Energy-Based GANs & other Adversarial things

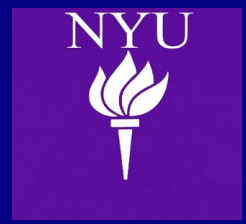
Yann Le Cun

Facebook AI Research

Center for Data Science, NYU

Courant Institute of Mathematical Sciences, NYU

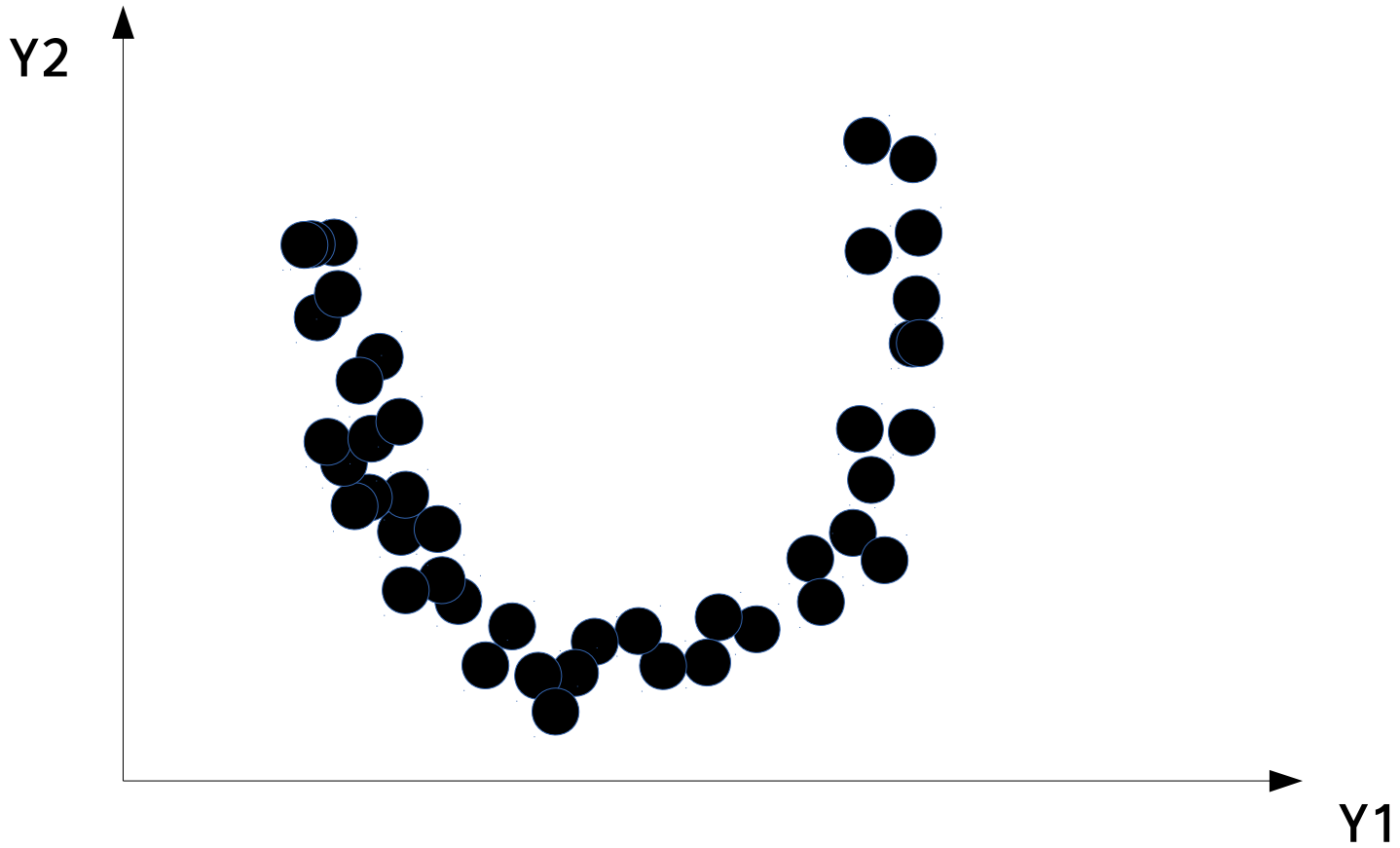
http://yann.lecun.com




Unsupervised Learning

Energy-Based Unsupervised Learning

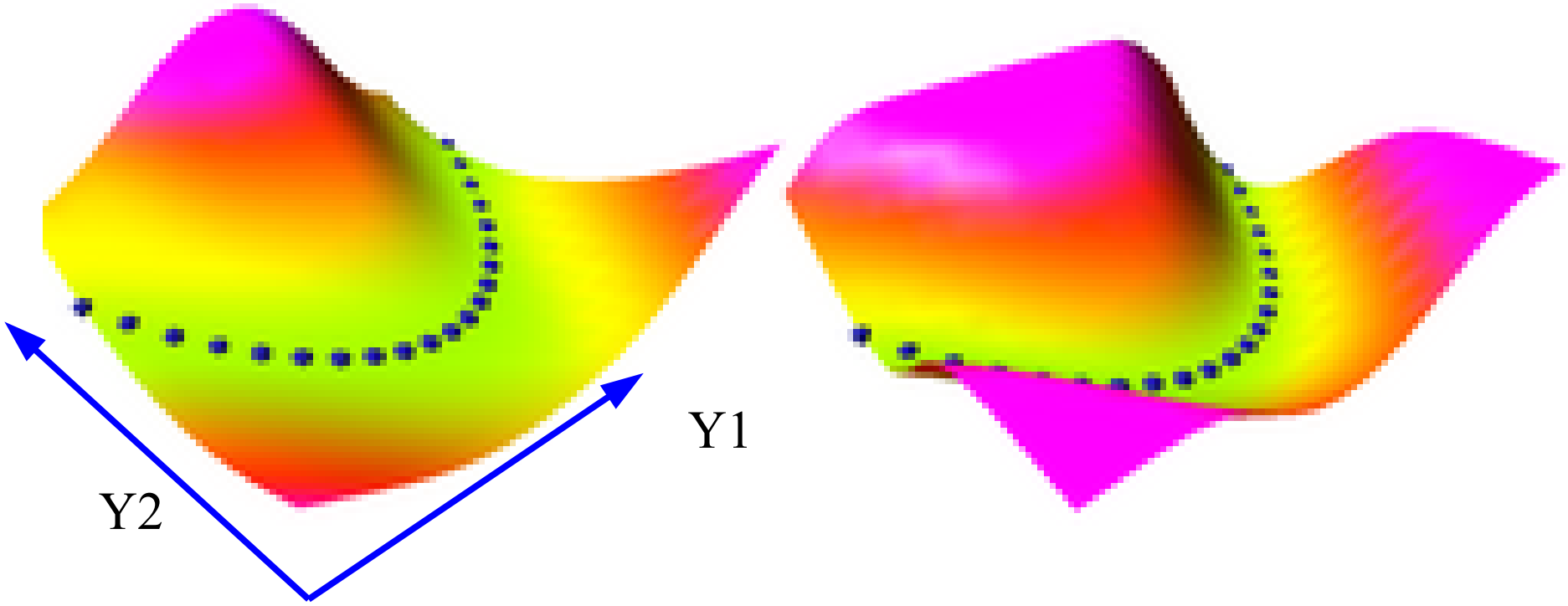
- Learning an energy function (or contrast function) that takes
  - Low values on the data manifold
  - Higher values everywhere else



Capturing Dependencies Between Variables with an Energy Function

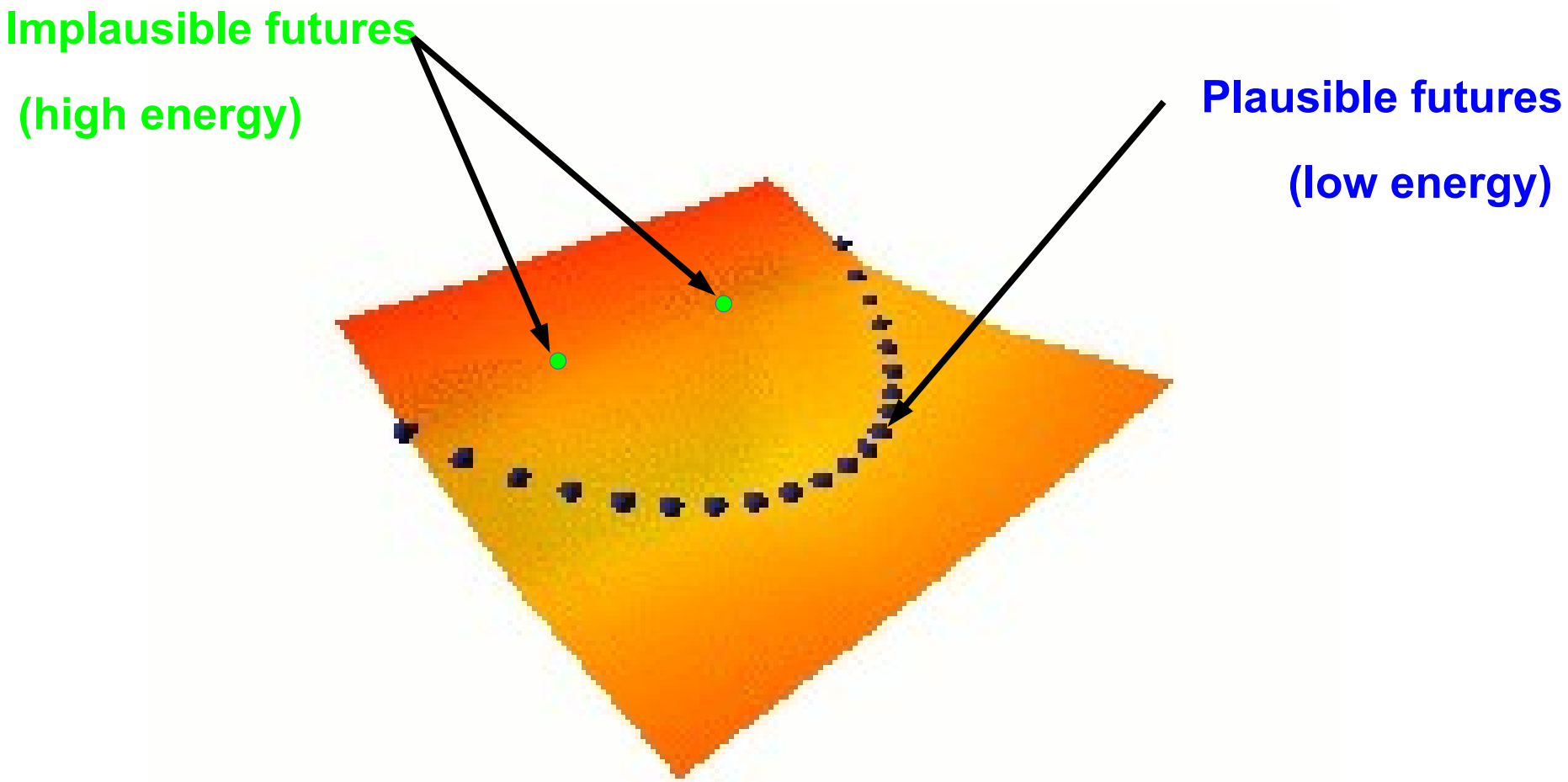
- The energy surface is a "contrast function" that takes low values on the data manifold, and higher values everywhere else
  - Special case: energy = negative log density
  - Example: the samples live in the manifold

$$Y_2 = (Y_1)^2$$



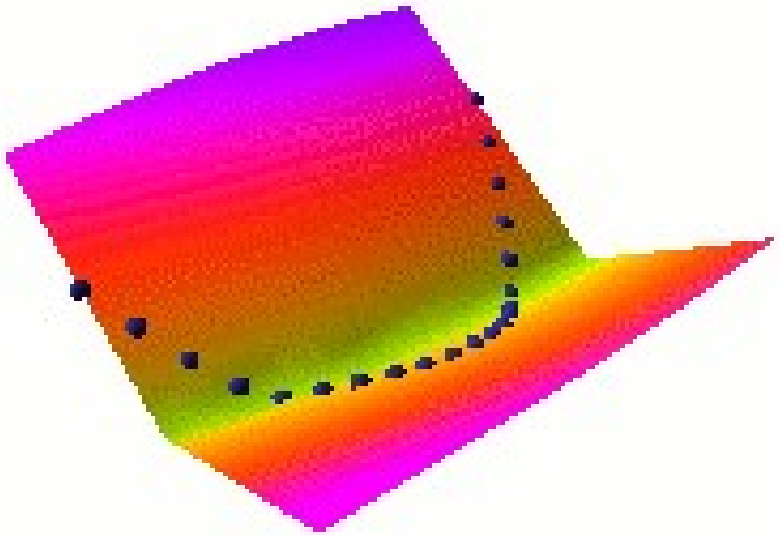
Energy-Based Unsupervised Learning

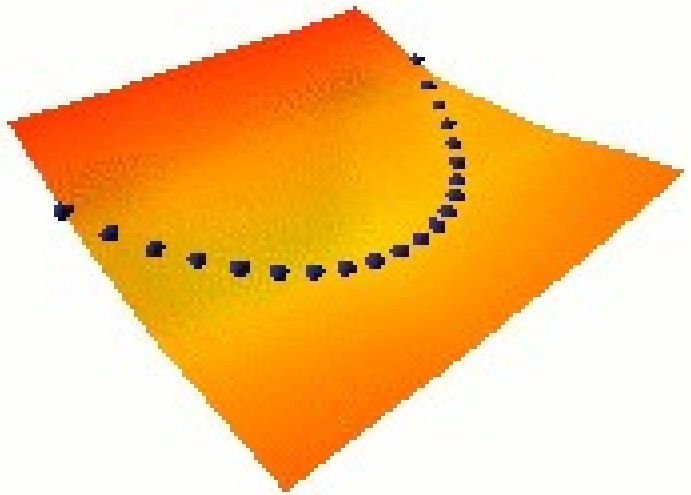
- **Energy Function:** Takes low value on data manifold, higher values everywhere else
- Push down on the energy of desired outputs. Push up on everything else.
- But how do we choose where to push up?



