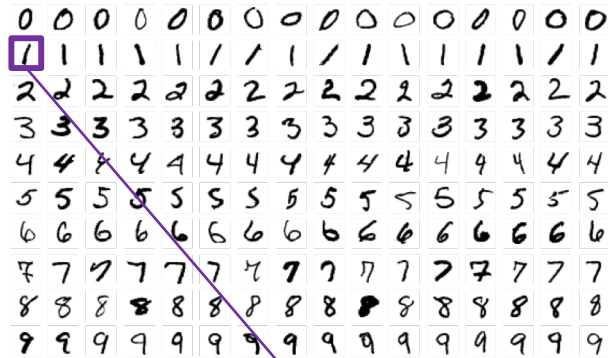
Rythm Reference C



Rest(B) + Rythm(C)



VAE Magic



How is it possible ?
Compress 784 variables into 2
and reconstruct the original ?

784

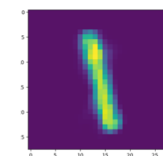
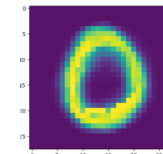
512

2

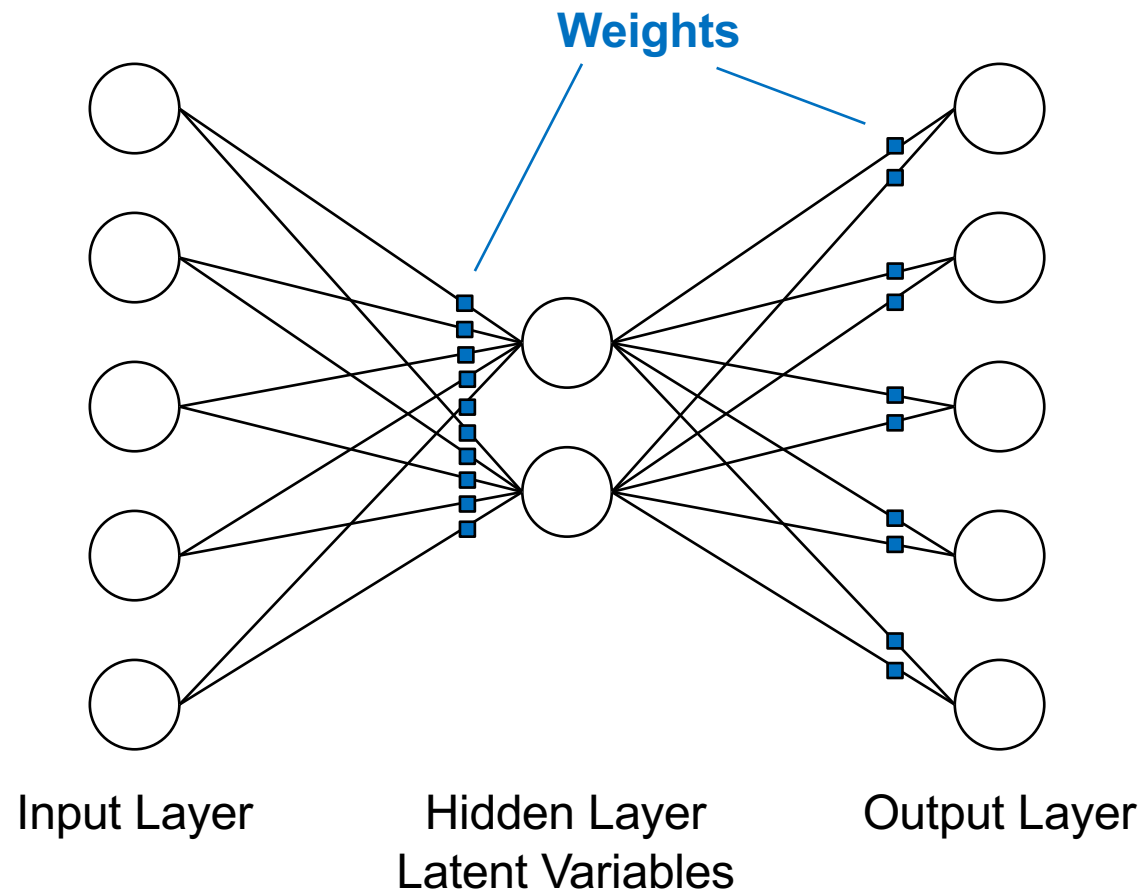
Label = (z_1, z_2)

digit1

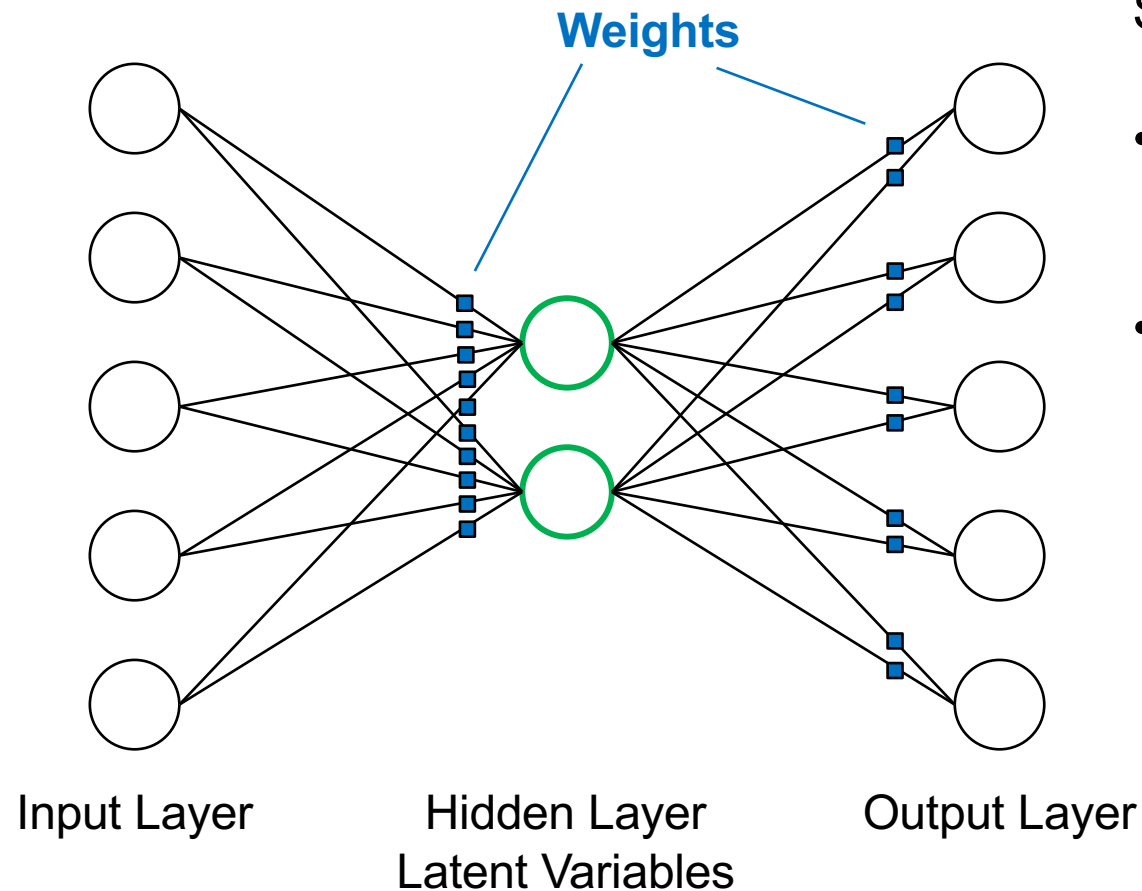
digit2



VAE Magic Revealed



VAE Magic Revealed



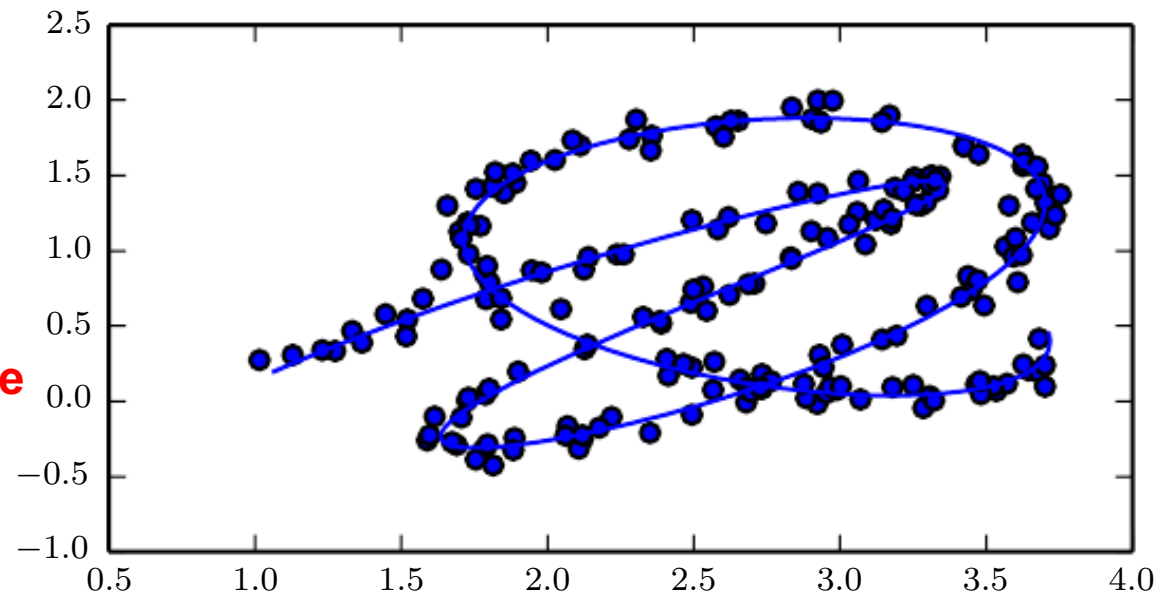
Split/Extract between

- Common Data:
Weights
- Variable/Discriminative Data:
Latent Variables

Manifold Representation

Representation/Manifold Learning

Manifold :
Set of connected points
In a high-dimensional space

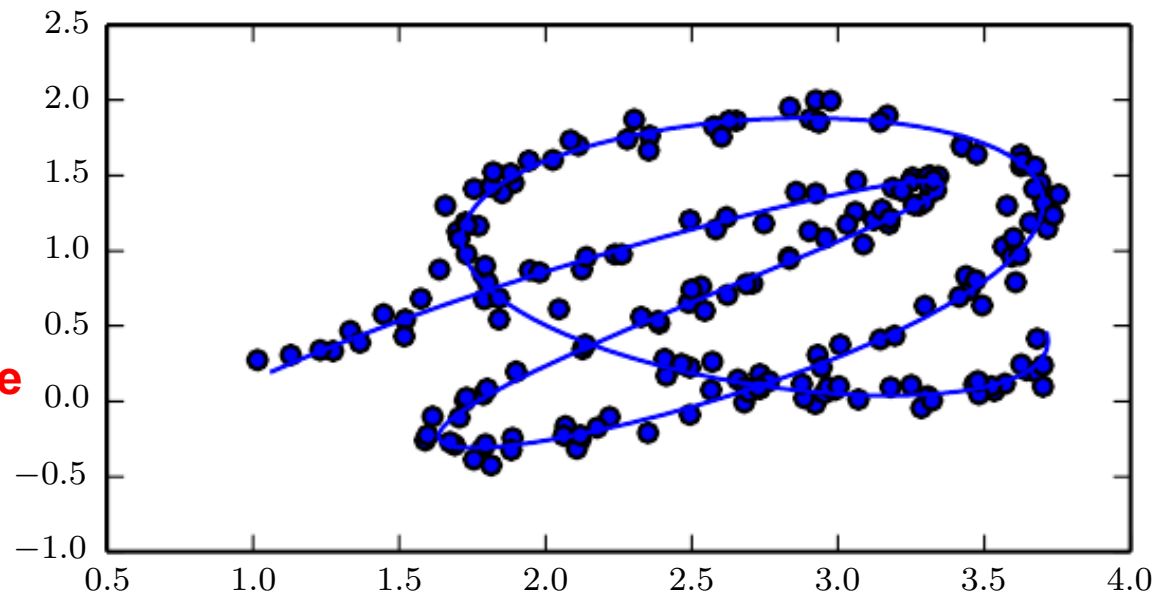


[Goodfellow et al., 2016]

