



# Self-Supervised Learning

Yann LeCun  
New York University  
Facebook AI Research  
<http://yann.lecun.com>

# Supervised Learning works but requires many labeled samples

- ▶ Training a machine by showing examples instead of programming it
- ▶ When the output is wrong, tweak the parameters of the machine
- ▶ Works well for:
  - ▶ Speech→words
  - ▶ Image→categories
  - ▶ Portrait→ name
  - ▶ Photo→caption
  - ▶ Text→topic
  - ▶ ....



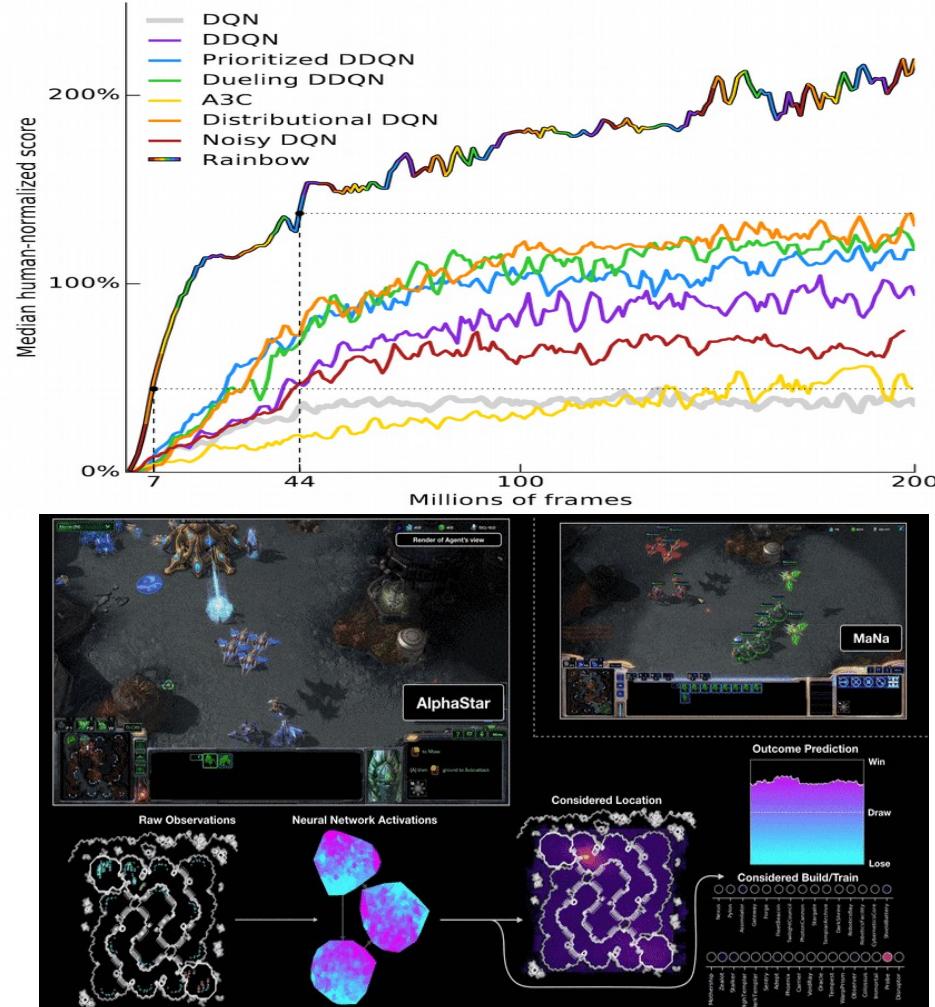
CAR



PLANE

# Reinforcement Learning: Model-Free RL works great for games.

- ▶ **57 Atari games: takes 83 hours equivalent real-time (18 million frames) to reach a performance that humans reach in 15 minutes of play.**
- ▶ [Hessel ArXiv:1710.02298]
- ▶ Elf OpenGo v2: 20 million self-play games. (2000 GPU for 14 days)
- ▶ [Tian arXiv:1902.04522]
- ▶ StarCraft: AlphaStar 200 years of equivalent real-time play
- ▶ [Vinyals blog post 2019]
- ▶ They all use ConvNets and a few other architectural concepts.