Learning the Energy Function

- parameterized energy function E(Y,W)
  - Make the energy low on the samples
  - Make the energy higher everywhere else
  - Making the energy low on the samples is easy
  - But how do we make it higher everywhere else?





Seven Strategies to Shape the Energy Function

- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA
- 2. push down of the energy of data points, push up everywhere else
  - Max likelihood (needs tractable partition function)
- 3. push down of the energy of data points, push up on chosen locations
  - contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- 4. minimize the gradient and maximize the curvature around data points
  - score matching
- 5. train a dynamical system so that the dynamics goes to the manifold
  - denoising auto-encoder
- 6. use a regularizer that limits the volume of space that has low energy
  - Sparse coding, sparse auto-encoder, PSD
- 7. if E(Y) = ||Y - G(Y)||^2, make G(Y) as "constant" as possible.
  - Contracting auto-encoder, saturating auto-encoder

#1: constant volume of low energy
Energy surface for PCA and K-means

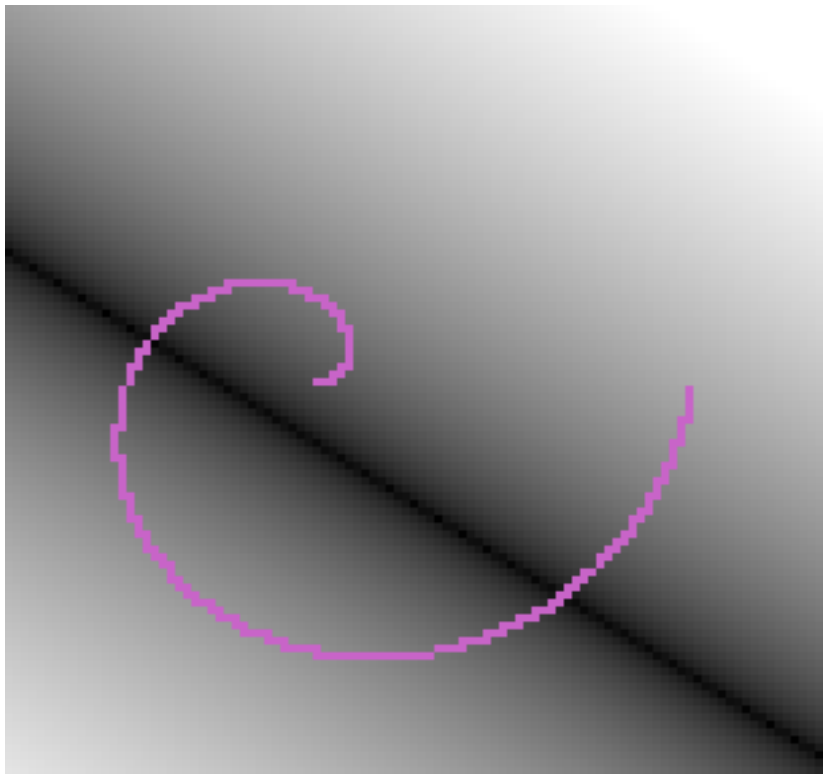
- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA...

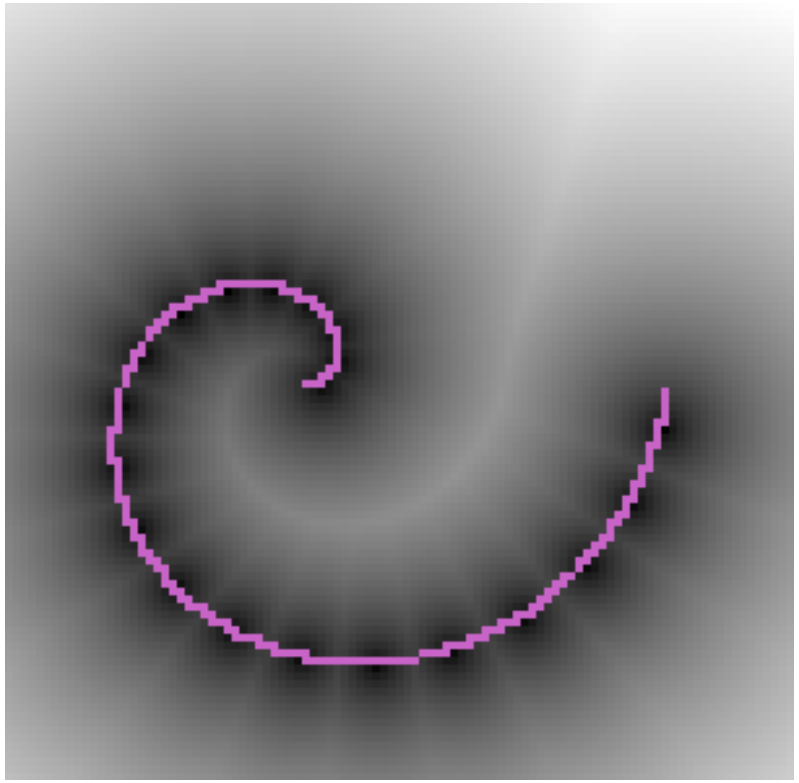
PCA

$$E(Y) = \|W^T WY - Y\|^2$$

K-Means,
Z constrained to 1-of-K code

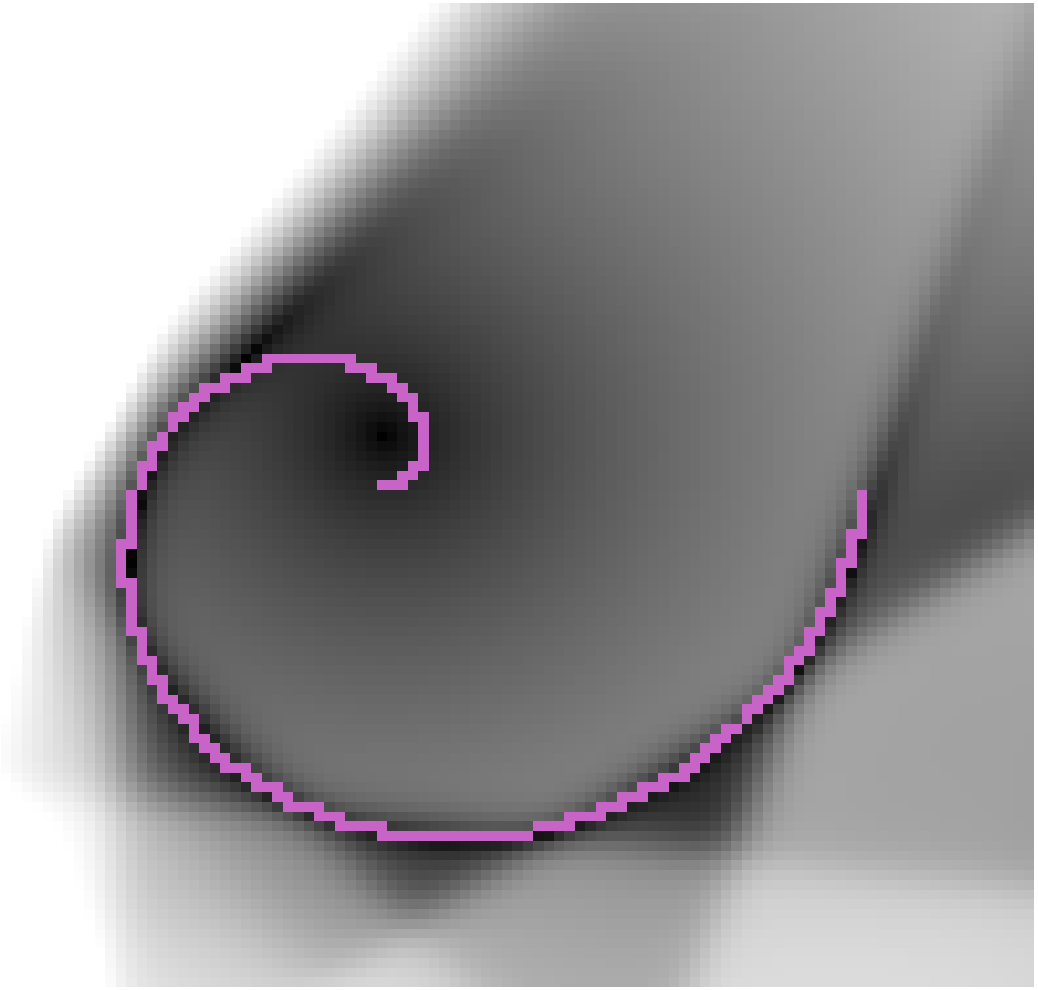
$$E(Y) = min_z \sum_i \|Y - W_i Z_i\|^2$$





#6. use a regularizer that limits the volume of space that has low energy

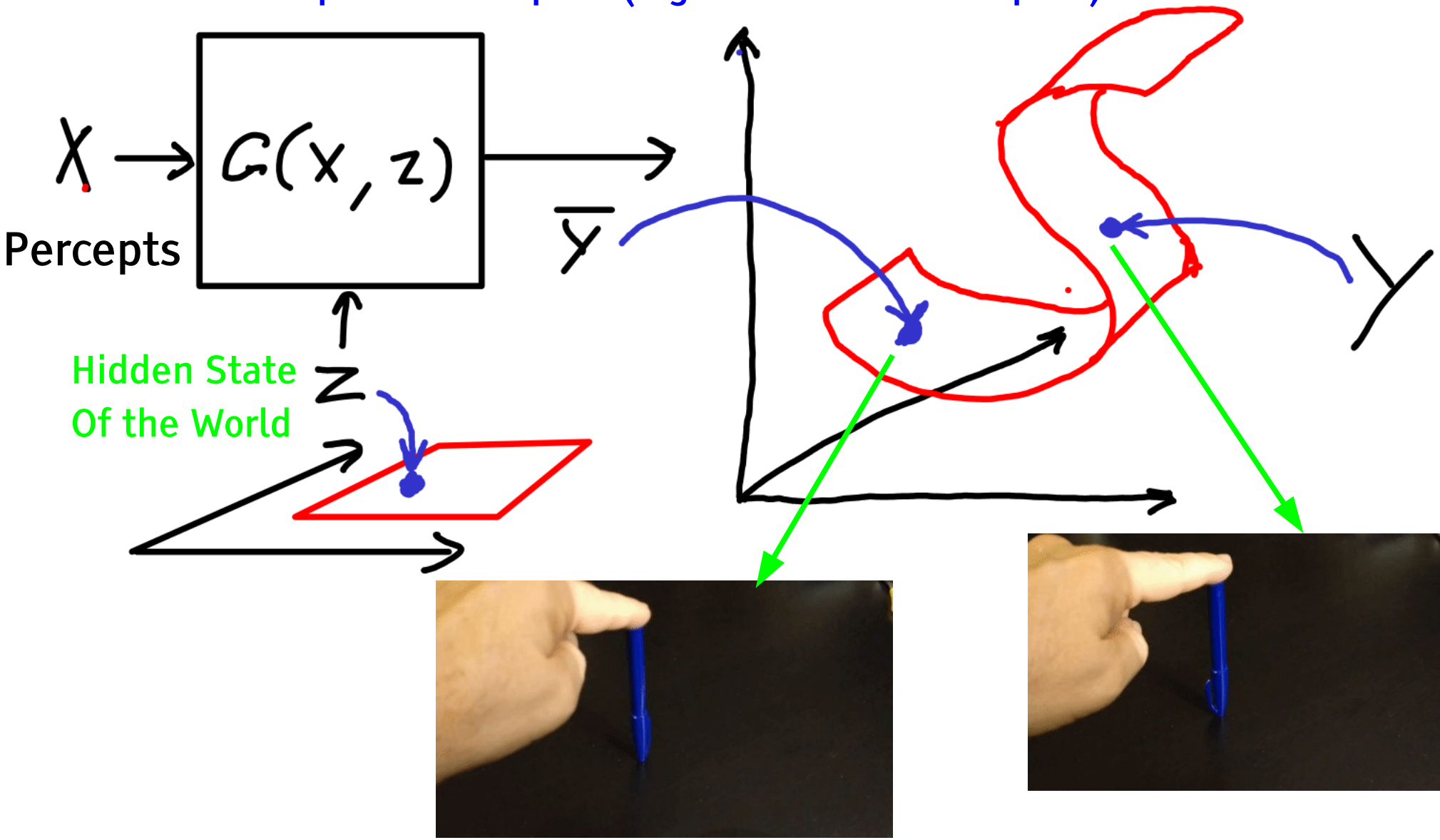
- Sparse coding, sparse auto-encoder, Predictive Sparse Decomposition




Adversarial Training

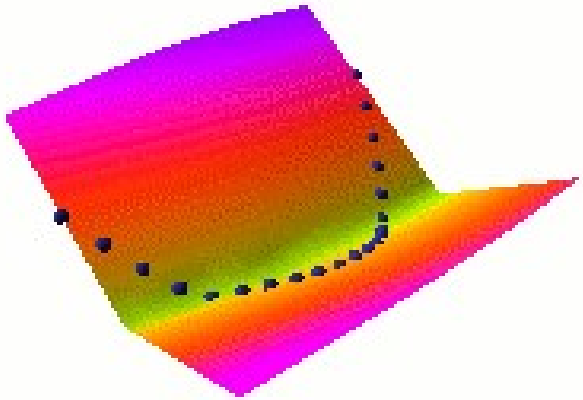
The Hard Part: Prediction Under Uncertainty

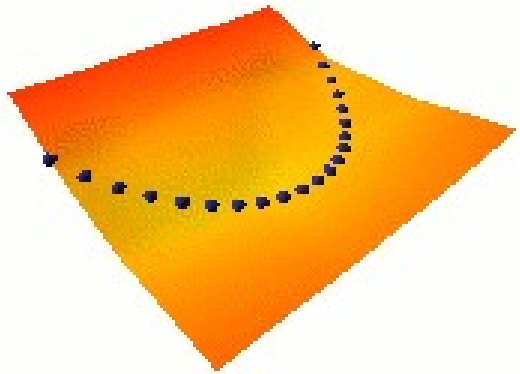
- Invariant prediction: The training samples are merely representatives of a whole set of possible outputs (e.g. a manifold of outputs).