Representation/Manifold Learning

Manifold :
Set of connected points

In a **high-dimensional space**

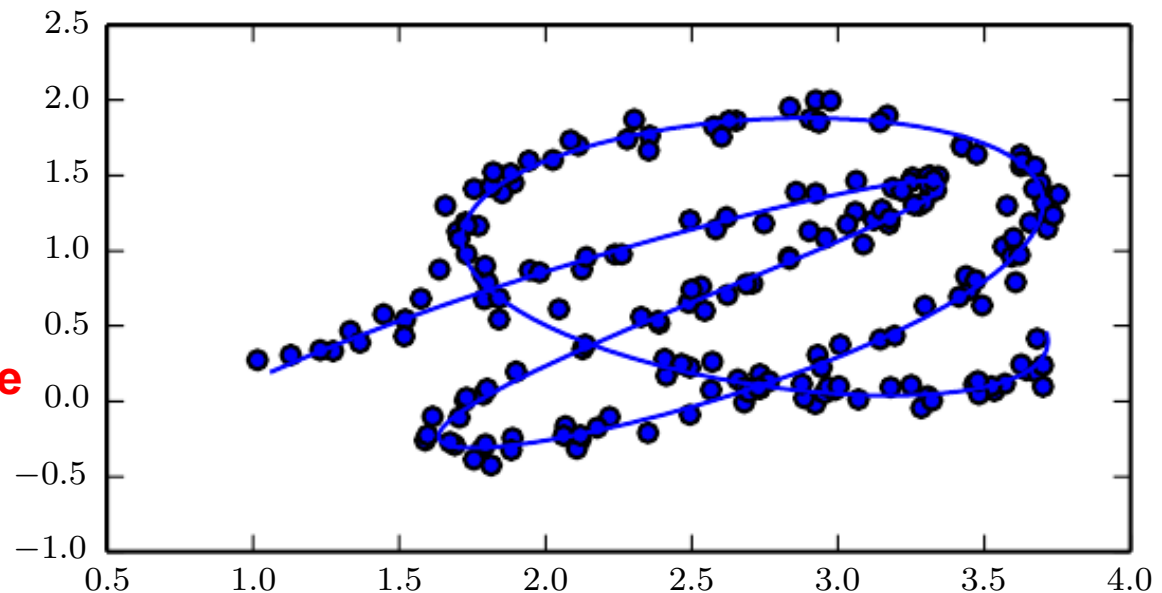


But can be approximated by
a **smaller number of dimensions**,
each dimension corresponding
to a **local variation**

Representation/Manifold Learning

Manifold :
Set of connected points

In a **high-dimensional space**

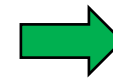


But can be approximated by
a **smaller number of dimensions**,
each dimension corresponding
to a **local variation**

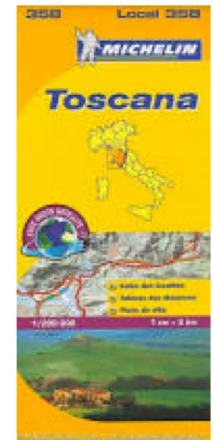
Analogy:



3D Earth



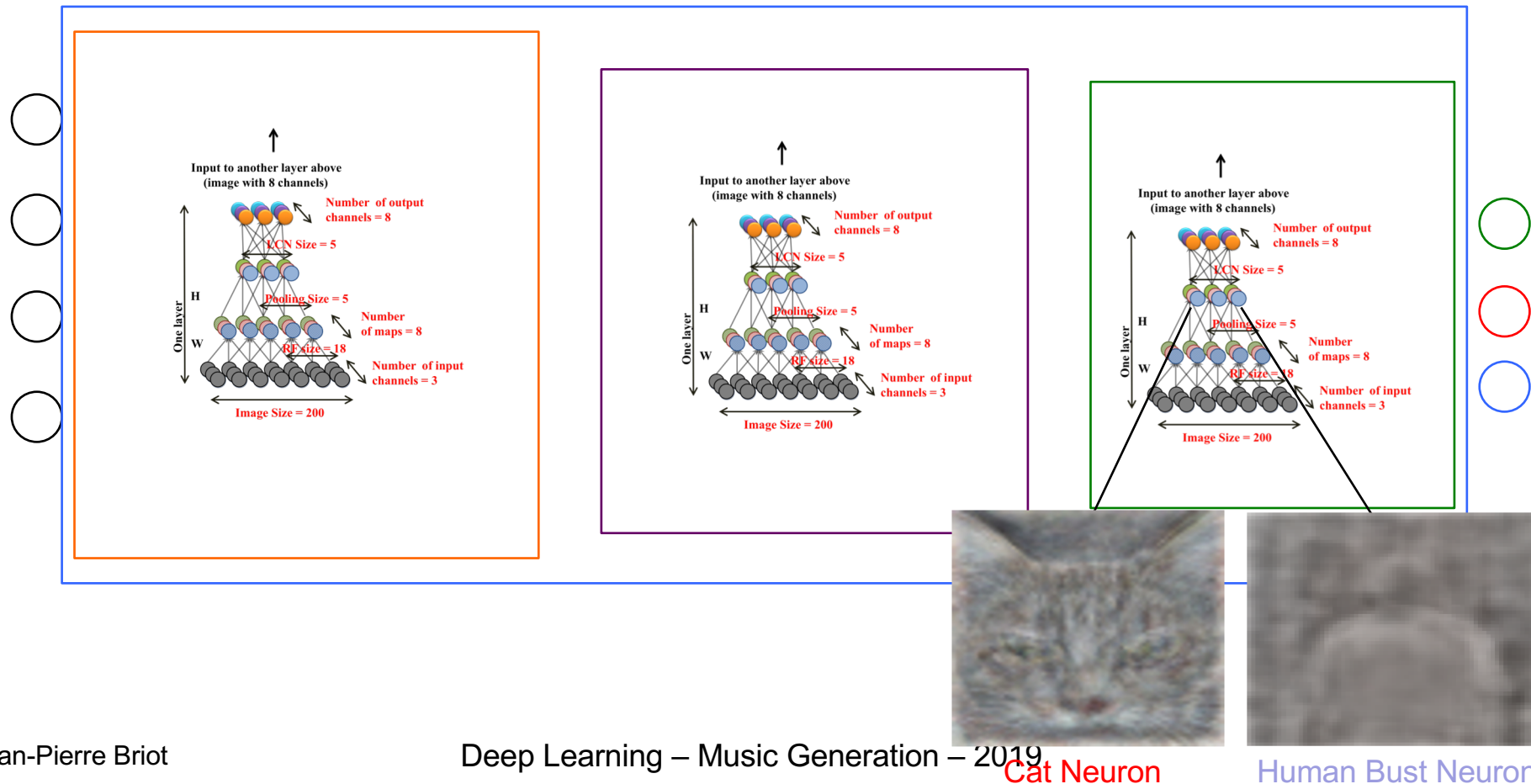
2D Map



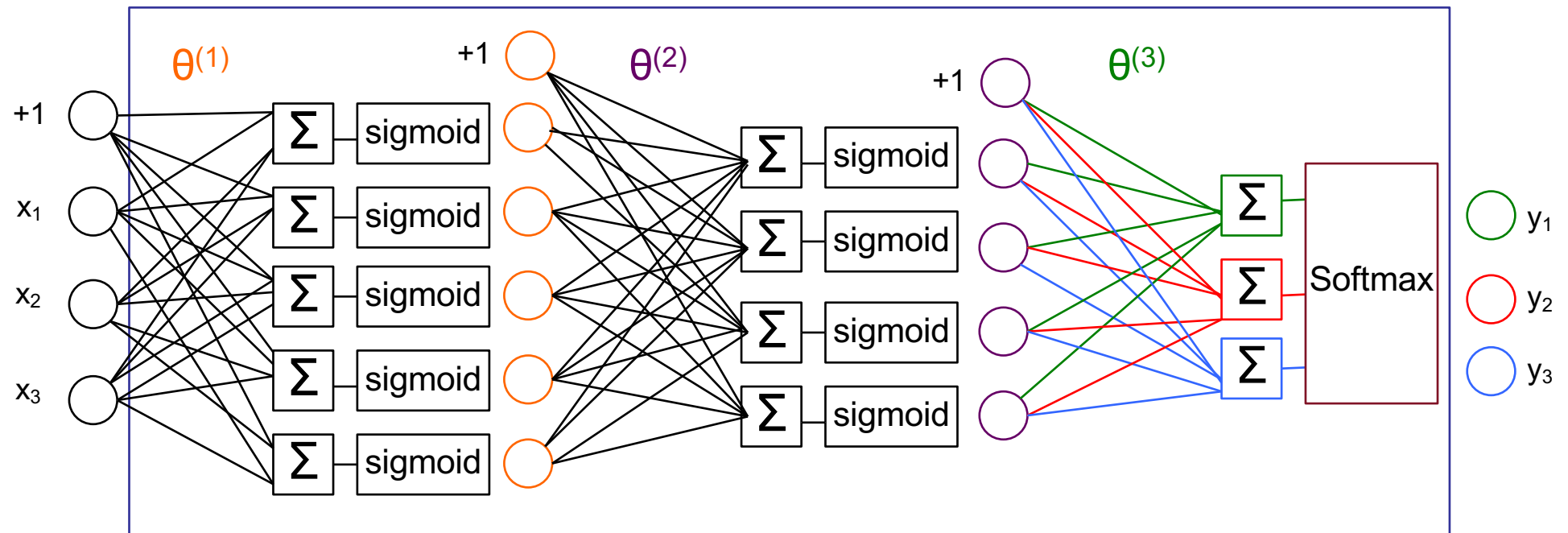
Pre-Training Deep Networks

Deep Unsupervised Learning

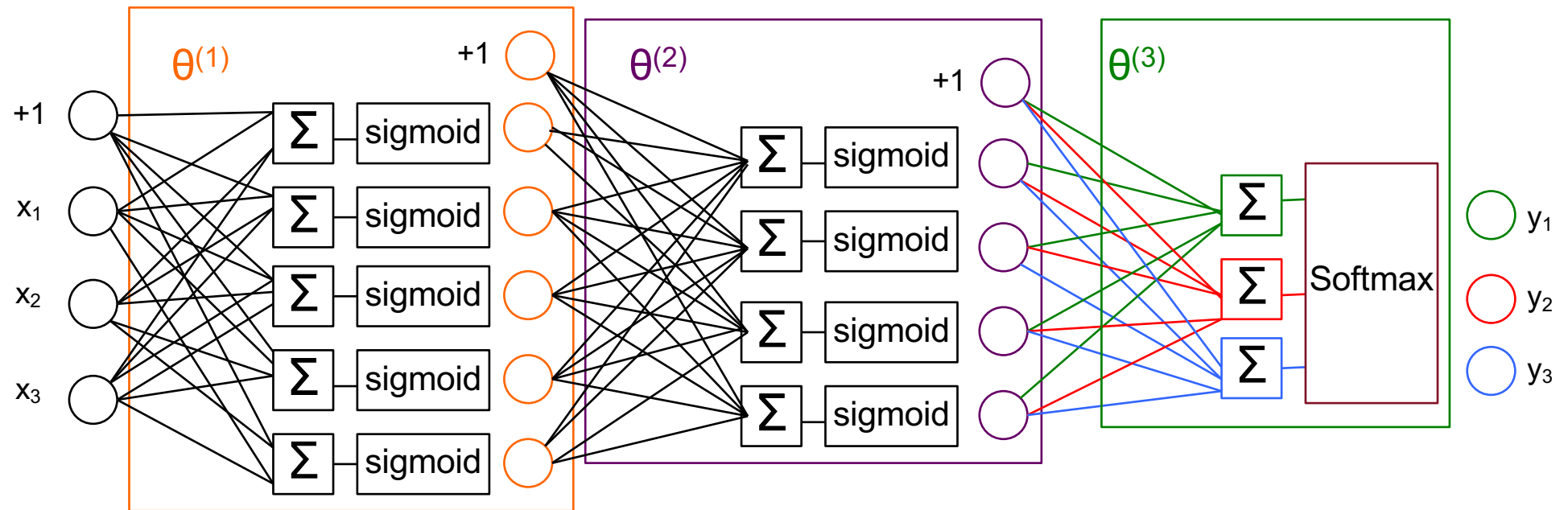
- Deep Learning may also be Applied to **Unsupervised** Learning
- Deep Brain Architecture (3 Layers of Autoencoders) [Le et al. 2012]
- (Purely) Unsupervised Learning (on Images extracted from YouTube videos)
- Discovery of Cats and Human Bust Patterns (Classifier/Neuron)



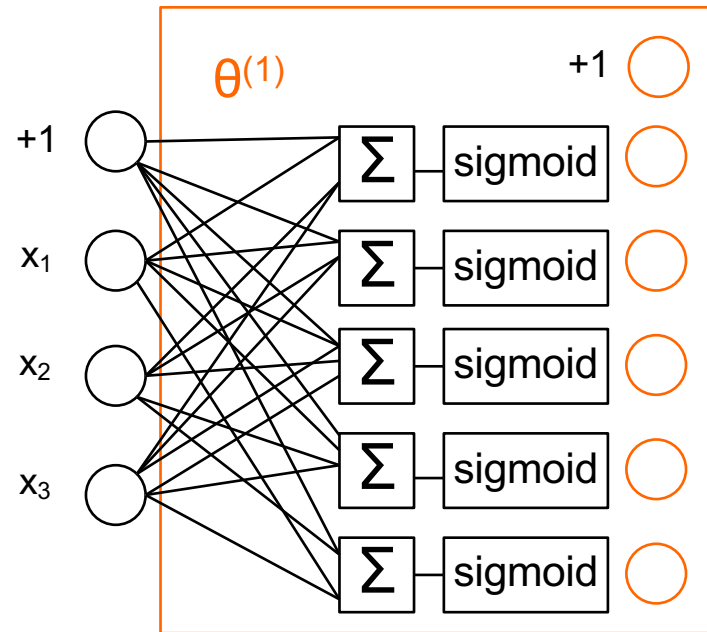
From Multilayer/Deep Networks...



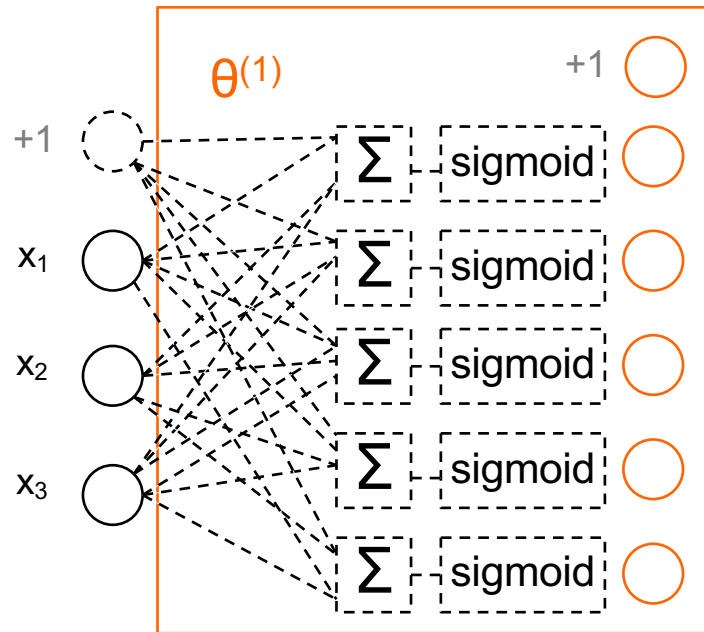
... to Pre-Trained Deep Networks [Hinton 2006]