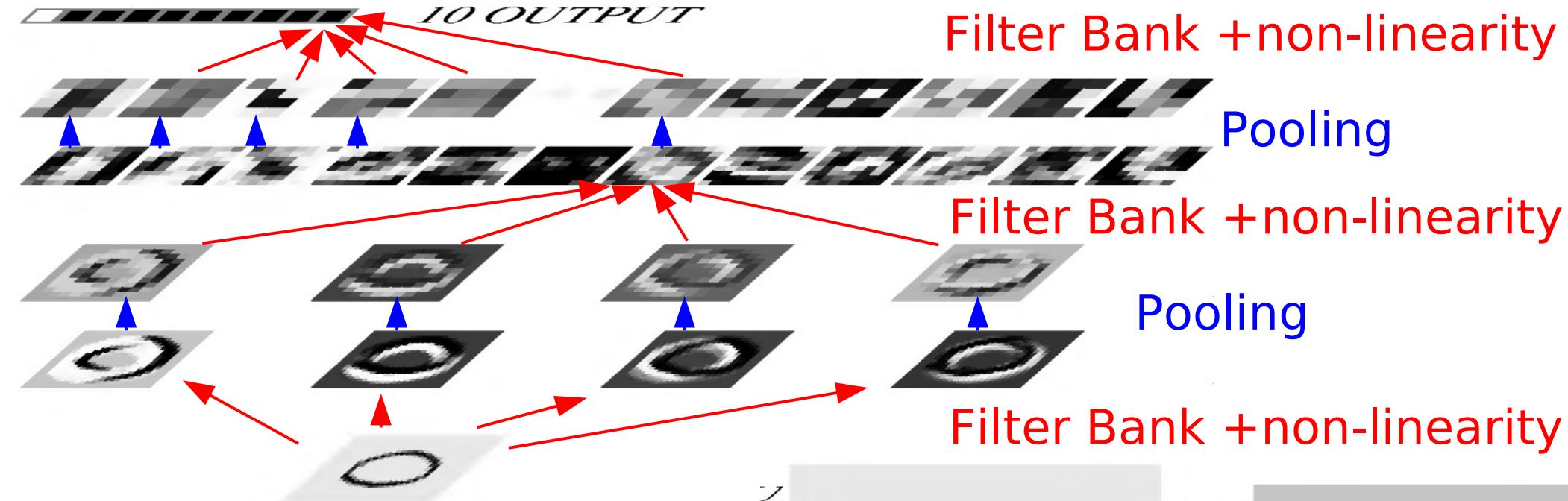
# But RL Requires too many trials in the real world

- ▶ Pure RL requires too many trials to learn anything
  - ▶ it's OK in a game
  - ▶ it's not OK in the real world
- ▶ RL works in simple virtual world that you can run faster than real-time on many machines in parallel.

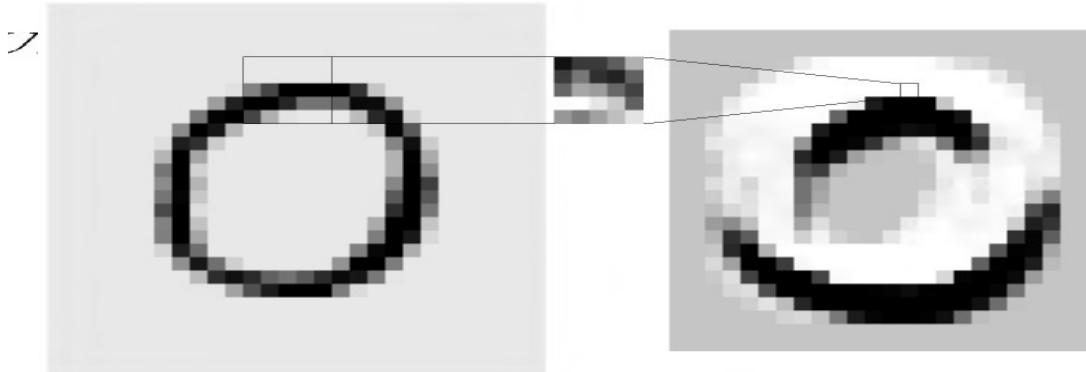


- ▶ Anything you do in the real world can kill you
- ▶ You can't run the real world faster than real time

# Convolutional Network Architecture [LeCun et al. NIPS 1989]



- Inspired by [Hubel & Wiesel 1962] & [Fukushima 1982] (Neocognitron):
  - ▶ simple cells detect local features
  - ▶ complex cells “pool” the outputs of simple cells within a retinotopic neighborhood.