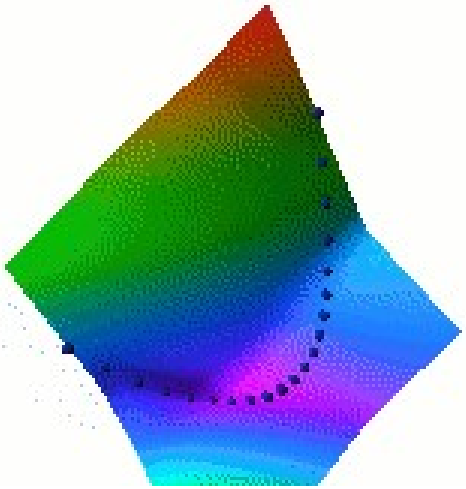


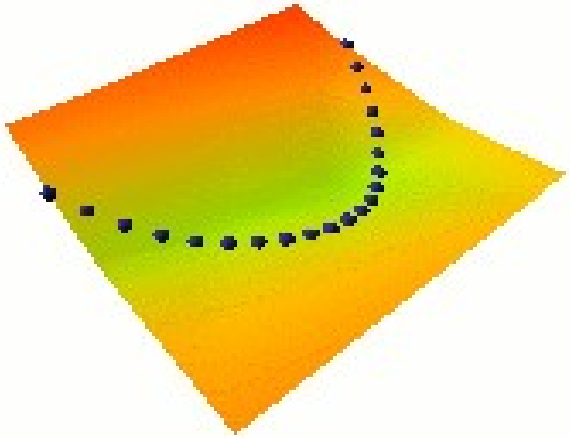
Energy-Based Unsupervised Learning

- Energy Function: Takes low value on data manifold, higher values everywhere else
- Push down on the energy of desired outputs. Push up on everything else.
- But how do we choose where to push up?



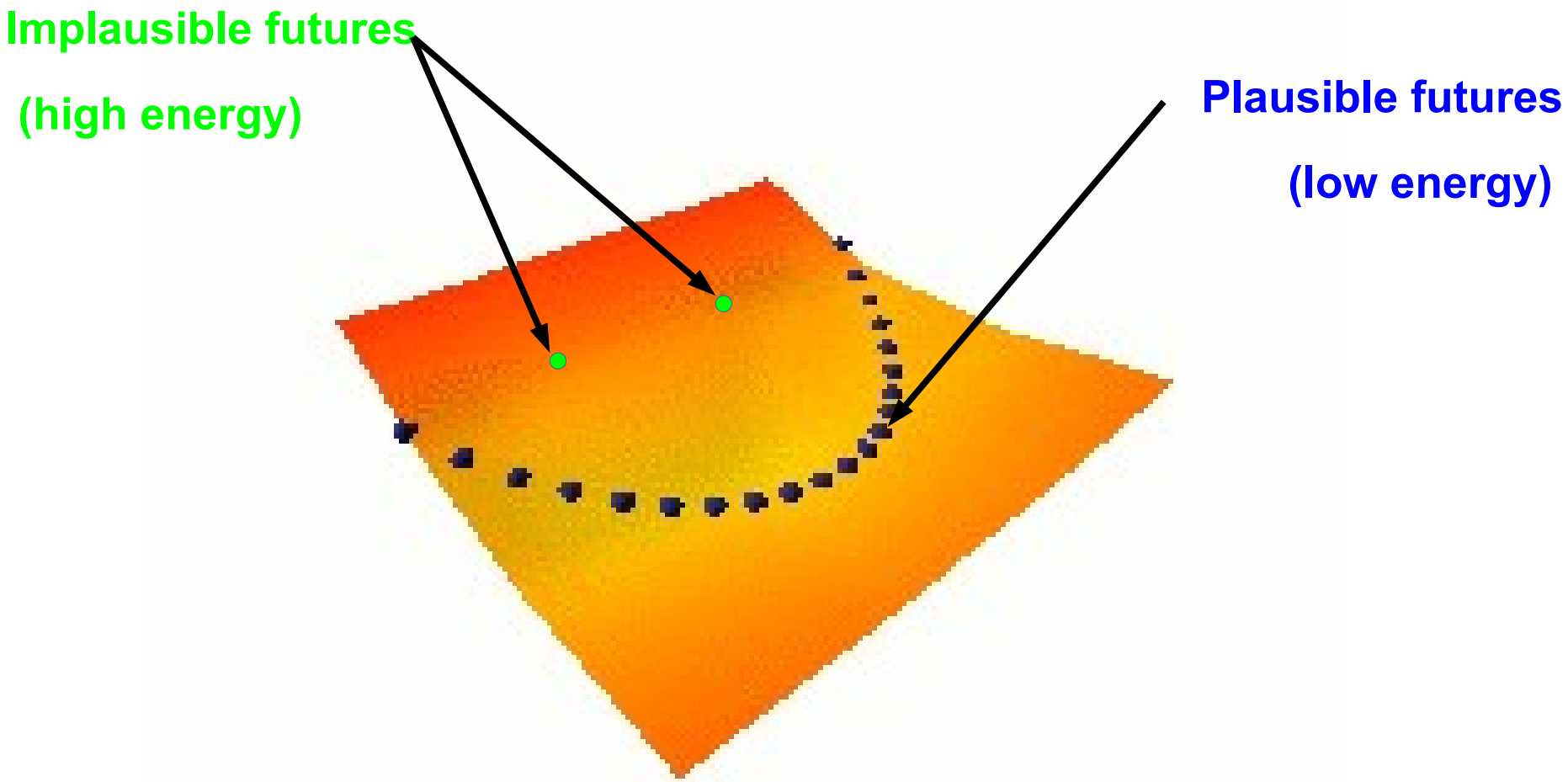






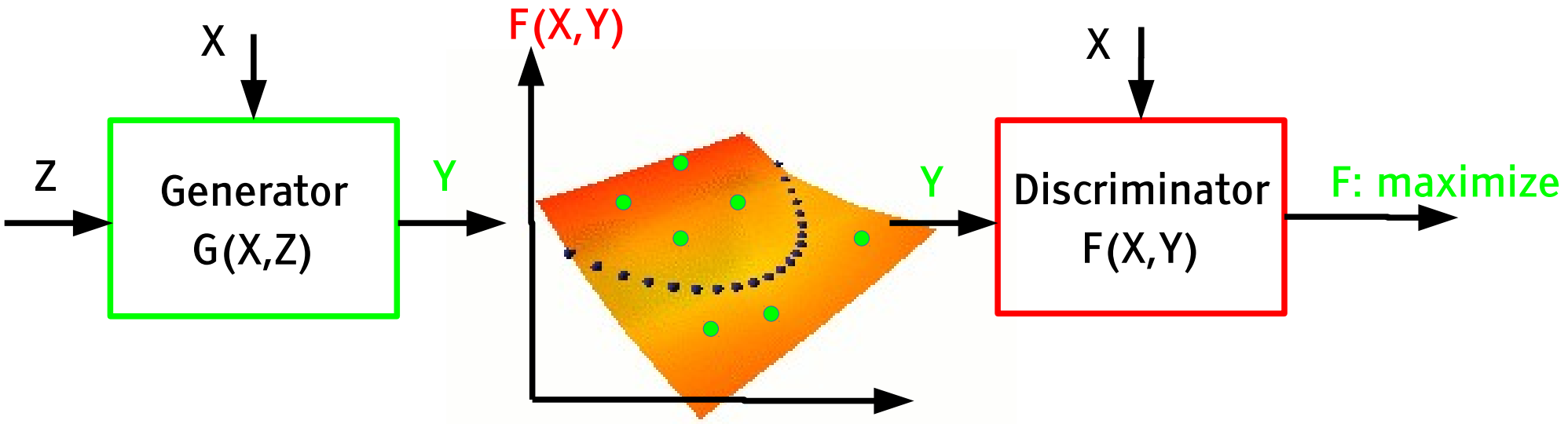
Energy-Based Unsupervised Learning

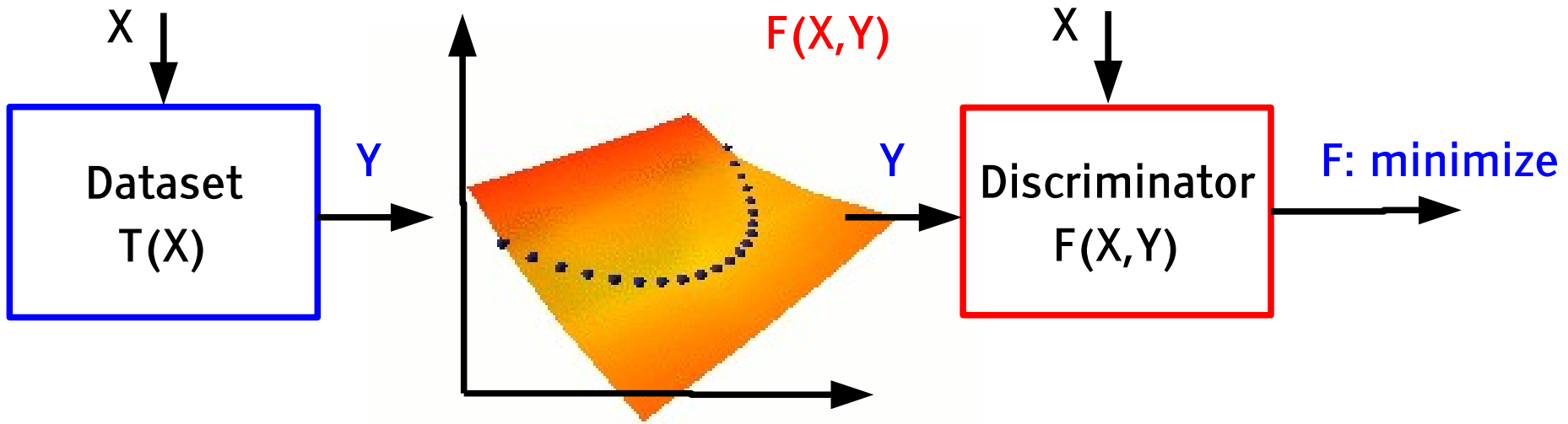
- **Energy Function:** Takes low value on data manifold, higher values everywhere else
- Push down on the energy of desired outputs. Push up on everything else.
- But how do we choose where to push up?



Adversarial Training: A Trainable Objective Function

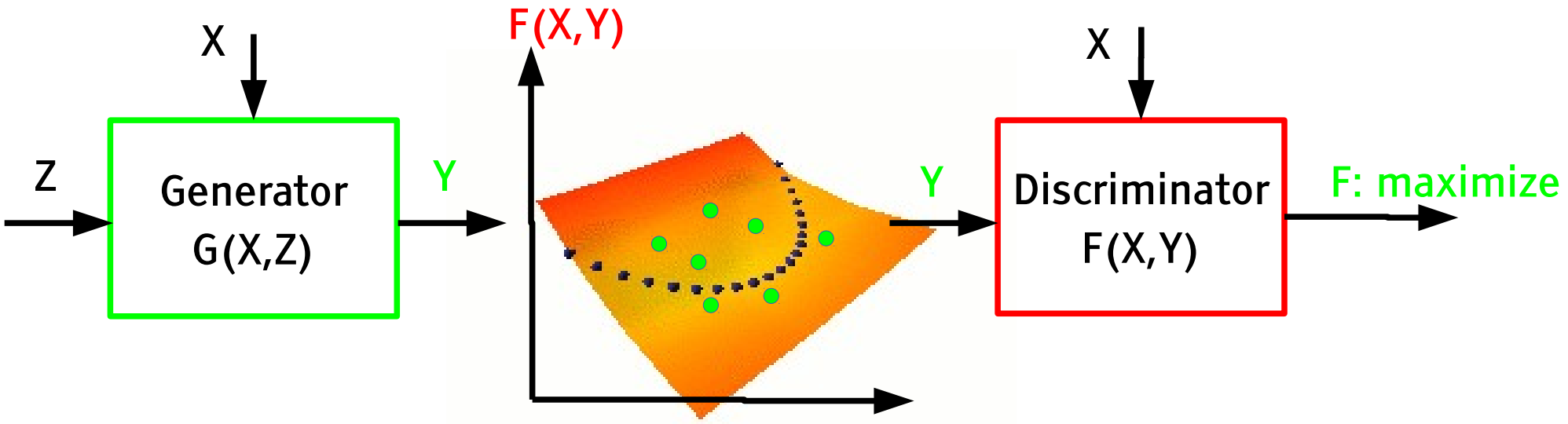
- **Adversarial Training [Goodfellow et al. NIPS 2014]**
- **Energy-based view of adversarial training: generator picks points to push up**