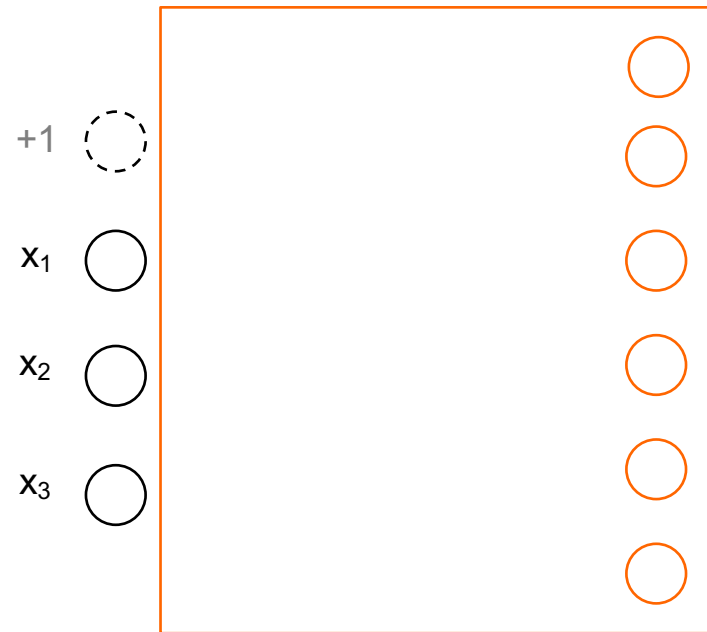
1st Hidden Layer



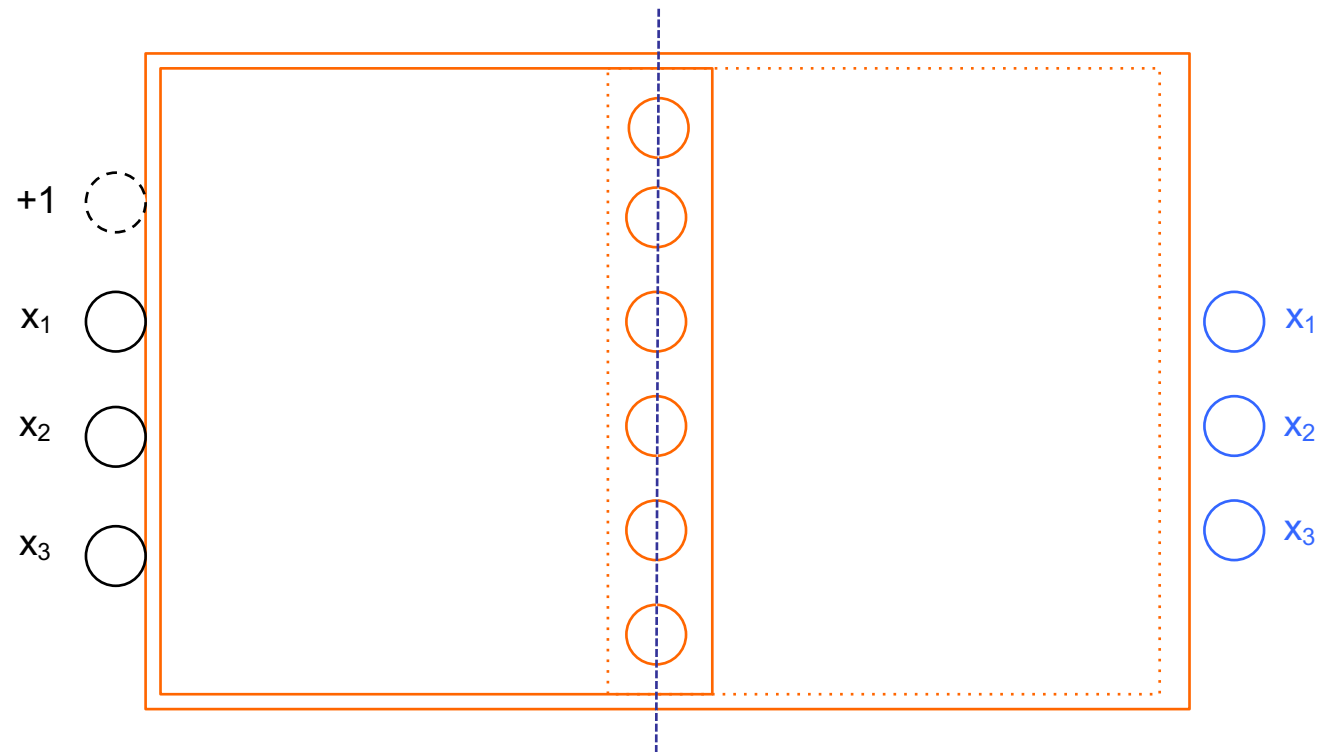
1st Hidden Layer



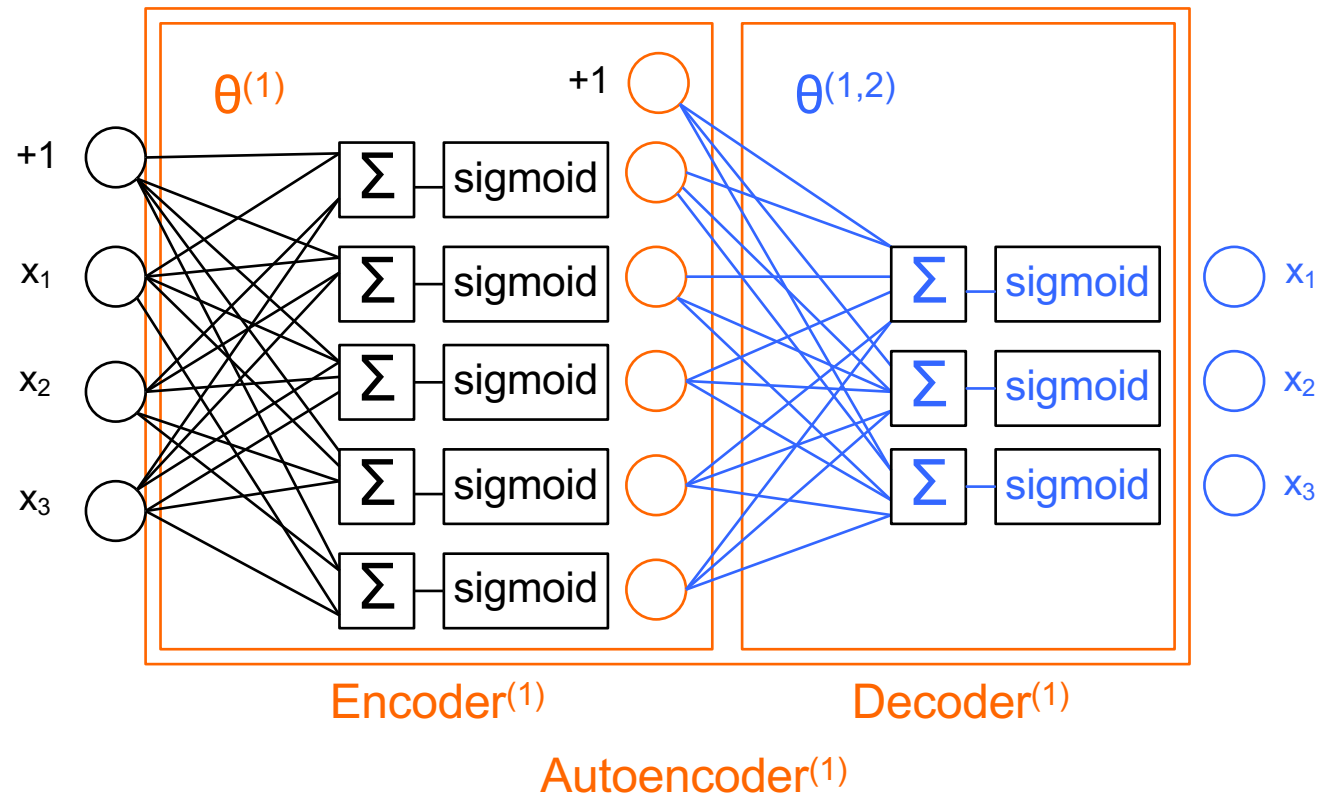
1st Hidden Layer



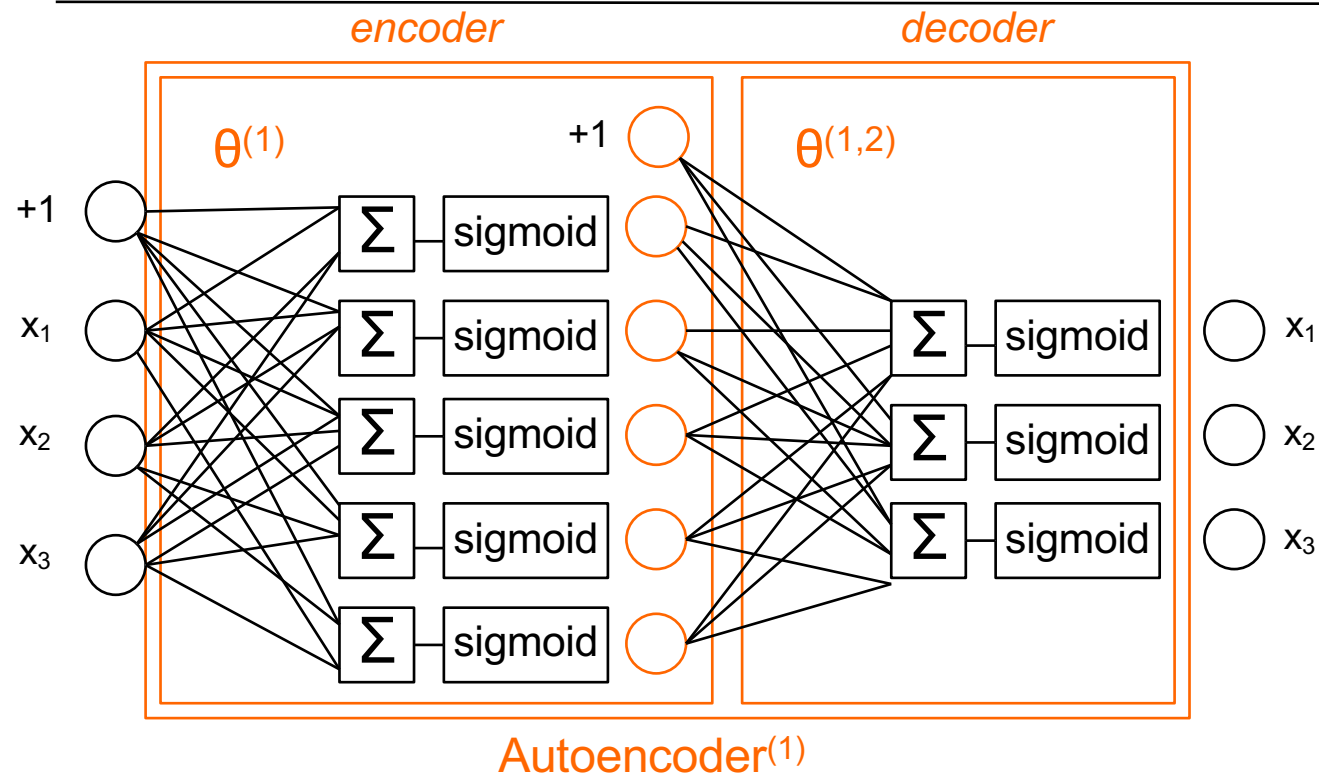
Mirroring Inputs into Outputs



Autoencoder

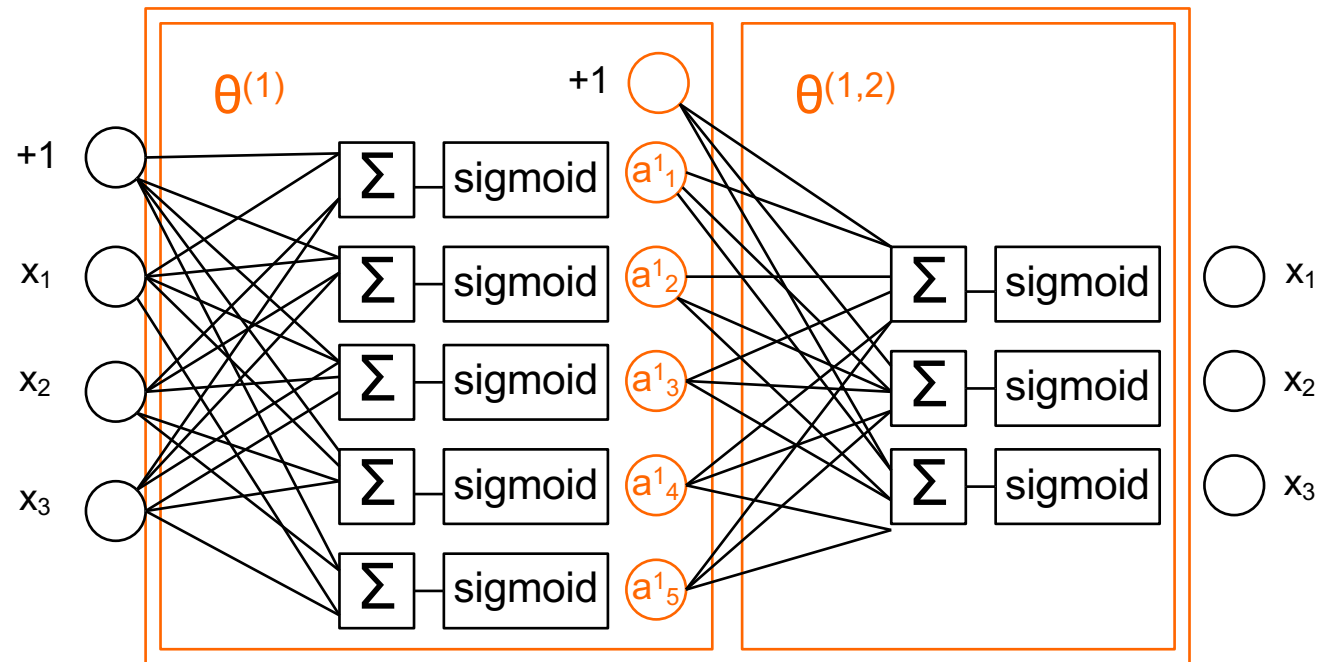


Autoencoder Self-Supervised Training



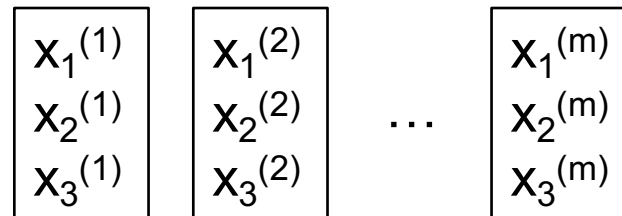
- Training (finding $\theta^{(1)}$ and $\theta^{(1,2)}$) on Input Dataset : X : $\begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix} \quad \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \end{bmatrix} \quad \dots \quad \begin{bmatrix} x_1^m \\ x_2^m \\ x_3^m \end{bmatrix}$
- **Self-Supervised Training** implemented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

1st Sparse Autoencoder Production

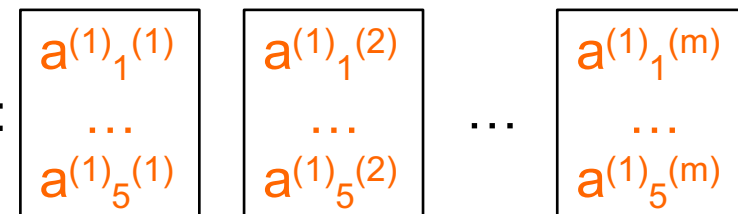


Autoencoder⁽¹⁾

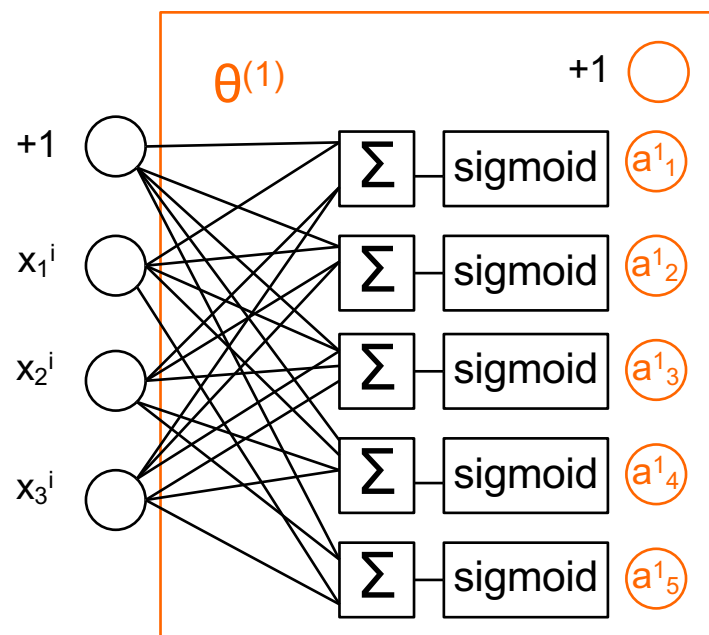
- Feedforward on Set of Inputs X :



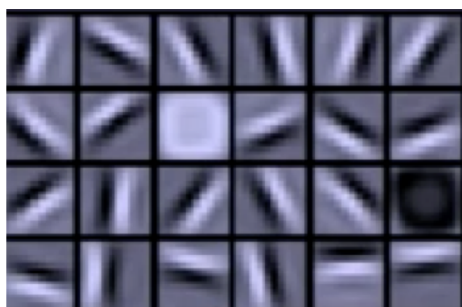
- Produces a Set of Internal States $A^1(X)$:



1st Sparse Autoencoder Finalization



Autoencoder⁽¹⁾

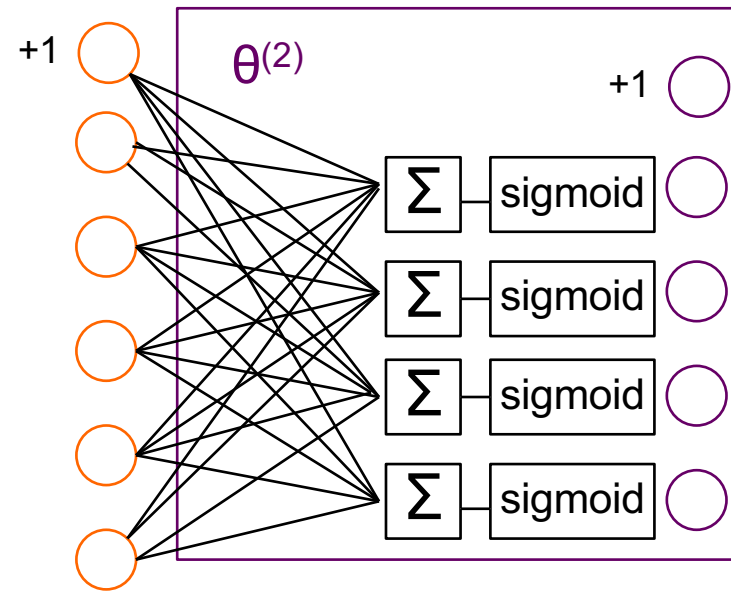


Learnt features⁽¹⁾

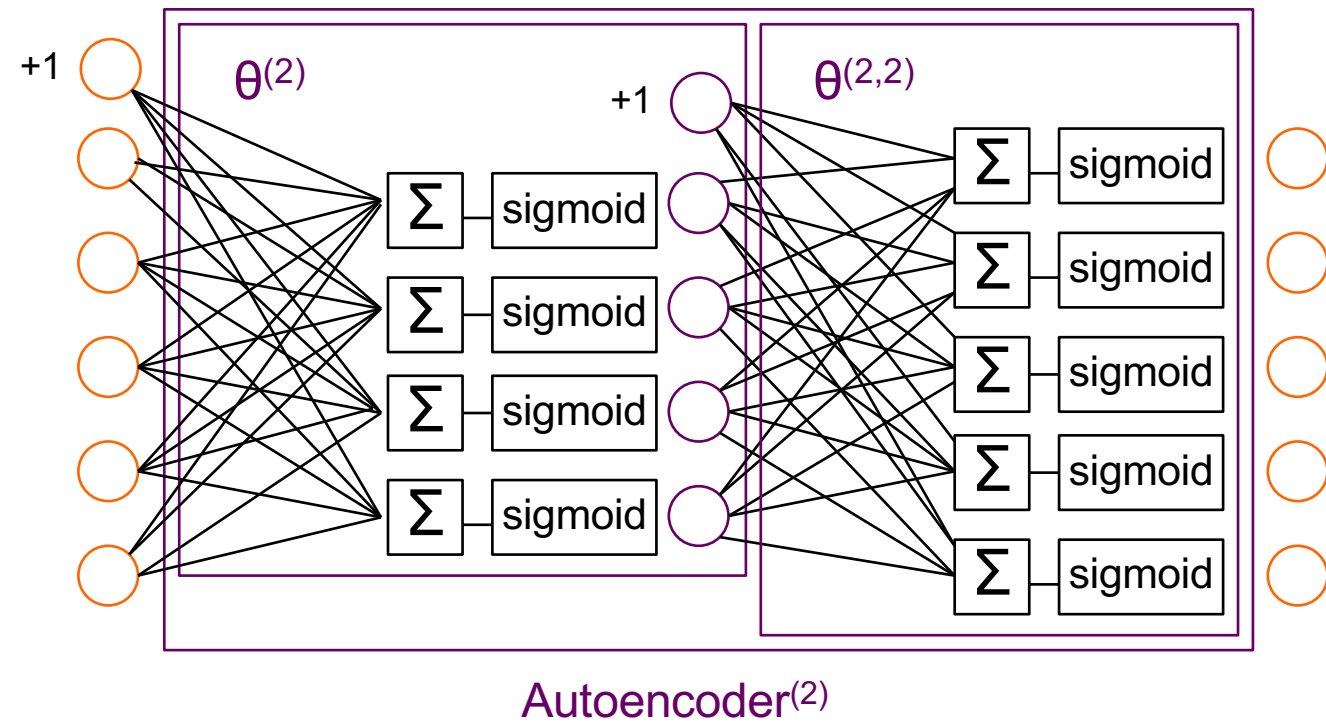
Note:

- 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)

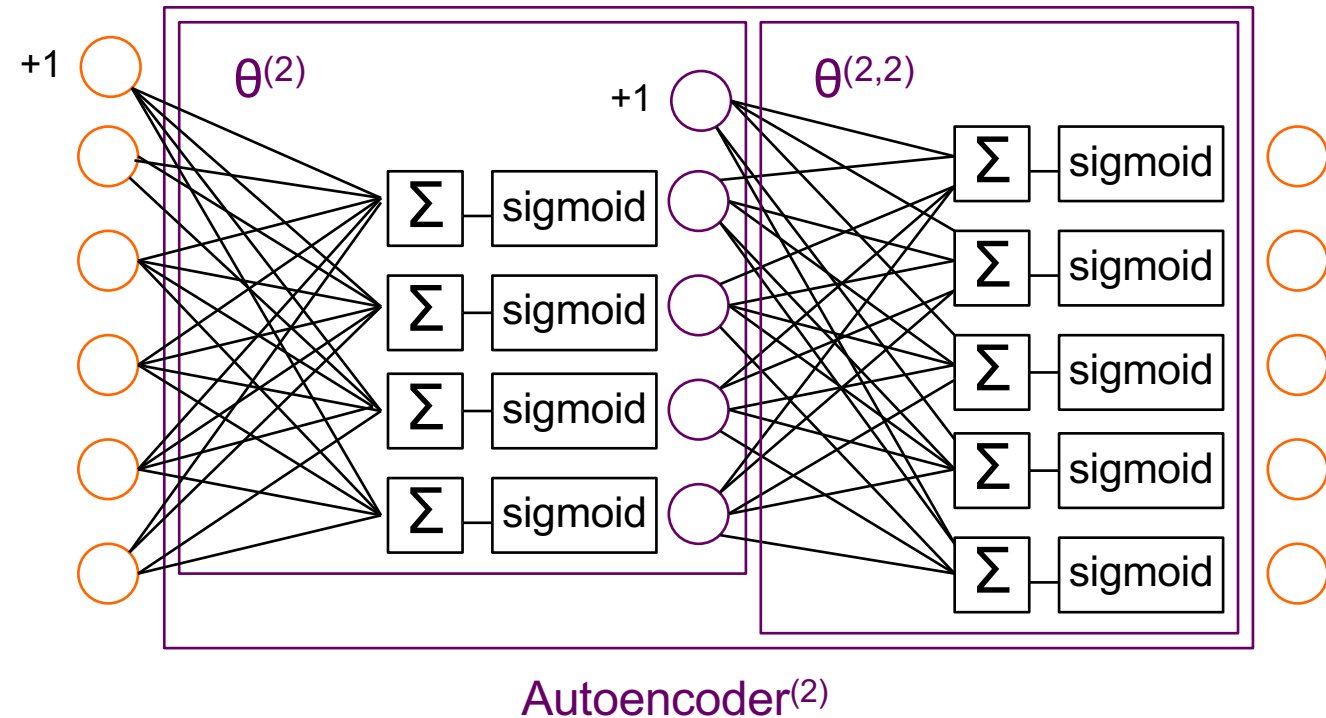
2nd Hidden Layer



2nd Sparse Autoencoder



2nd Sparse Autoencoder Training



- Training ($\theta^{(2)}$ and $\theta^{(2,2)}$) on Input Dataset : $A^1(X)$:

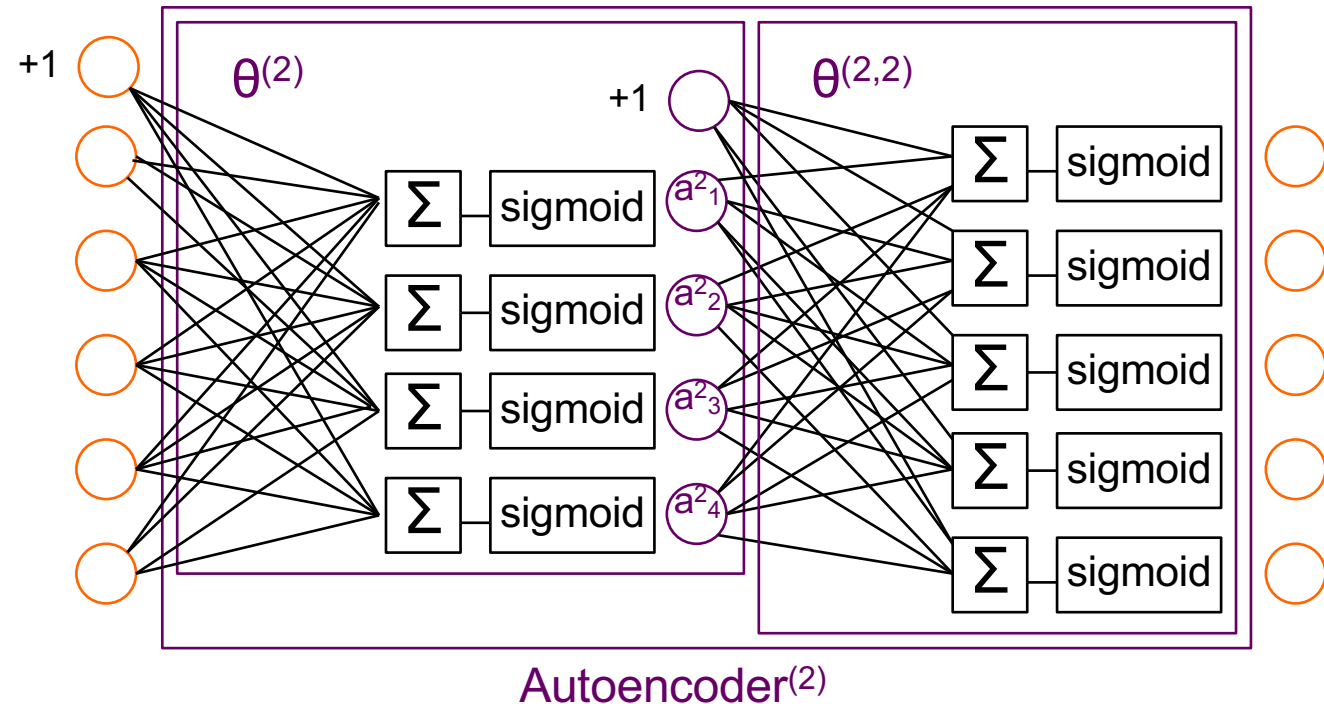
$a^{(1)}_1(1)$
...
$a^{(1)}_5(1)$

$a^{(1)}_1(2)$
...
$a^{(1)}_5(2)$

 ...

$a^{(1)}_1(m)$
...
$a^{(1)}_5(m)$

2nd Sparse Autoencoder Production



- Feedforward on Input Dataset : $A^1(X)$:

$a^{(1)}_1(1)$
...
$a^{(1)}_5(1)$

$a^{(1)}_1(2)$
...
$a^{(1)}_5(2)$

 ...