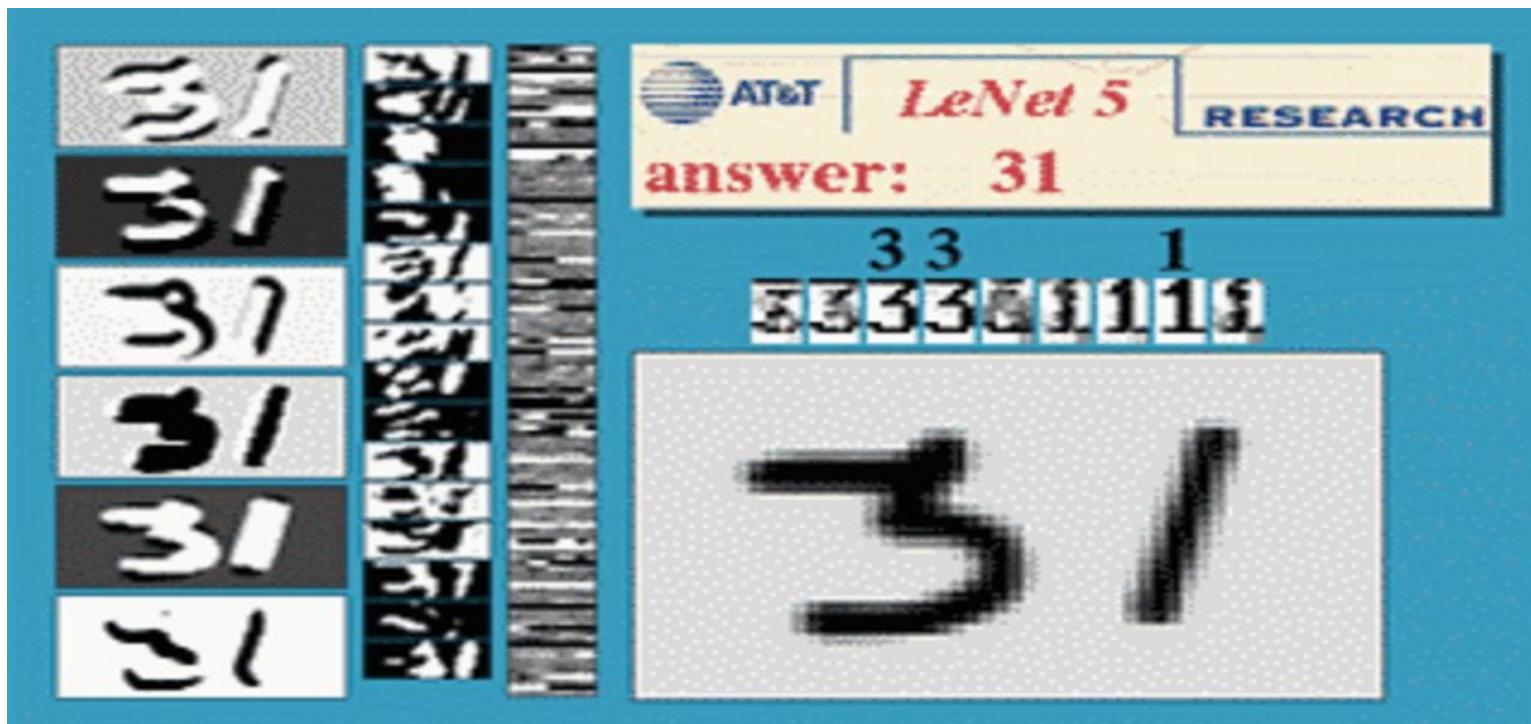
# Convolutional Network (LeNet5, vintage 1990)