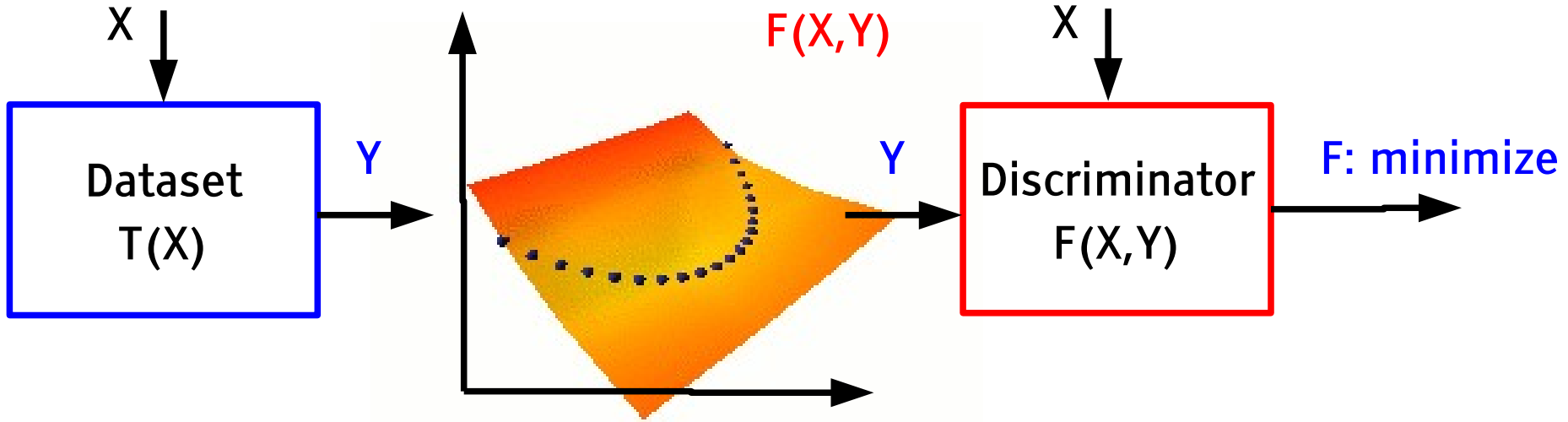




Adversarial Training: A Trainable Objective Function

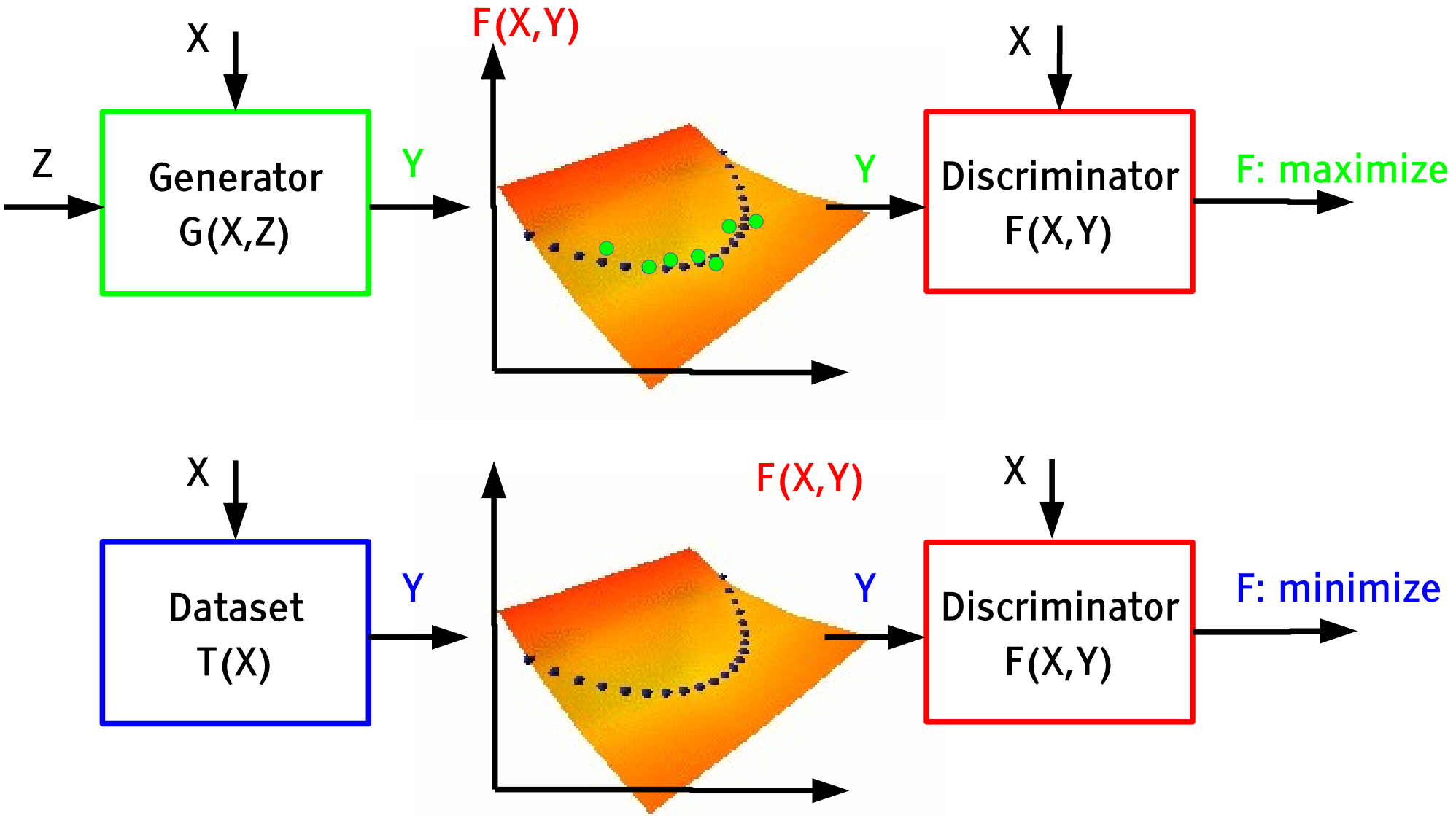
- **Adversarial Training [Goodfellow et al. NIPS 2014]**
- **Energy-based view of adversarial training: generator picks points to push up**





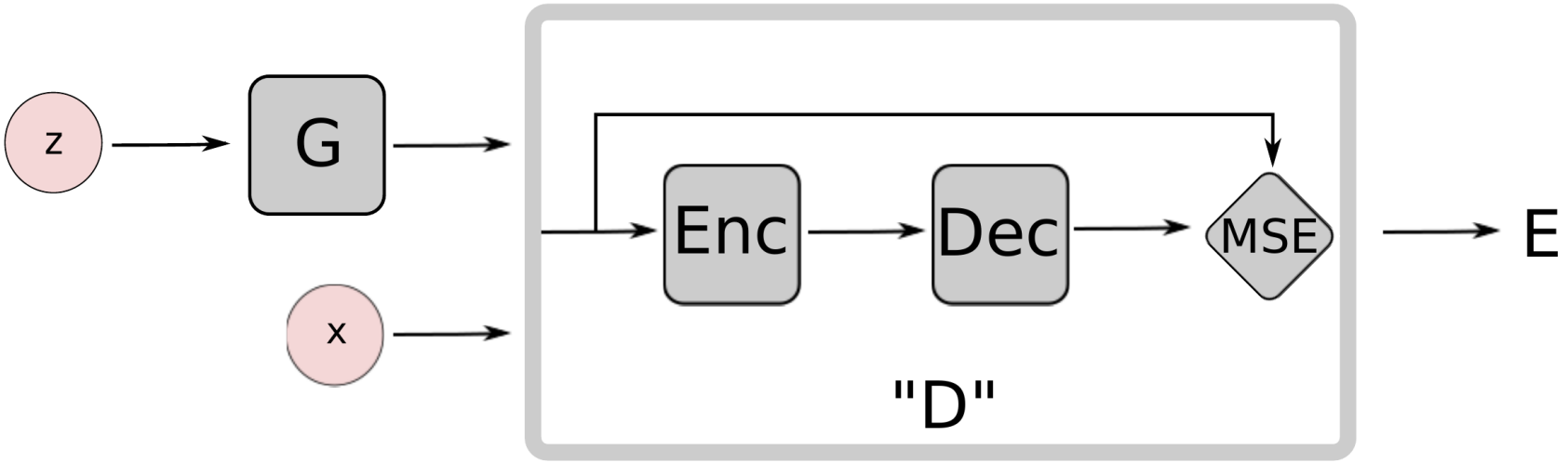
Adversarial Training: A Trainable Objective Function

- **Energy-based GAN [Zhao, Mathieu, LeCun: arXiv:1609.03.126]**



Energy-Based GAN [Zhao, Mathieu, LeCun: arXiv:1609.03.126]

- **Architecture: discriminator is an auto-encoder**



- **Loss functions**

$$f_D(x,z) = D(x) + [m - D\big(G(z)\big)]^+$$
$$= \|Dec\big(Enc(x)\big) - x\| + [m - \|Dec\Big(Enc\big(G(z)\big)\Big) - G(z)\|]^+,$$

$$f_G(z) = \|D\big(G(z)\big)\|$$
$$= \|Dec\Big(Enc\big(G(z)\big)\Big) - G(z)\|$$

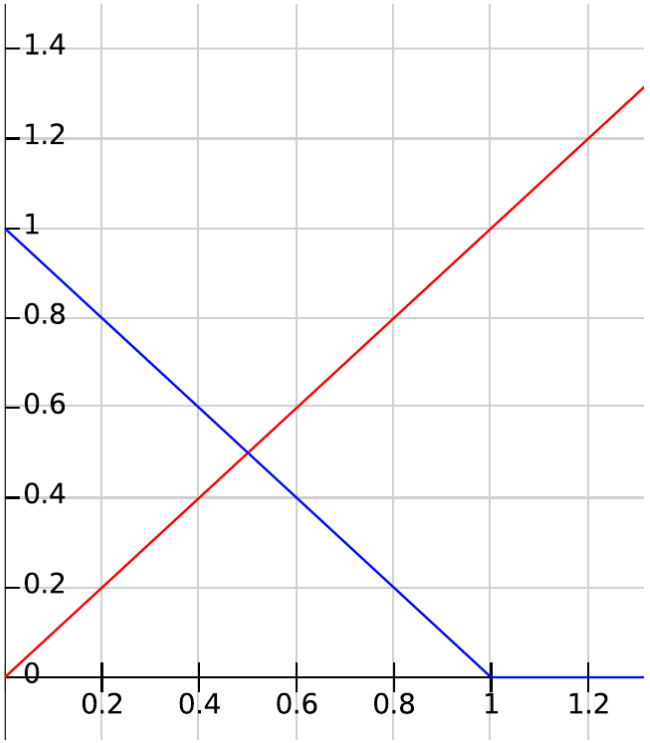
EBGAN Loss function

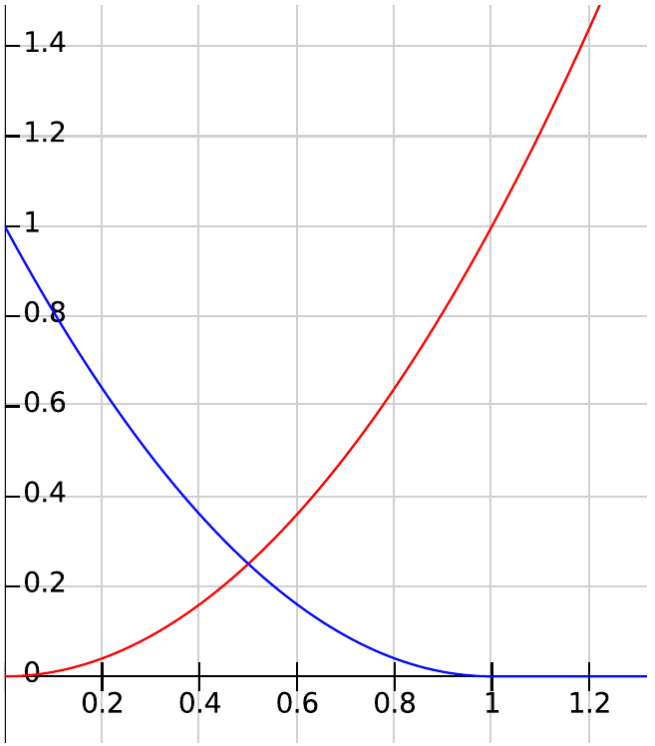
- **Loss functions for Discriminator and Generator.** **Assume D(x) is positive.**

$$L_D(x,z) = f(D(x)) + f([m - D(G(z))]^+)$$

$$L_G(z) = f(D(G(z)))$$

- **f must be strictly increasing & convex, with f(0)=0**
  - Examples: half-wave rectification, square





EBGAN solutions are Nash Equilibria

- **Loss functions for Discriminator and Generator. D(x) is positive.**

$$L_D(x,z)=f(D(x))+f([m-D(G(z))]^+)$$

$$L_G(z)=f(D(G(z)))$$

- **f must be strictly increasing & convex with f(0)=0**

- **(1) there is a Nash equilibrium, (2) if it is reached, the distributions are equal**

We define $V(G,D) = \int_{x,z} \mathcal{L}_D(x,z) p_{data}(x) p_z(z) \mathrm{d}x \mathrm{d}z$ and $U(G,D) = \int_z \mathcal{L}_G(z) p_z(z) \mathrm{d}z$.

$$V(G^*,D^*) \leq V(G^*,D) \qquad \forall D$$
$$U(G^*,D^*) \leq U(G,D^*) \qquad \forall G$$

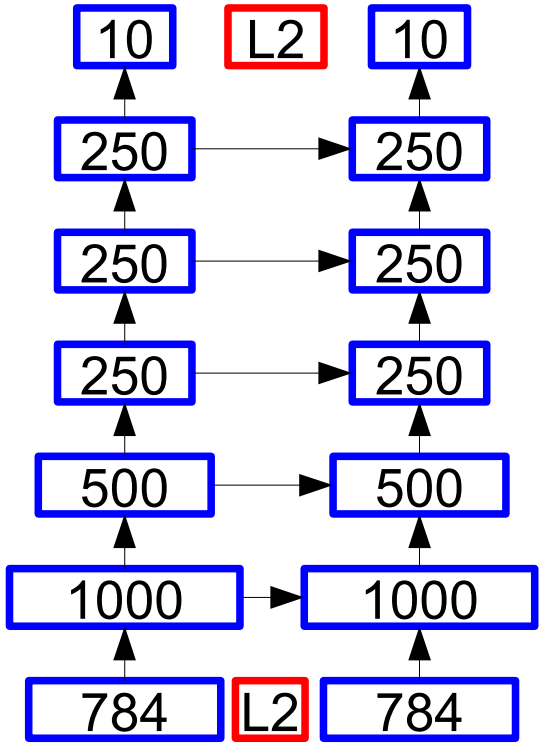
Theorem 1. *If (D^*,G^*) is a Nash equilibrium of the system, then $p_{G^*} = p_{data}$ almost everywhere, and $V(D^*,G^*) = m$.*