$a^{(1)}_1(m)$
...
$a^{(1)}_5(m)$
- Produces a Set of Internal States $A^2(A^1(X))$:

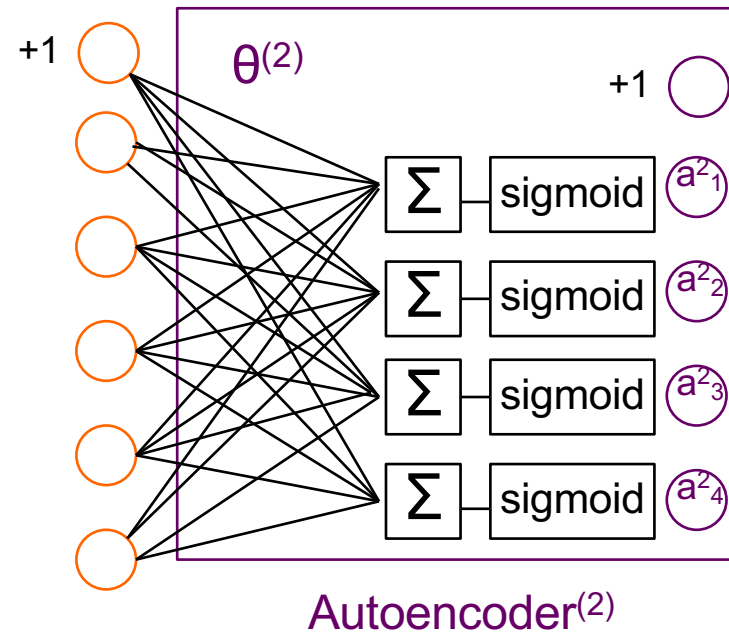
$a^{(2)}_1(1)$
...
$a^{(2)}_4(1)$

$a^{(2)}_1(2)$
...
$a^{(2)}_4(2)$

 ...

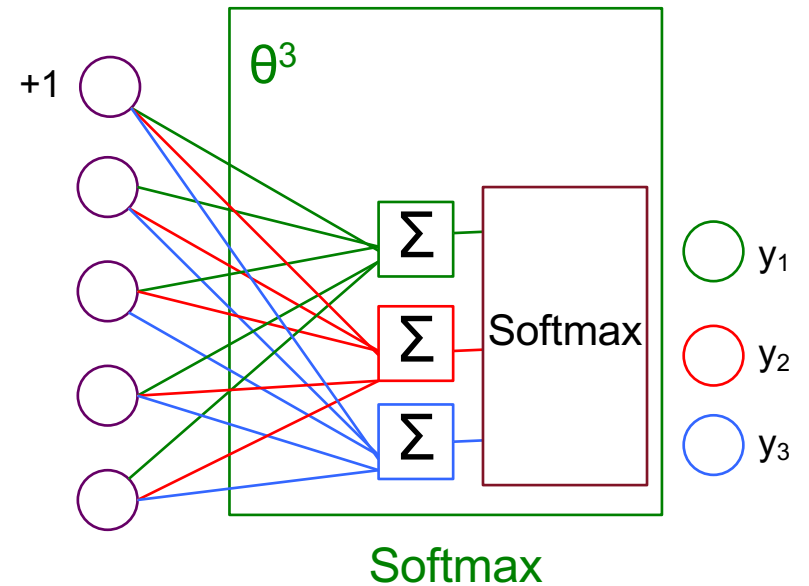
$a^{(2)}_1(m)$
...
$a^{(2)}_4(m)$

2nd Sparse Autoencoder Finalization



Learnt features⁽²⁾

(Final) Softmax Layer



- (Supervised) Training on Input Dataset : $A^2(A^1(X))$:

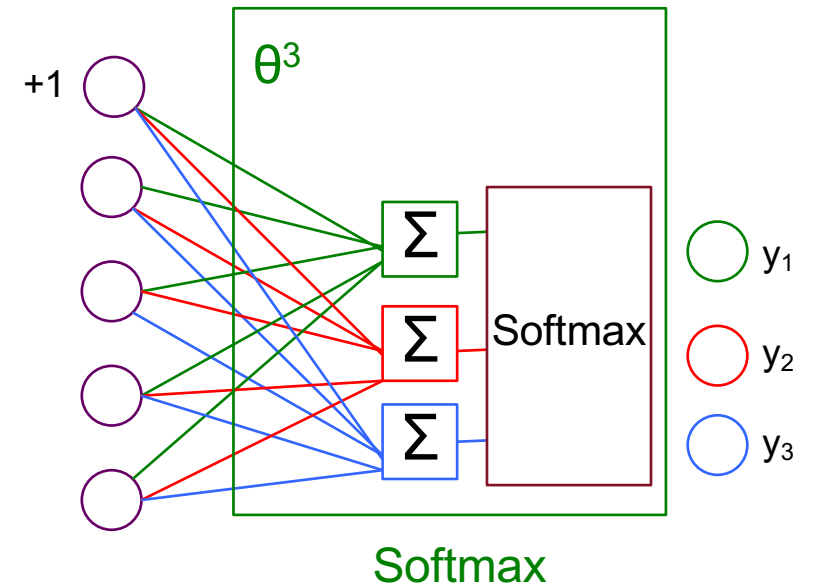
$a^{(2)}_1(1)$
...
$a^{(2)}_4(1)$

$a^{(2)}_1(2)$
...
$a^{(2)}_4(2)$

 ...

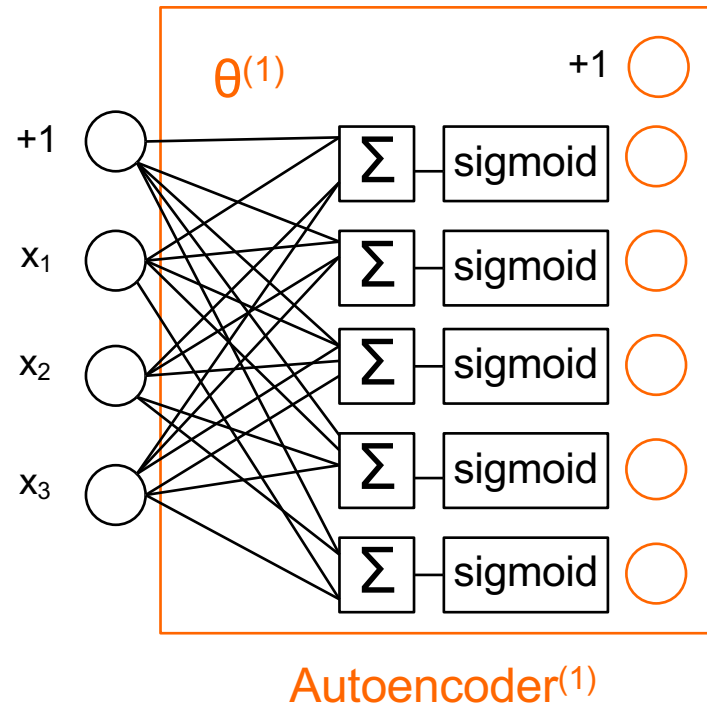
$a^{(2)}_1(m)$
...
$a^{(2)}_4(m)$
- and Output Dataset : y :
- | |
|-------------|
| $y_1^{(1)}$ |
| $y_2^{(1)}$ |
| $y_3^{(1)}$ |
- | |
|-------------|
| $y_1^{(2)}$ |
| $y_2^{(2)}$ |
| $y_3^{(2)}$ |
- ...
- | |
|-------------|
| $y_1^{(m)}$ |
| $y_2^{(m)}$ |
| $y_3^{(m)}$ |