Filters-tanh → pooling → filters-tanh → pooling → filters-tanh

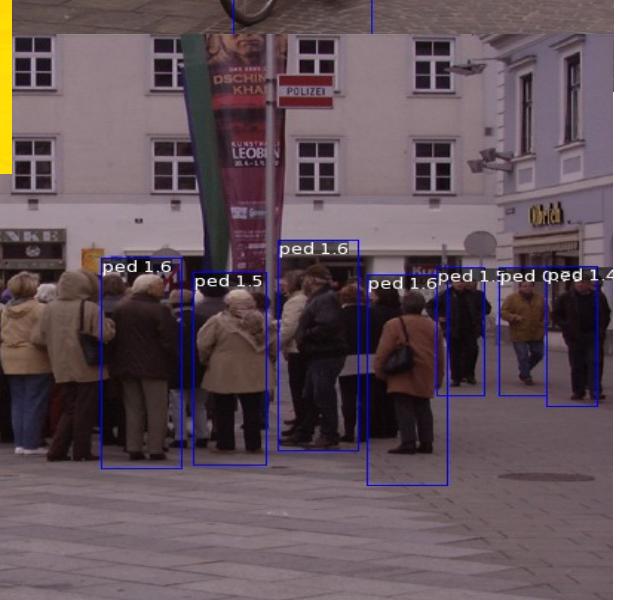
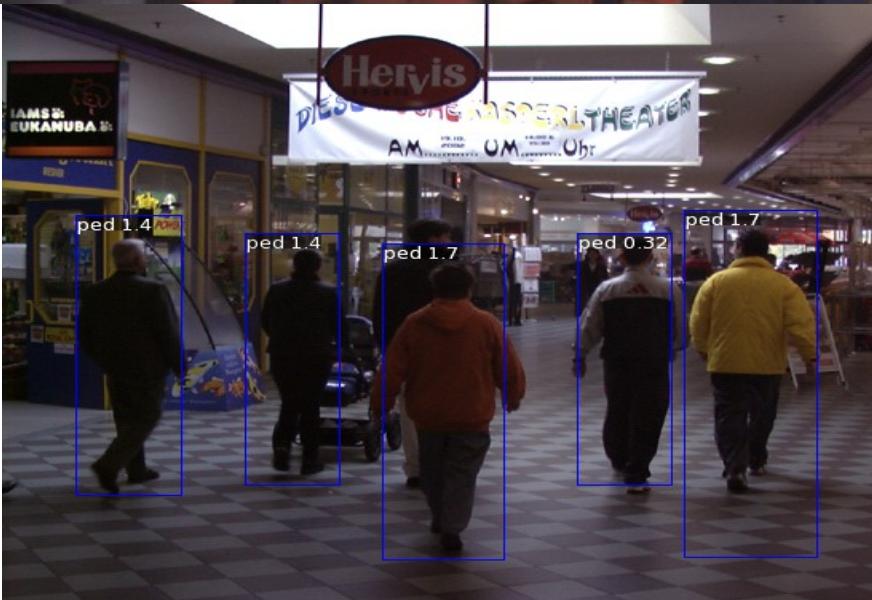
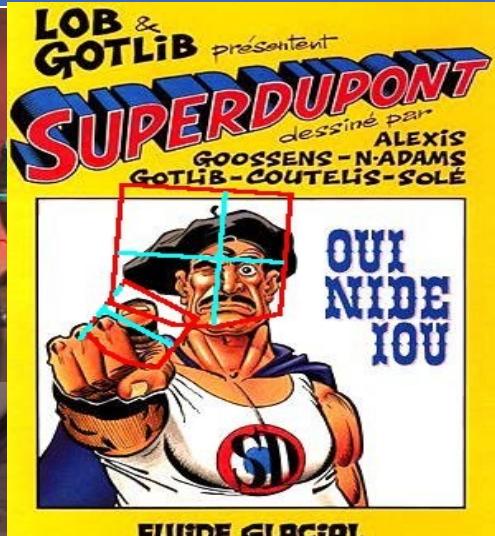
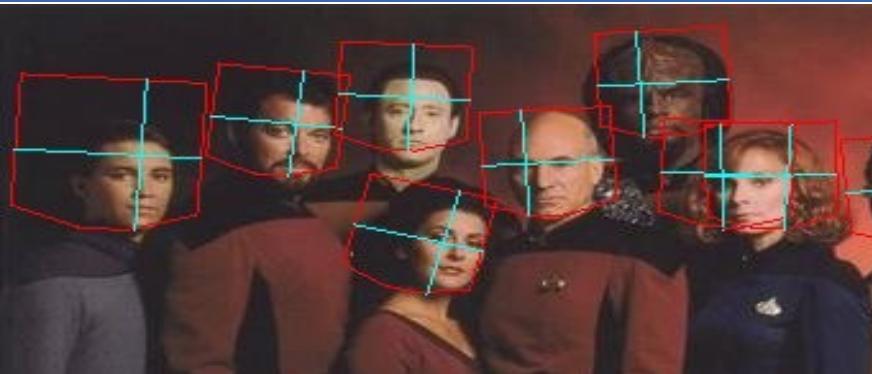


# ConvNets can recognize multiple objects

- ▶ All layers are convolutional
- ▶ Networks performs simultaneous segmentation and recognition



# Face & Pedestrian Detection with ConvNets (1993-2005)



# Semantic Segmentation with ConvNets (33 categories)

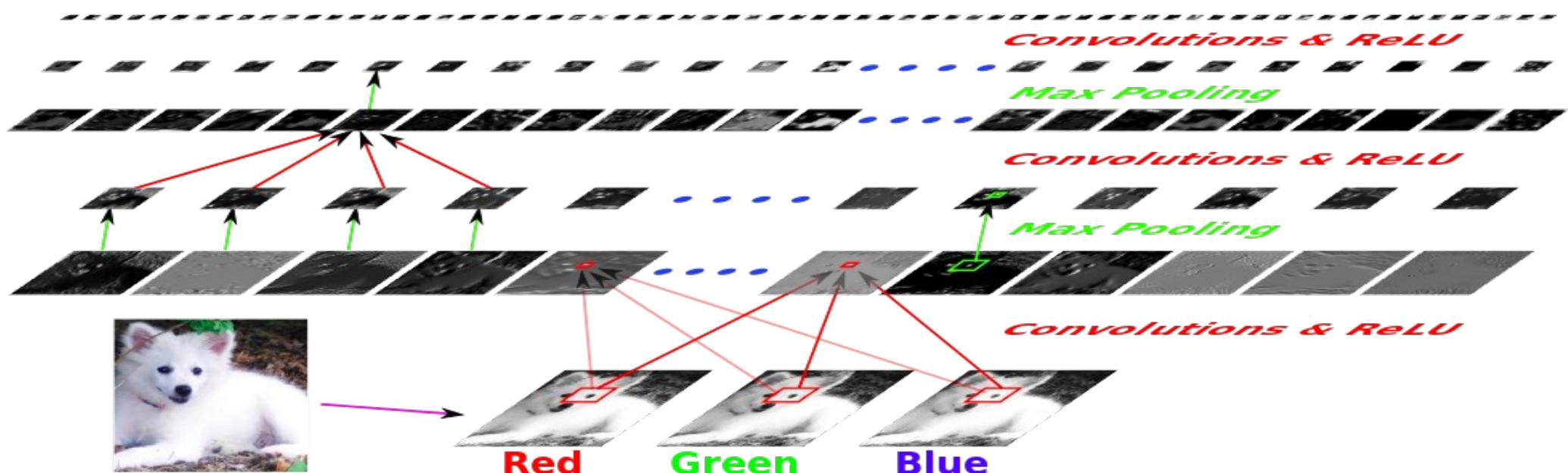
► [Farabet et al. ICML 2012]



# Deep ConvNets for Object Recognition (on GPU)

- AlexNet [Krizhevsky et al. NIPS 2012], OverFeat [Sermanet et al. 2013]
- 1 to 10 billion connections, 10 million to 1 billion parameters, 8 to 20 layers.

Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic Fox (1.0); Eskimo Dog (0.6); White Wolf (0.4); Siberian Husky (0.4)



# Deep ConvNets (depth inflation)

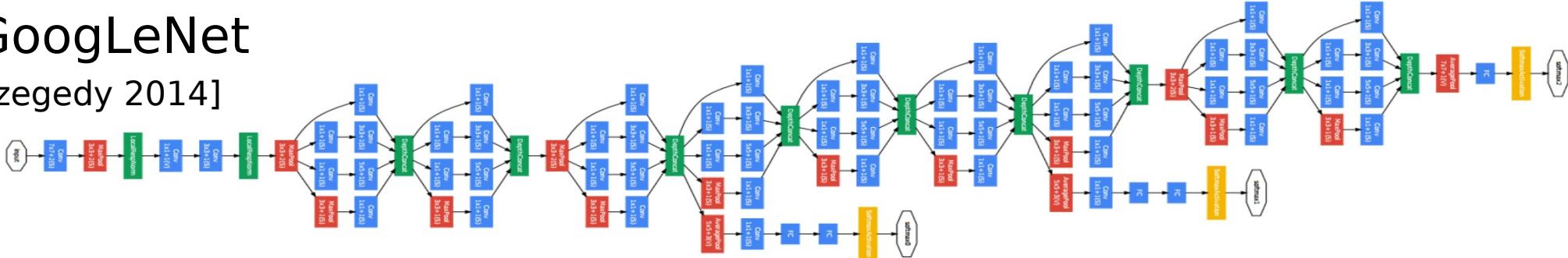
# VGG

[Simonyan 2013]



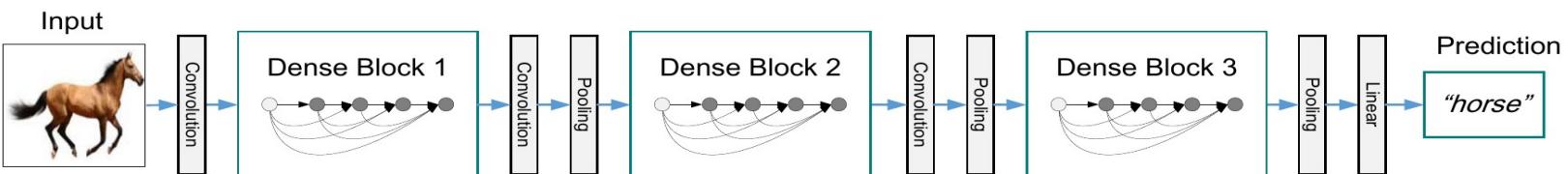
# GoogLeNet

Szegedy 2014]



## ResNet

[He et al. 2015]



# Progress in Computer Vision

► [He 2017]