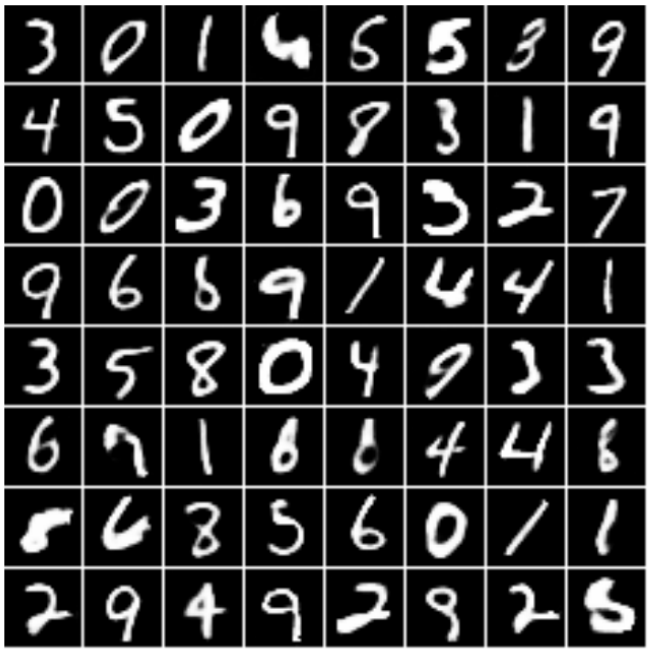
Theorem 2. *Nash equilibrium of this system exists and is characterized by (a) $p_{G^*} = p_{data}$ (almost everywhere) and (b) there exists a constant $\gamma \in [0,m]$ such that $D^*(x) = \gamma$ (almost everywhere).*[1]

EBGAN in which D is a Ladder Network

- **Ladder Network: auto-encoder with skip connections [Rasmus et al 2015]**
- **Permutation-invariant MNIST (fully connected nets)**

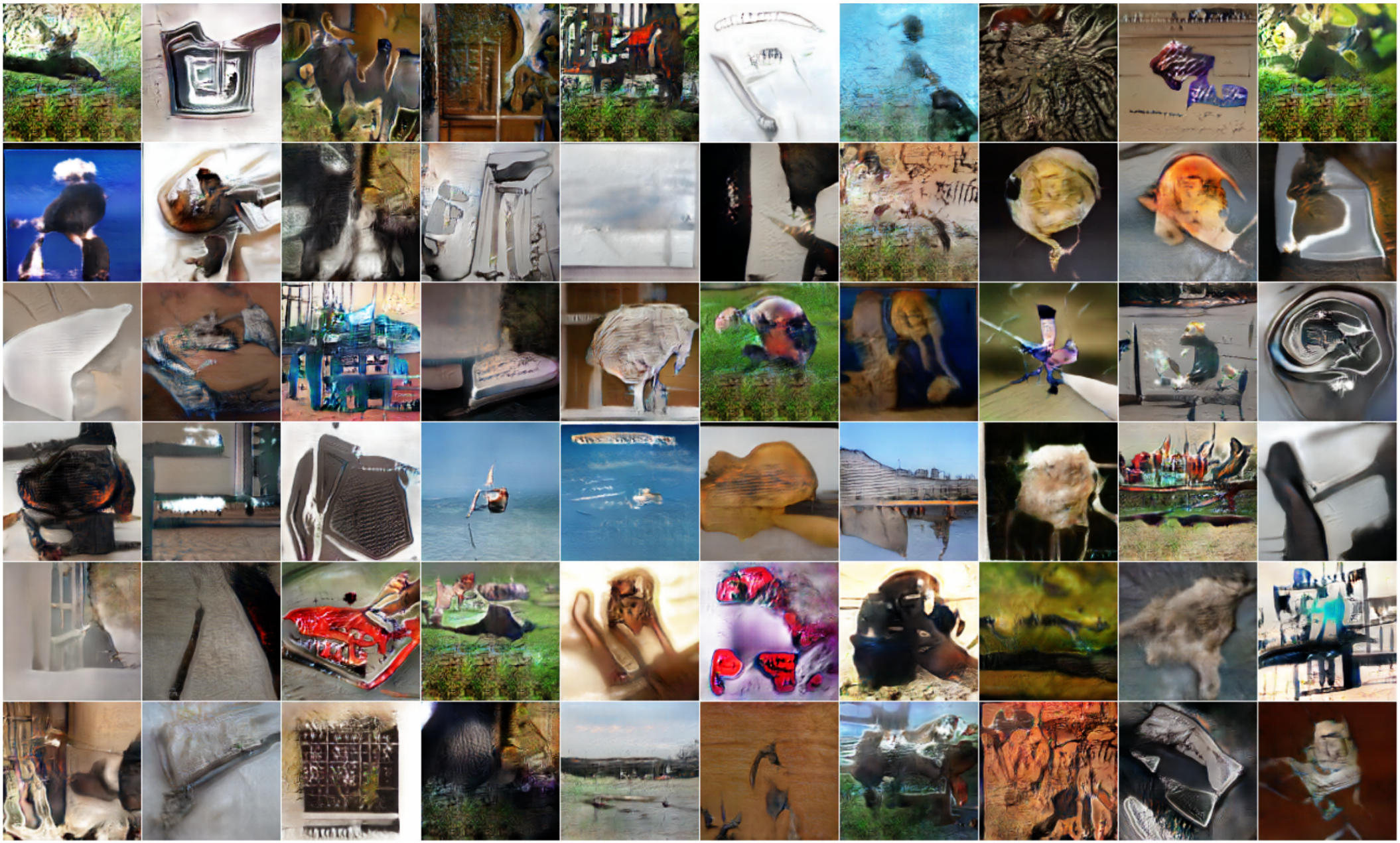
model	100	200	1000
LN bottom-layer-cost, reported in Pezeshki et al. (2015)	1.69±0.18	-	1.05±0.02
LN bottom-layer-cost, reported in Rasmus et al. (2015)	1.09±0.32	-	0.90±0.05
LN bottom-layer-cost, reproduced in this work (see appendix D)	1.36±0.21	1.24±0.09	1.04±0.06
LN bottom-layer-cost within EBGAN framework	**1.04±0.12**	**0.99±0.12**	**0.89±0.04**
Relative percentage improvement	23.5%	20.2%	14.4%





Energy-Based GAN trained on ImageNet at 128x128 pixels

Y LeCun



Energy-Based GAN trained on ImageNet at 256x256 pixels

Y LeCun

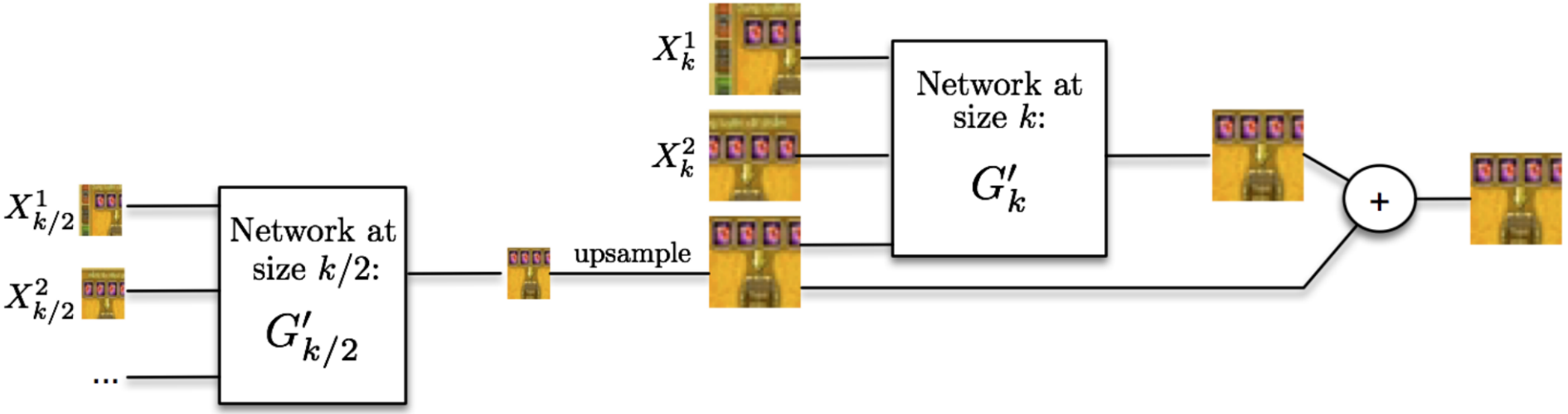
- Trained on dogs


Video Prediction
(with adversarial training)
[Mathieu, Couprie, LeCun ICLR 2016]
arXiv:1511:05440

Multi-Scale ConvNet for Video Prediction

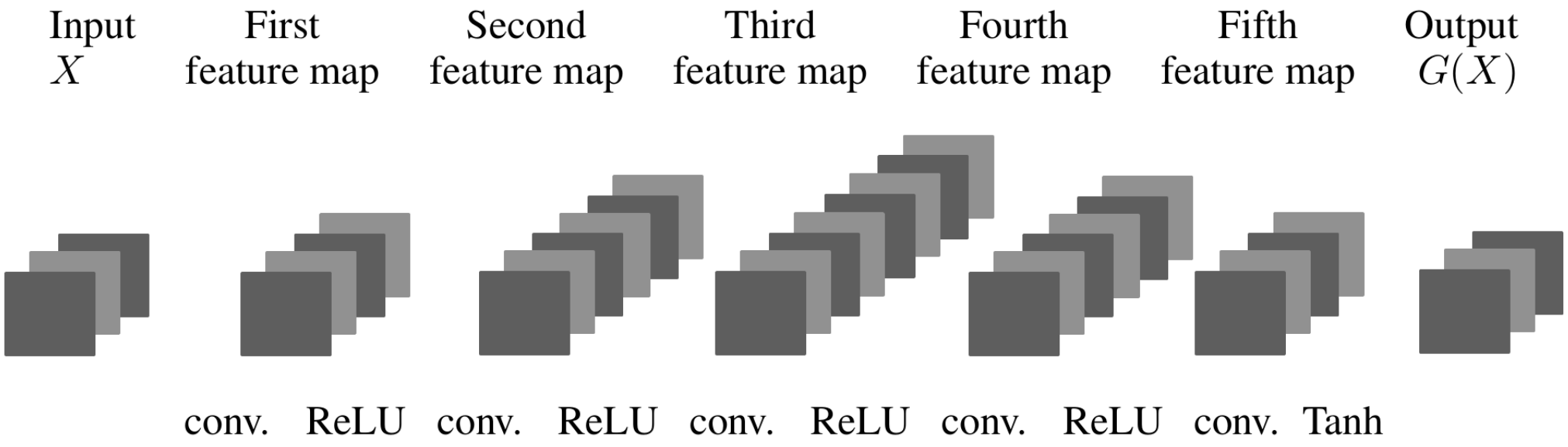
- 4 to 8 frames input → ConvNet → 1 to 8 frames output
- Multi-scale ConvNet, without pooling
- If trained with least square: blurry output

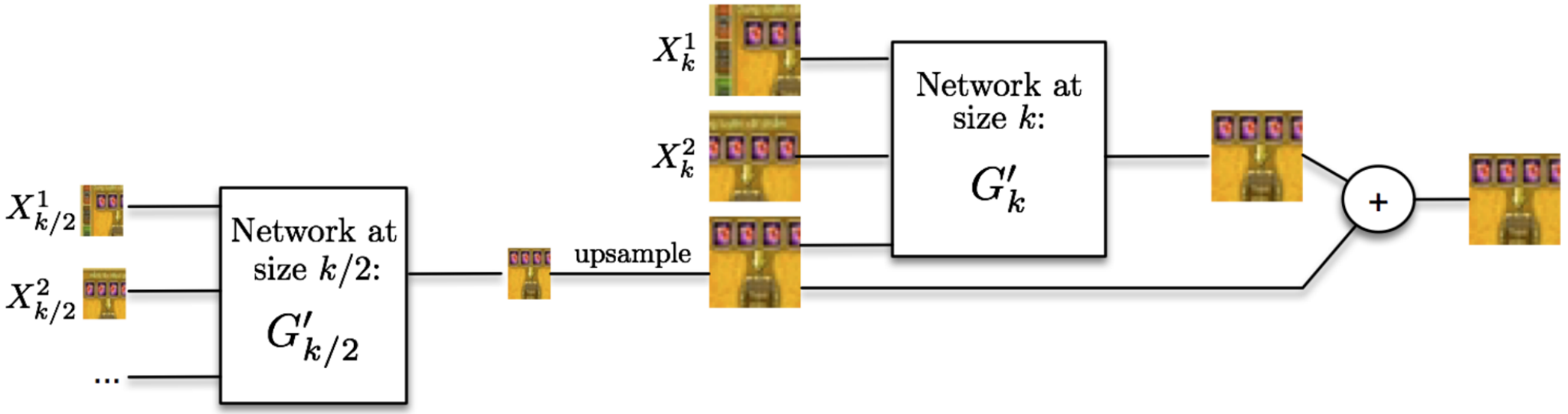
Predictor (multiscale ConvNet Encoder-Decoder)