(Final) Softmax Layer Finalization

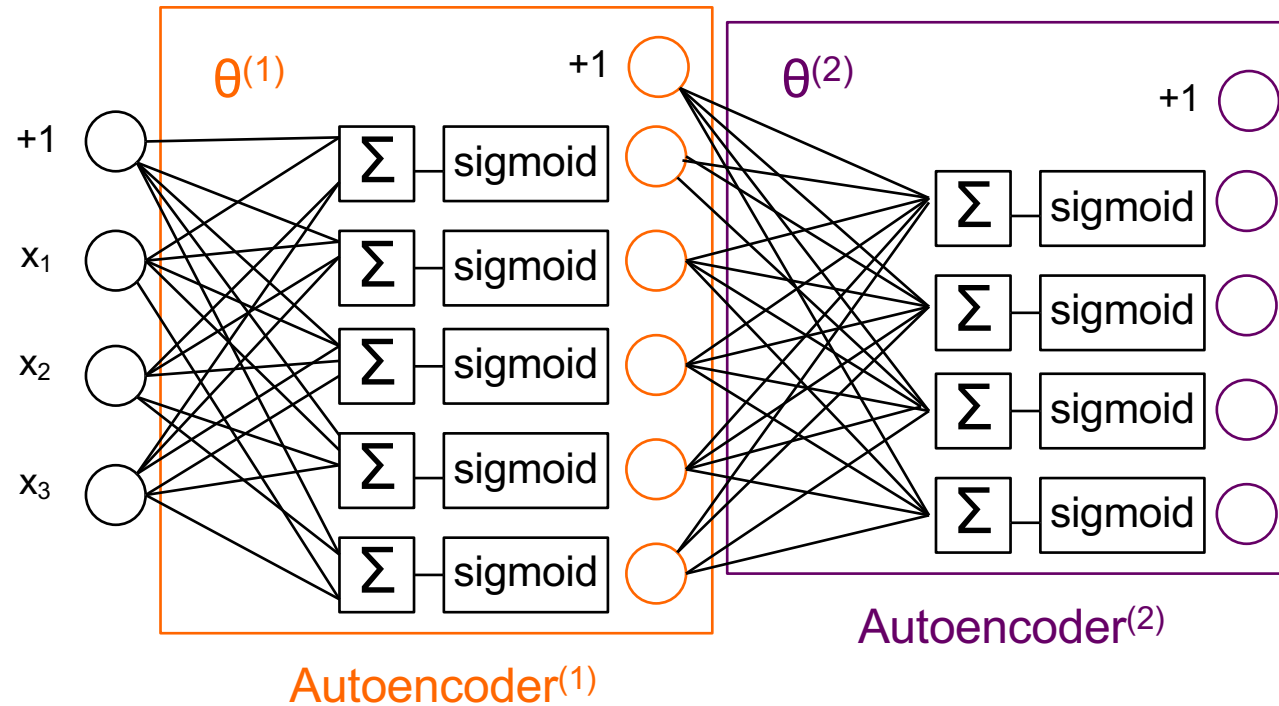


Learnt features⁽³⁾

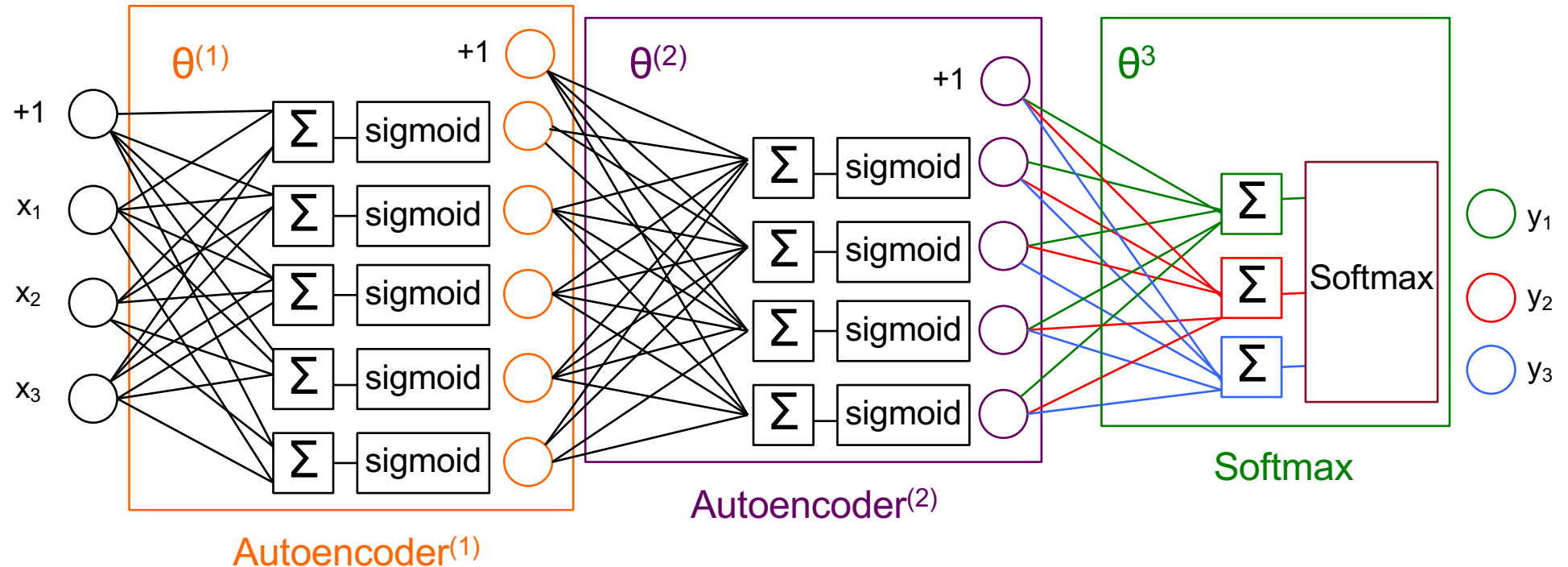
Assembling the Pre-Trained Deep Network



Assembling the Pre-Trained Deep Network

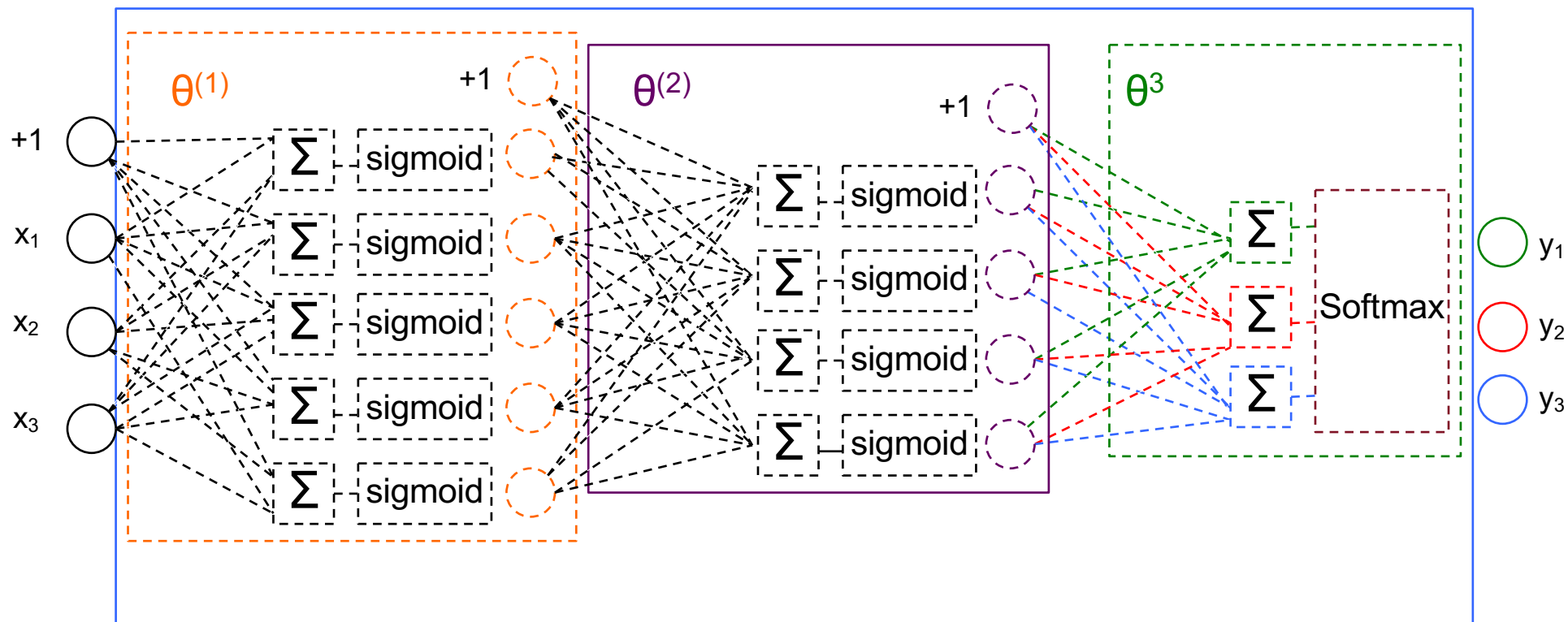


Assembling the Pre-Trained Deep Network Functional Composition

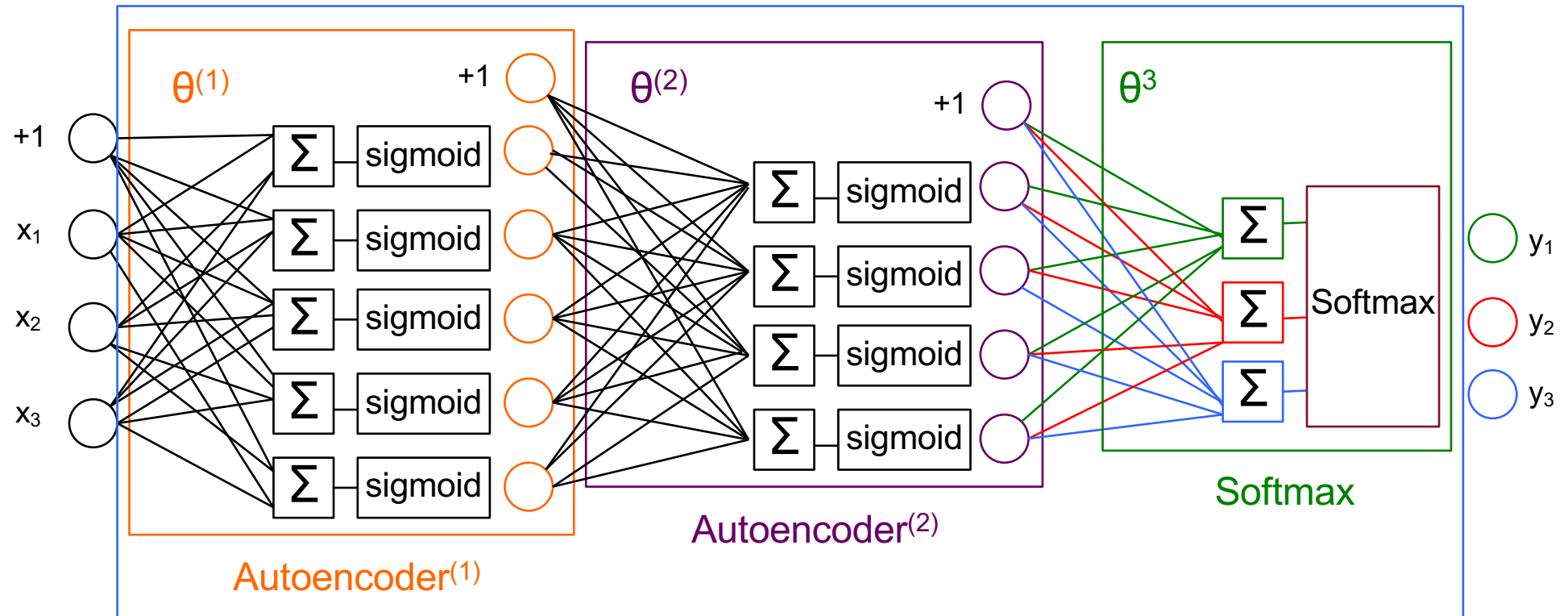


Assembling the Pre-Trained Deep Network

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 :

x_1^1
x_2^1
x_3^1

x_1^2
x_2^2
x_3^2

 ...

x_1^m
x_2^m
x_3^m
- and Output Dataset : y :

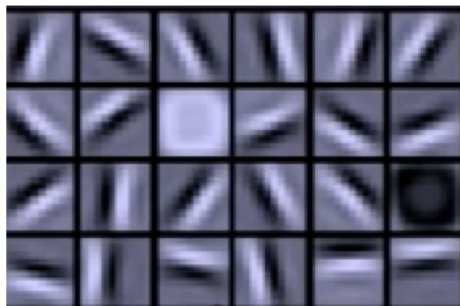
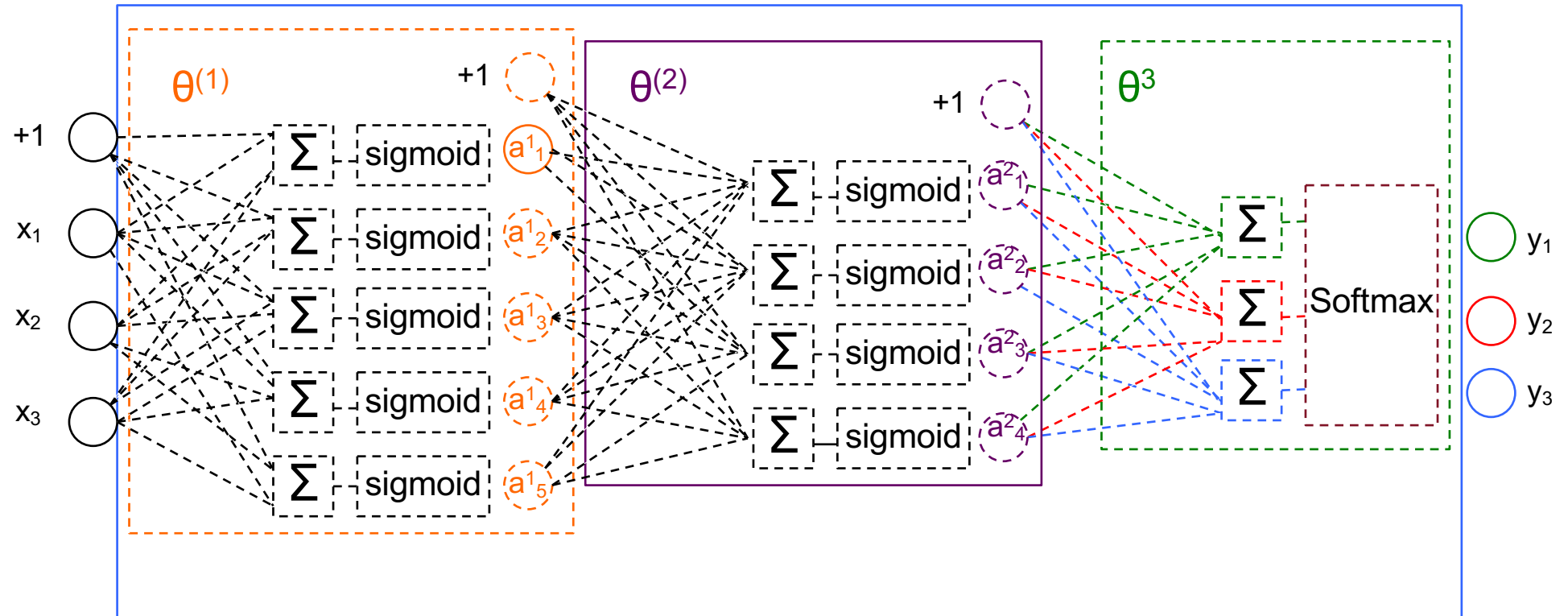
y_1^1
y_2^1
y_3^1

y_1^2
y_2^2
y_3^2

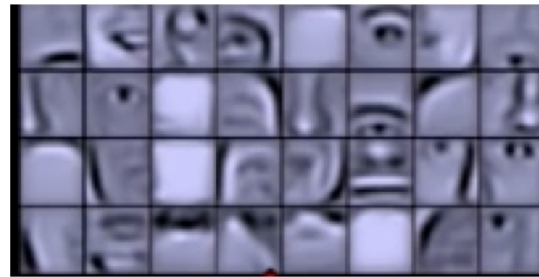
 ...