ALEXNET | 2012



PERSON

MSRA\_2015 | 2015



MASK R-CNN | 2017

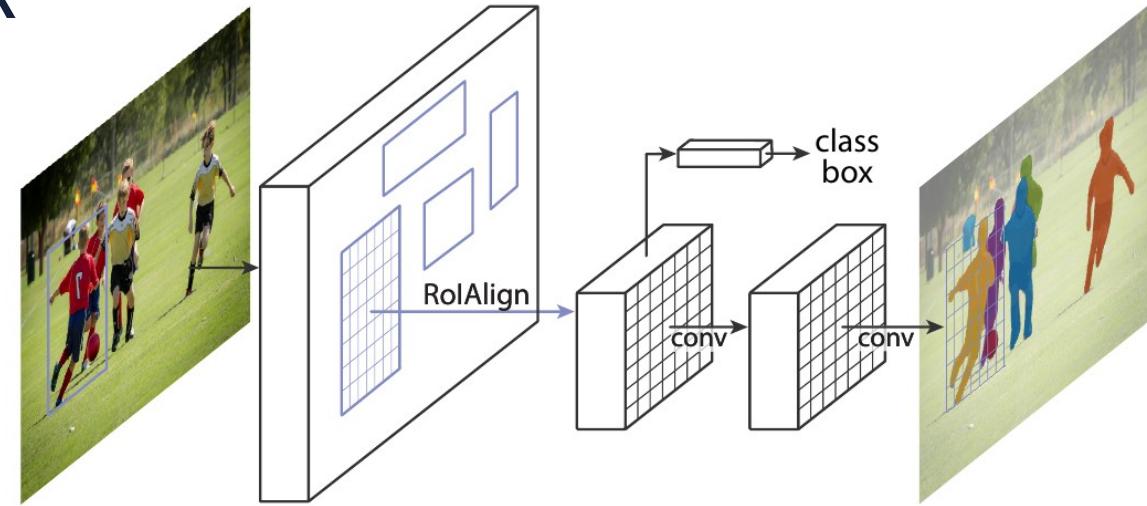


MASK R-CNN | 2017



# Mask R-CNN: instance segmentation

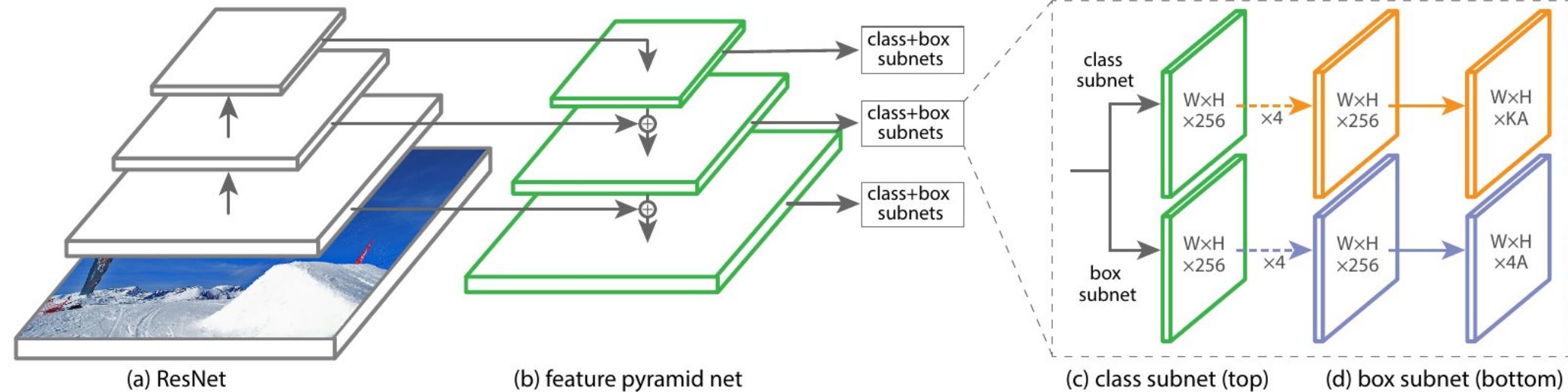
- ▶ [He, Gkioxari, Dollar, Girshick  
arXiv:1703.06870]
- ▶ ConvNet produces an object mask for each region of interest
- ▶ Combined ventral and dorsal pathways



	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
MNC [7]	ResNet-101-C4	24.6	44.3	24.8	4.7	25.9	43.6
FCIS [20] +OHEM	ResNet-101-C5-dilated	29.2	49.5	-	7.1	31.3	50.0
FCIS+++ [20] +OHEM	ResNet-101-C5-dilated	33.6	54.5	-	-	-	-
<b>Mask R-CNN</b>	ResNet-101-C4	33.1	54.9	34.8	12.1	35.6	51.1
<b>Mask R-CNN</b>	ResNet-101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
<b>Mask R-CNN</b>	ResNeXt-101-FPN	<b>37.1</b>	<b>60.0</b>	<b>39.4</b>	<b>16.9</b>	<b>39.9</b>	<b>53.5</b>

# RetinaNet, feature pyramid network

- ▶ One-pass object detection
- ▶ [Lin et al. ArXiv:1708.02002]