Multi-Scale ConvNet for Video Prediction

- 4 to 8 frames input → ConvNet with no pooling → 1 to 8 frames output

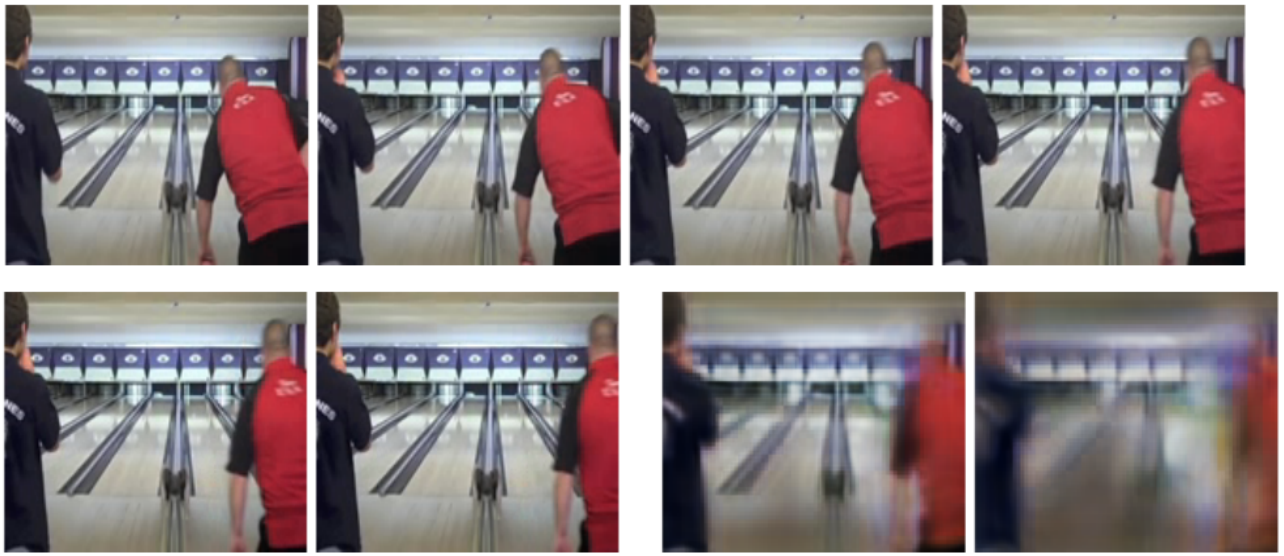




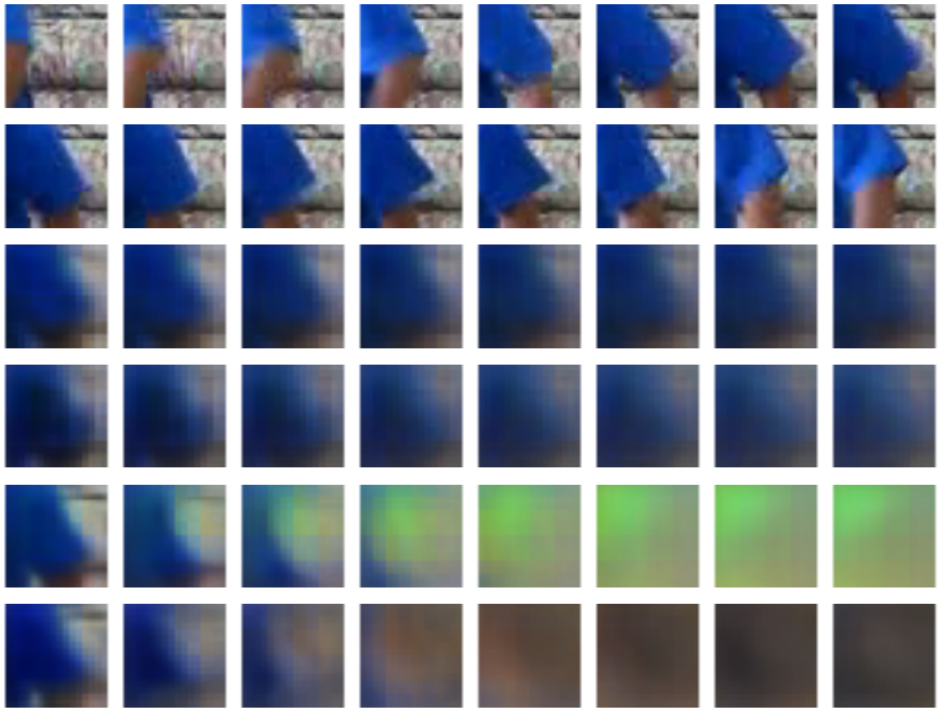
Can't Use Squared Error: blurry predictions

- The world is unpredictable
- MSE training predicts the average of possible futures:

  blurry images.



Ground truth ℓ_2 result



Input

Ground truth

ℓ_1

ℓ_2

ℓ_1 recursive

ℓ_2 recursive

Multi-Scale ConvNet for Video Prediction

Architectures

Models 4 frames in input – 1 frame in output

Generative network scales	G_1	G_2	G_3	G_4
Number of feature maps	128, 256, 128	128, 256, 128	128, 256, 512, 256, 128	128, 256, 512, 256, 128
Conv. kernel size	3, 3, 3, 3	5, 3, 3, 5	5, 3, 3, 3, 5	7, 5, 5, 5, 5, 7
Adversarial network scales	D_1	D_2	D_3	D_4
Number of feature maps	64	64, 128, 128	128, 256, 256	128, 256, 512, 128
Conv. kernel size (no padding)	3	3, 3, 3	5, 5, 5	7, 7, 5, 5
Fully connected	512, 256	1024, 512	1024, 512	1024, 512

Models 8 frames in input – 8 frames in output

Generative network scales	G_1	G_2	G_3	G_4
Number of feature maps	16, 32, 64	16, 32, 64	32, 64, 128	32, 64, 128, 128
Conv. kernel size	3, 3, 3, 3	5, 3, 3, 3	5, 5, 5, 5	7, 5, 5, 5, 5
Adversarial network scales	D_1	D_2	D_3	D_4
Number of feature maps	16	16, 32, 32	32, 64, 64	32, 64, 128, 128
Conv. kernel size (no padding)	3	3, 3, 3	5, 5, 5	7, 7, 5, 5
Fully connected	128, 64	256, 128	256, 128	256, 128

Multi-Scale ConvNet for Video Prediction

Results on UCF101 (10% of test images)

- 8 frames input → 8 frames output

	1st frame prediction scores		8th frame prediction scores	
	PSNR	Sharpness	PSNR	Sharpness
ℓ_2	18.3	0.47	16.4	0.36
Adv	21.0	0.65	18.9	0.54
ℓ_1	21.2	0.57	18.3	0.51
GDL ℓ_1	**21.8**	**0.87**	**19.2**	**0.79**

Results on UCF101 (10% of test images)

- 4 frames input → 1 frames output

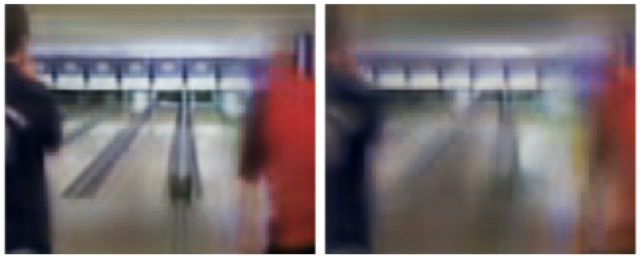
	1st frame prediction scores		2nd frame prediction scores	
	PSNR	Sharpness	PSNR	Sharpness
ℓ_2	20.1	0.48	14.1	0.29
ℓ_1	22.2	0.58	16.0	0.48
GDL ℓ_1	23.9	0.69	18.5	0.60
Adv	24.4	**0.95**	18.9	**1.00**
Adv+GDL	**27.2**	0.77	**22.6**	0.68

Multi-Scale ConvNet for Video Prediction

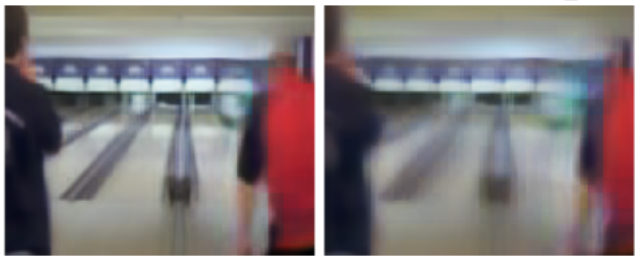
Examples

Input frames

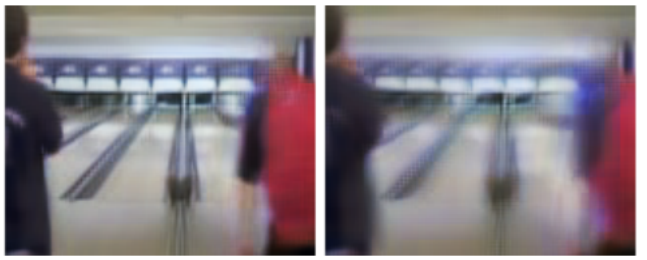
Ground truth



ℓ_2 result



ℓ_1 result



GDL ℓ_1 result

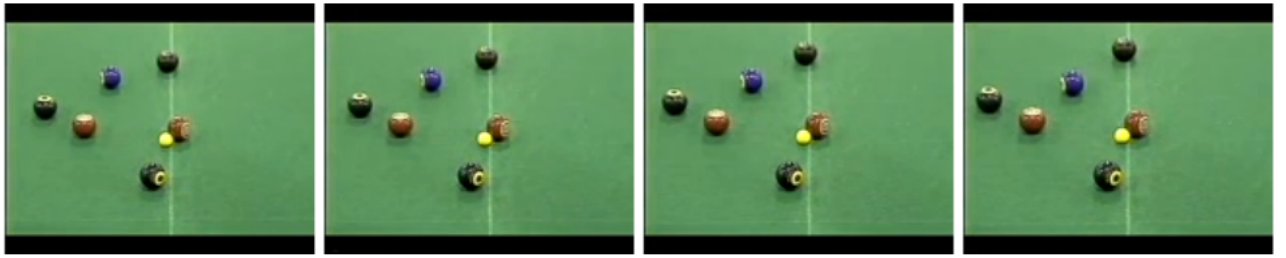
Adversarial result

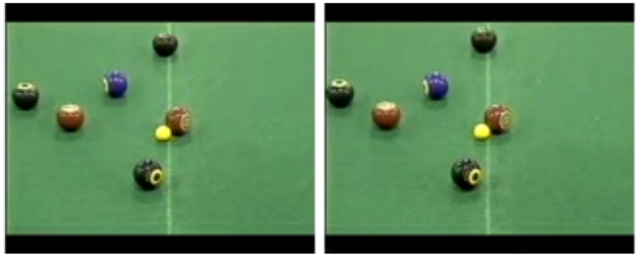
Adversarial+GDL result

Multi-Scale ConvNet for Video Prediction

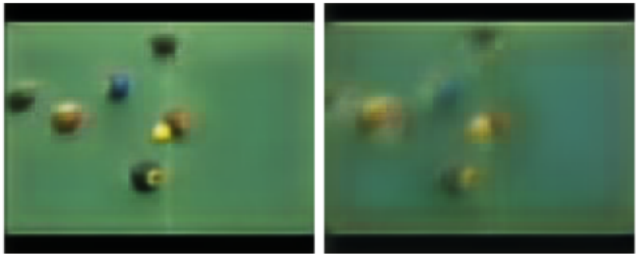
Examples

Input frames

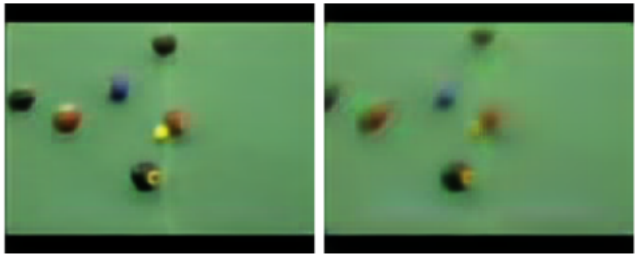




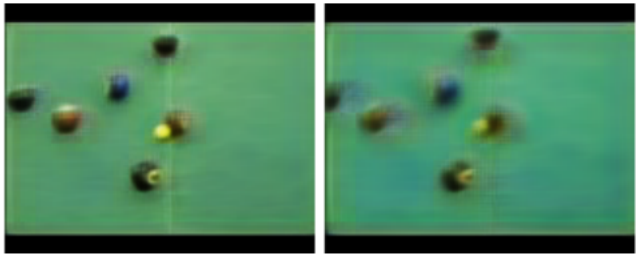
Ground truth



ℓ_2 result

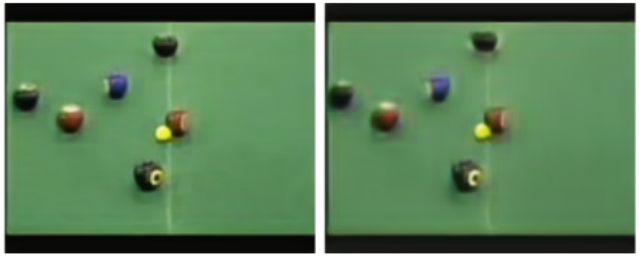


ℓ_1 result



GDL ℓ_1 result

Adversarial result

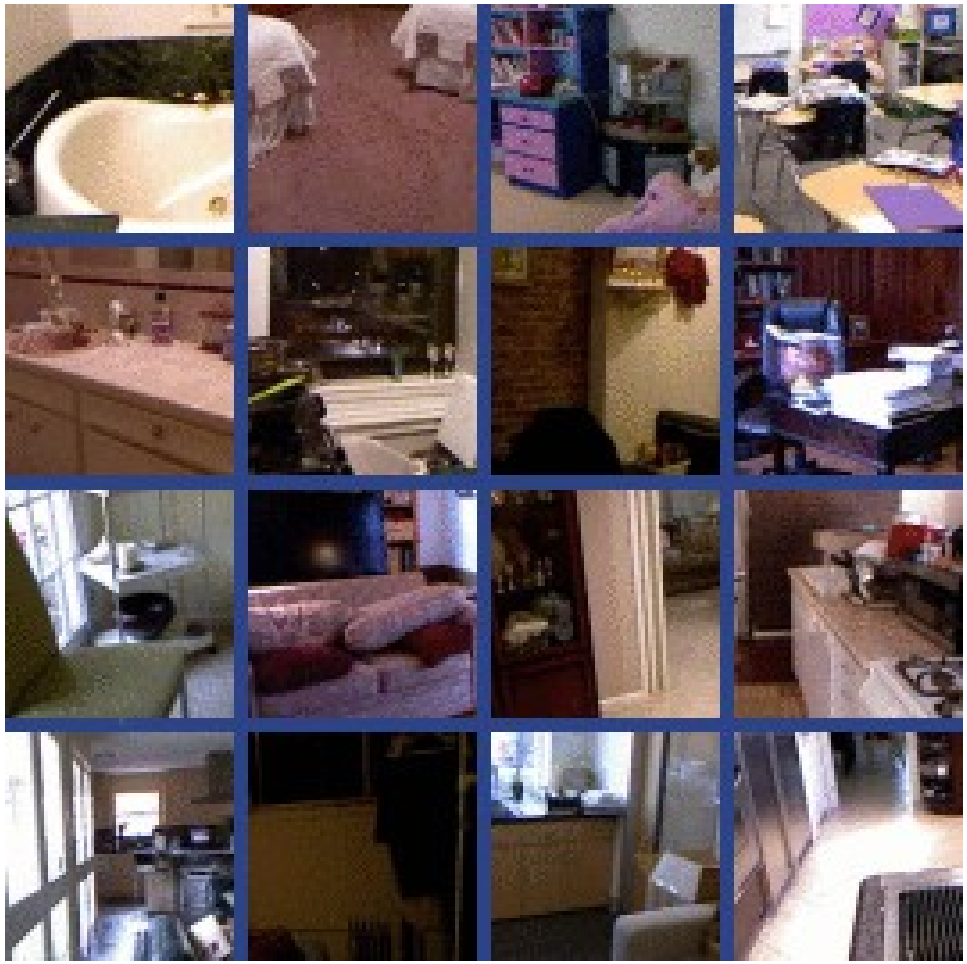


Adversarial+GDL result

Predictive Unsupervised Learning

- Our brains are "prediction machines"
- Can we train machines to predict the future?
- Some success with "adversarial training"
  - [Mathieu, Couprie, LeCun arXiv:1511:05440]
- But we are far from a complete solution.