y_1^m
y_2^m
y_3^m

Successive Features/Abstractions Constructed



Jean-Pierre Briot

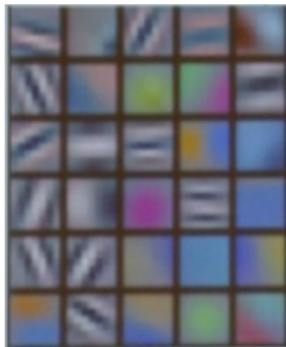
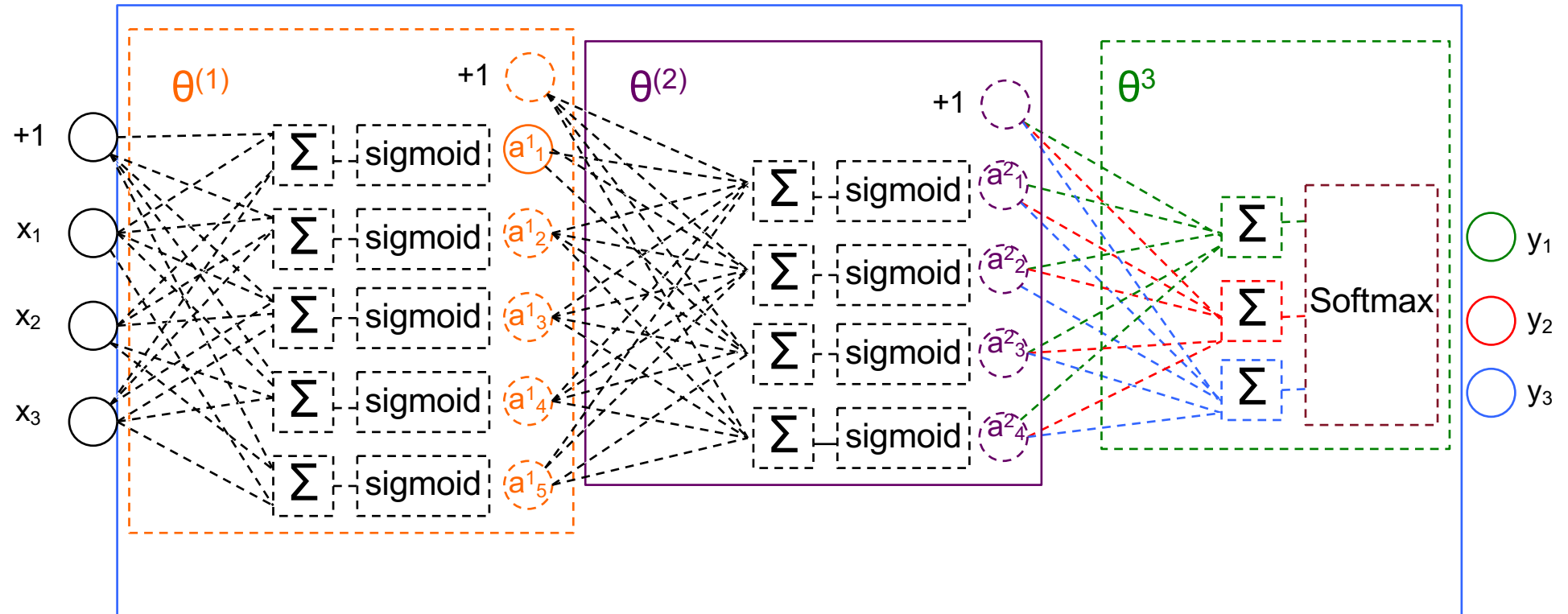


Deep Learning – Music Generation – 2019

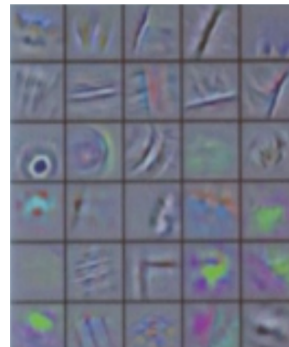


[Ng 2013]

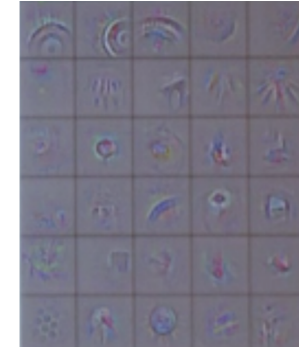
Successive Features/Abstractions Constructed



Jean-Pierre Briot



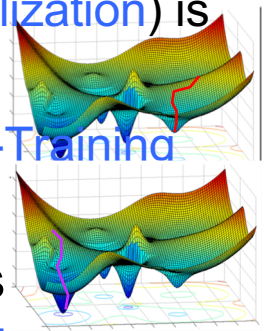
Deep Learning – Music Generation – 2019



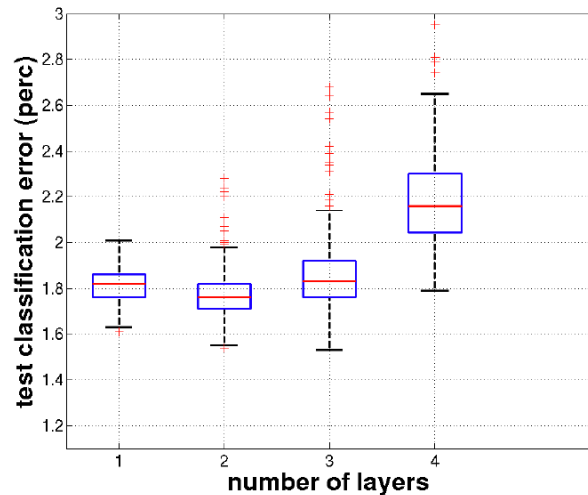
[Zeiler and Fergus 2013] 80

Summary – Key Ideas

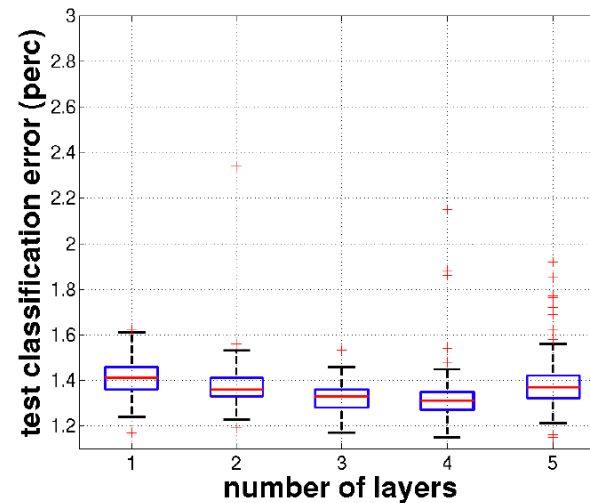
1. Automatic Construction (Cascade Learning) of **Hierarchical Abstractions/Features**, which are **Exposed** and **Useable** (and **Used**, see 2.)
 - In Standard Multilayer Neural Networks, Features are **Hidden** (**Black Box**)
2. Accurate **Initialization** of **Weights Matrixes** for each Layer
 - In Standard Multilayer Neural Networks, **Initialization** of Weights Matrixes of each Layer is **Heuristic**, and is **NOT** Based on **Information** (Training Data/Examples)
 - In Pre-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]

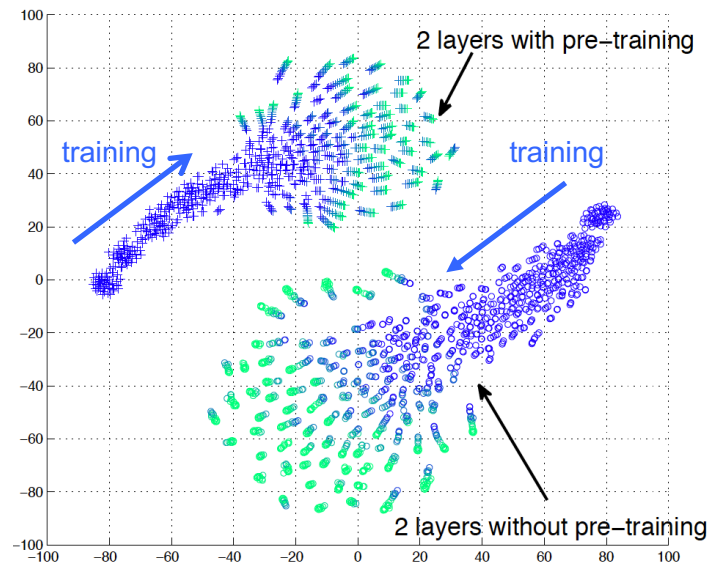


Without pre-training

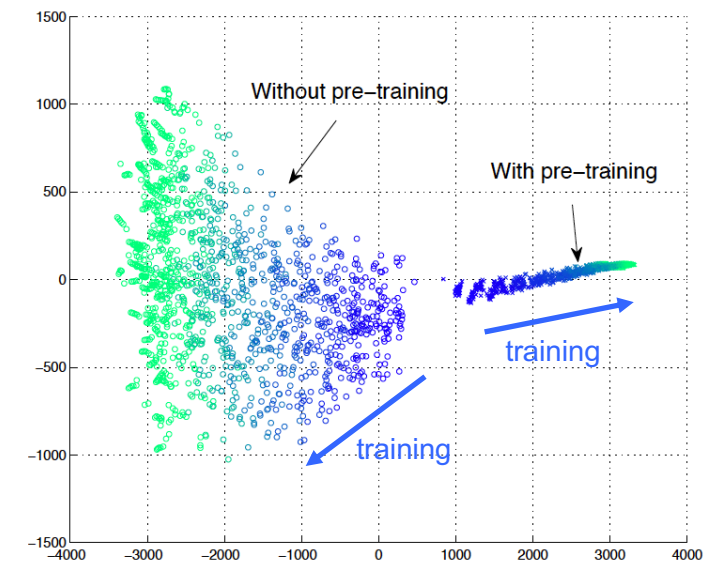


With pre-training

Training/Evolution of Hypotheses



Local structure-focused density reduction (tNSE)



Global structure-focused density reduction (ISOMAP)

Gains

- (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 $x \rightarrow x+y$ and not $x \rightarrow y$

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