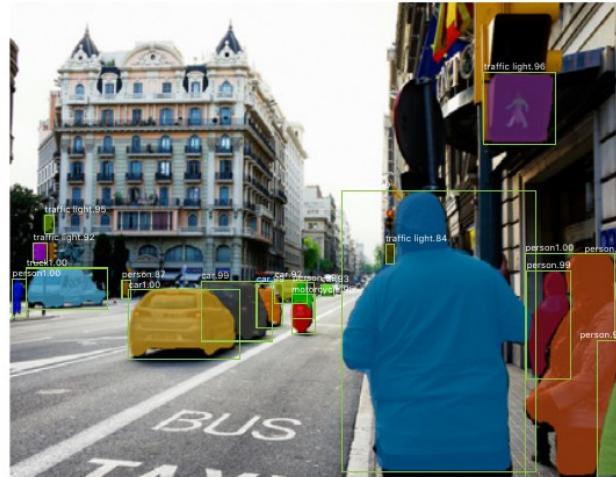
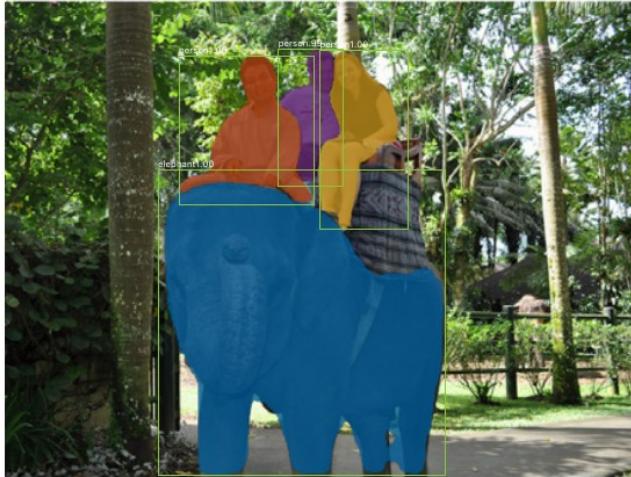
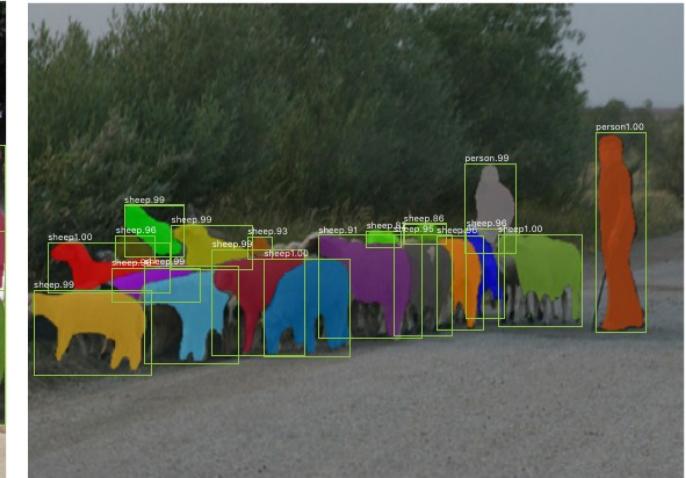
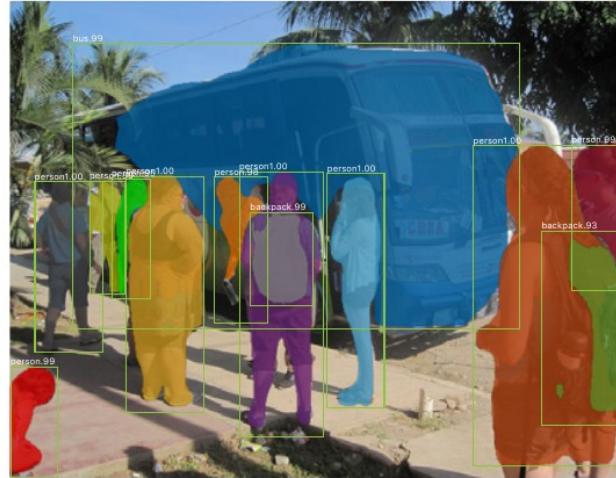
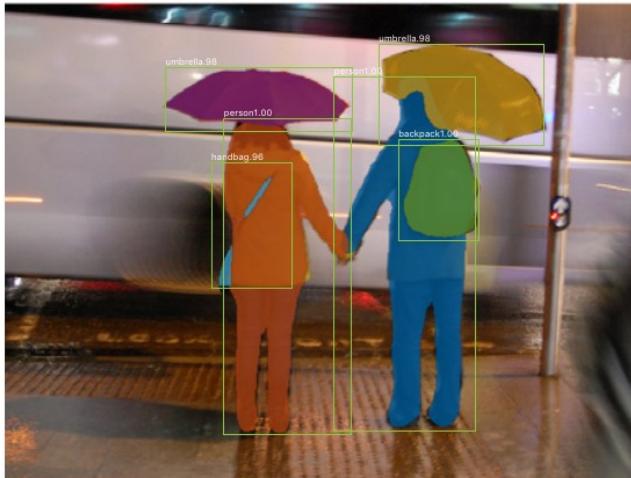


# Mask-RCNN Results on COCO dataset

- ▶ Individual objects are segmented.

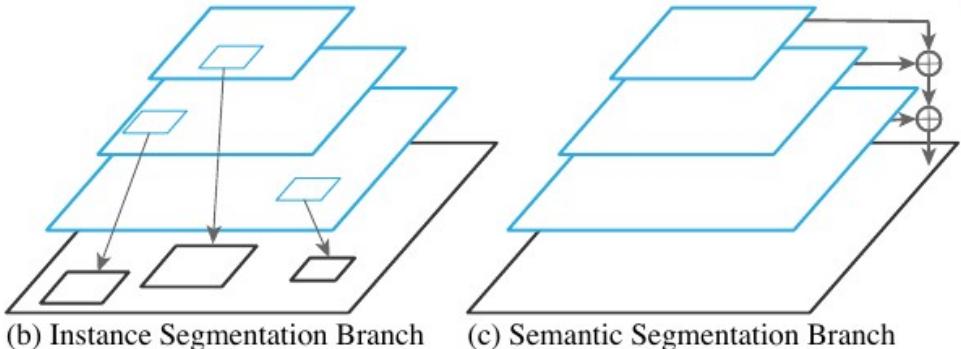
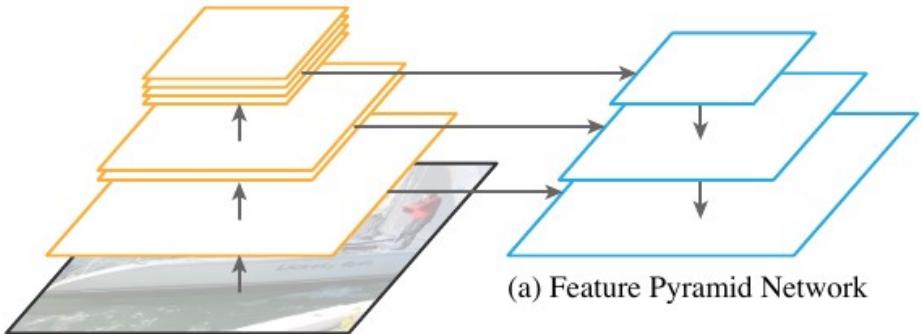