Video Prediction: predicting 5 frames



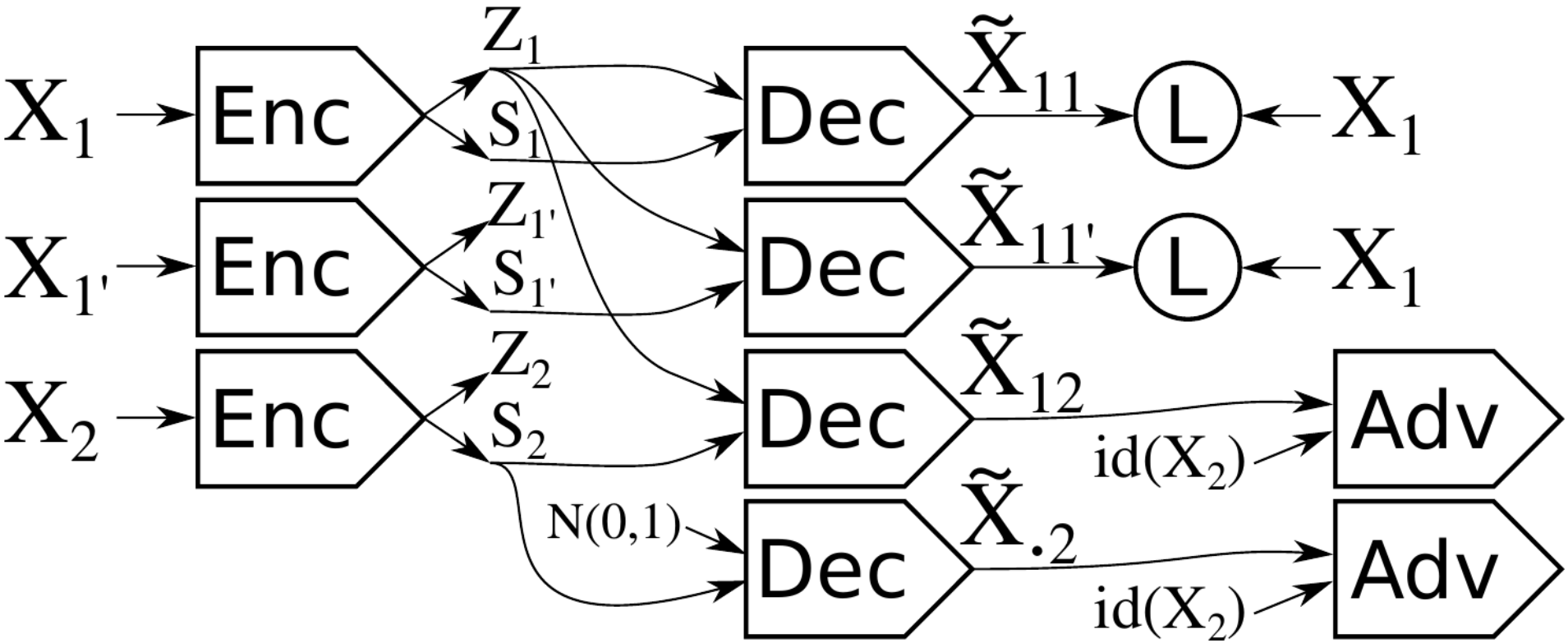
Video Prediction: predicting 5 frames

Video Prediction: predicting 50 frames


Style Transfer
(Mathieu et al. NIPS 2016)

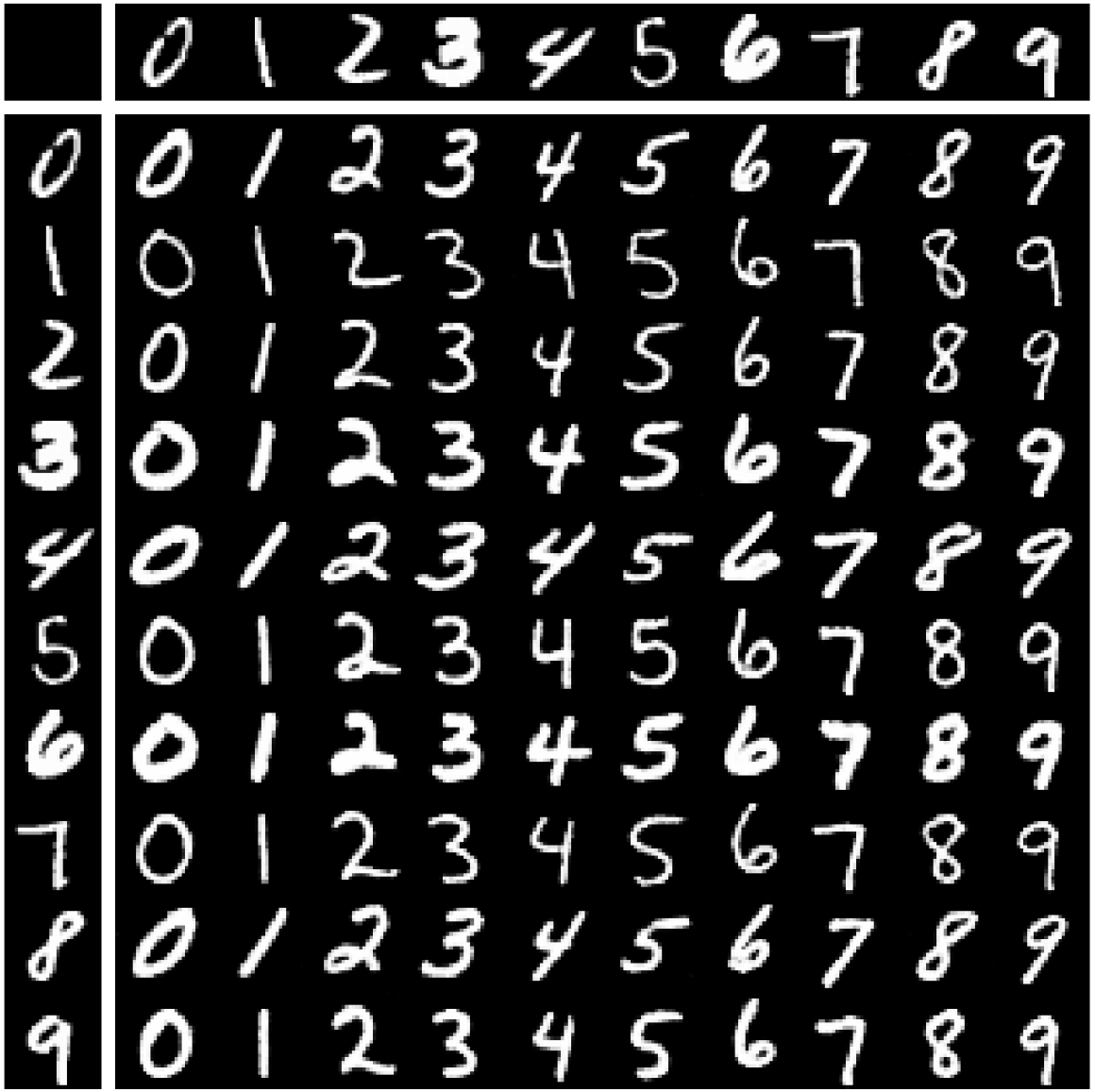
Style transfer architecture

- X1 and X1′ have same "label" (or known features)
- X2 can have any label
- S1 and S1′ are meant to represent the "label" (the known part of the representation)
- Z1, Z1′ and Z2 are the unspecified part (eg the pose)



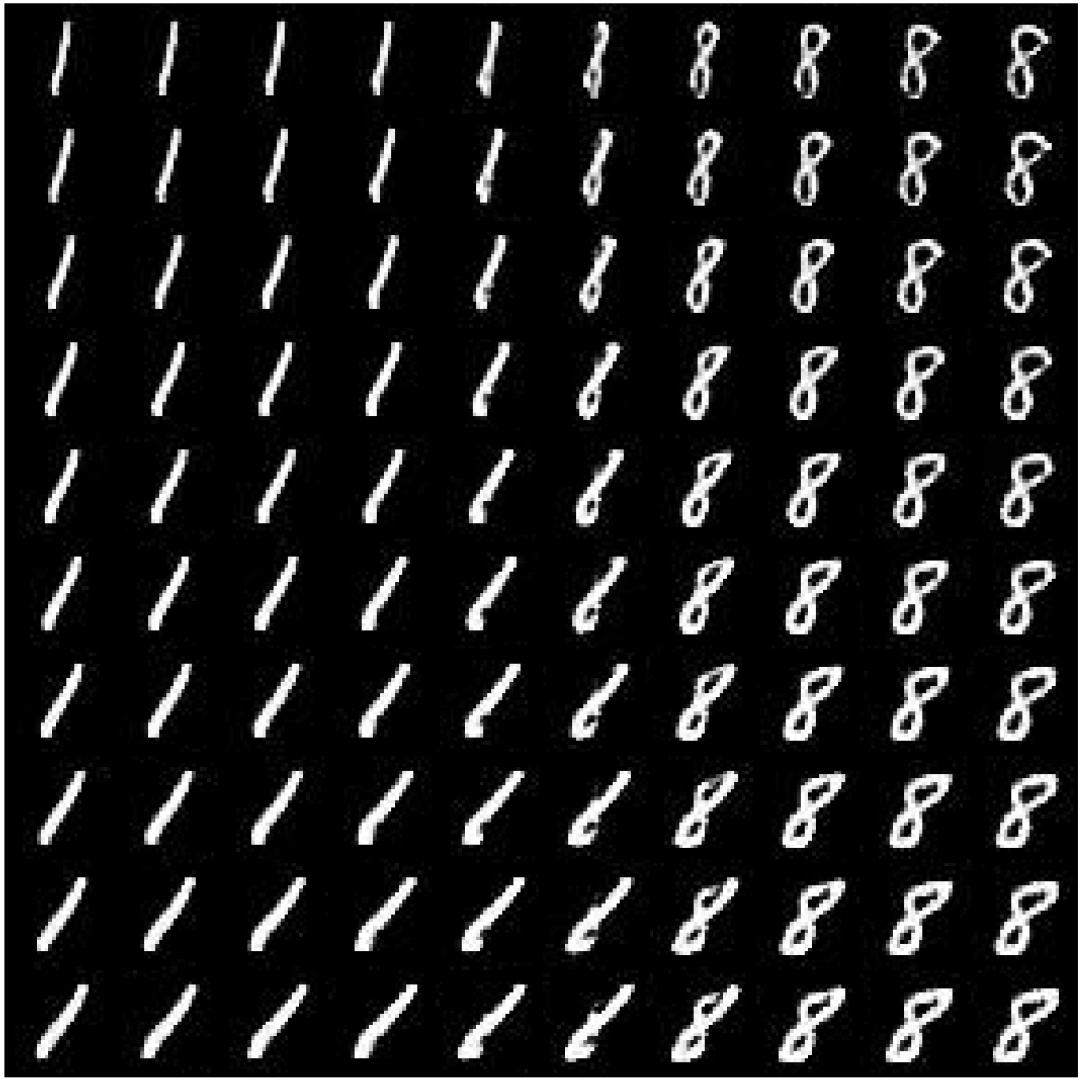
Style transfer results

- Transfer category from top row to style of left column



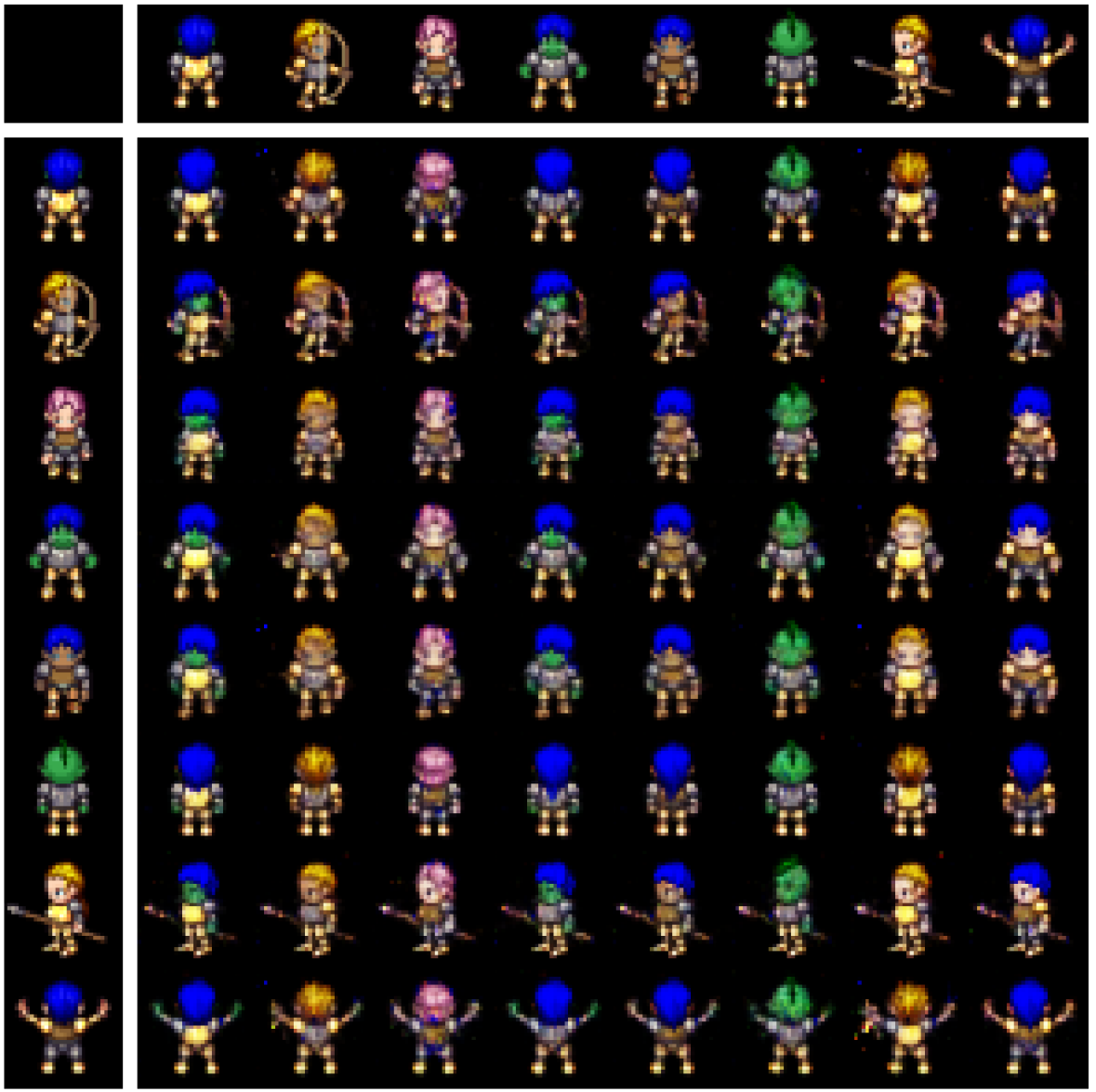
Style transfer: interpolation

- Interpolate between top left and bottom right characters
- Style changes vertically, identity changes horizontally.



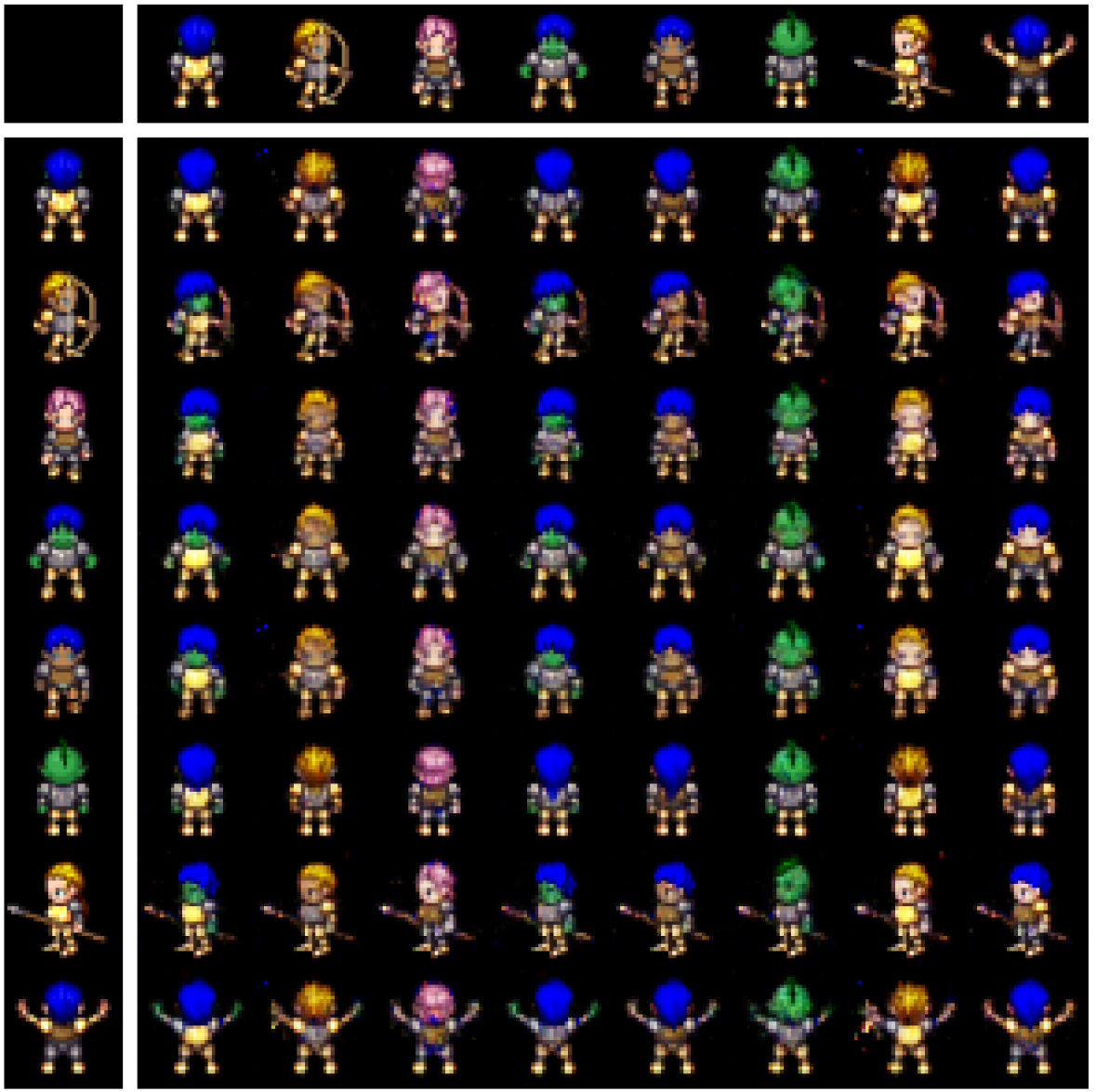
Style transfer results

- Transfer category from top row to style of left column



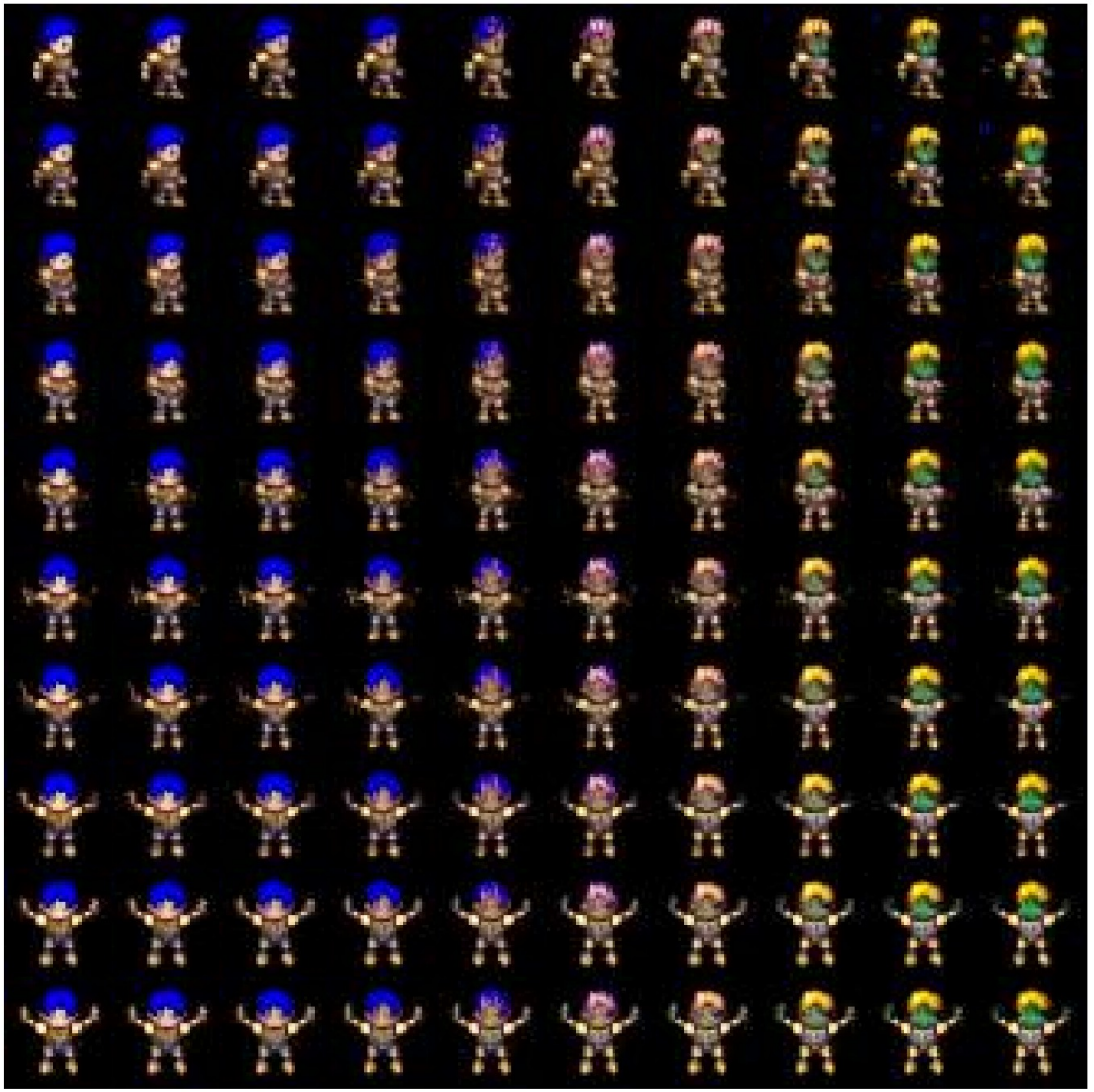
Style transfer results

- Transfer category from top row to style of left column



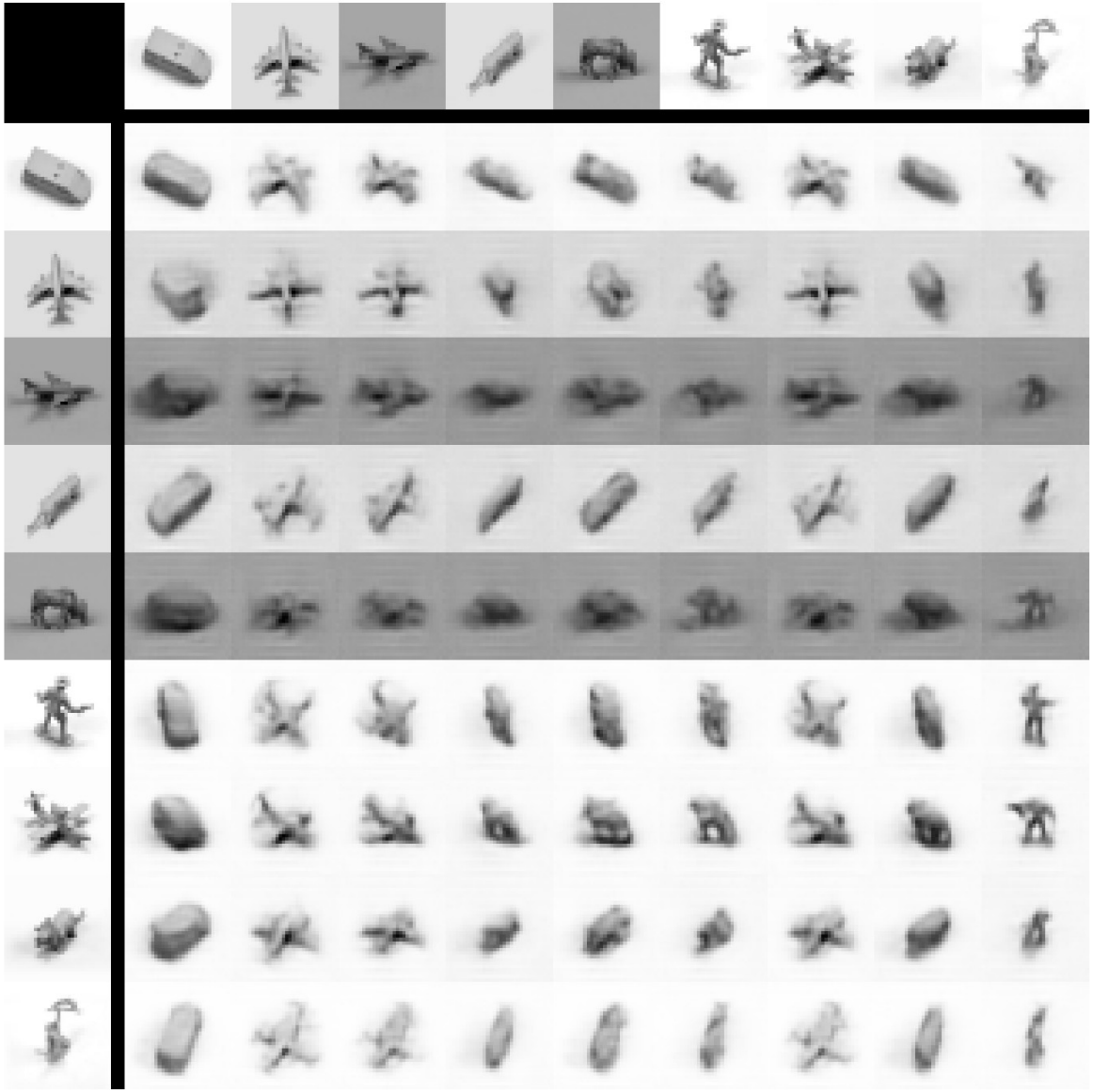
Style transfer: interpolation

- Interpolate between top left and bottom right characters
- Style changes vertically, identity changes horizontally.



Pose transfer results

- Transfer category from top row to orientation of left column



Pose transfer results

- Transfer category from top row to orientation of left column