# Mask R-CNN Results on COCO test set



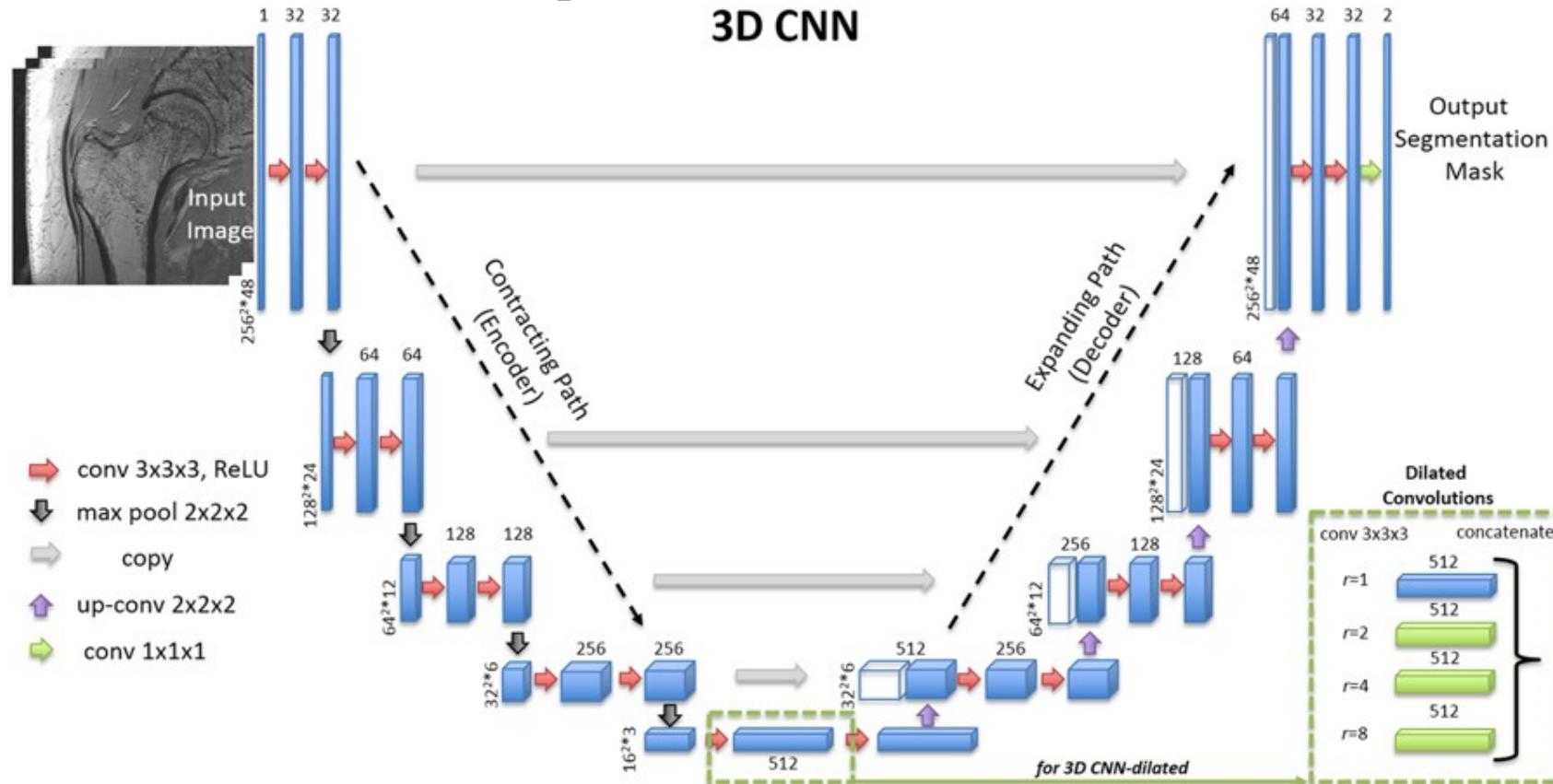
# Panoptic Feature Pyramid Network

- ▶ Segments and recognizes object instances and regions
- ▶ [Kirillov arXiv:1901.0244]

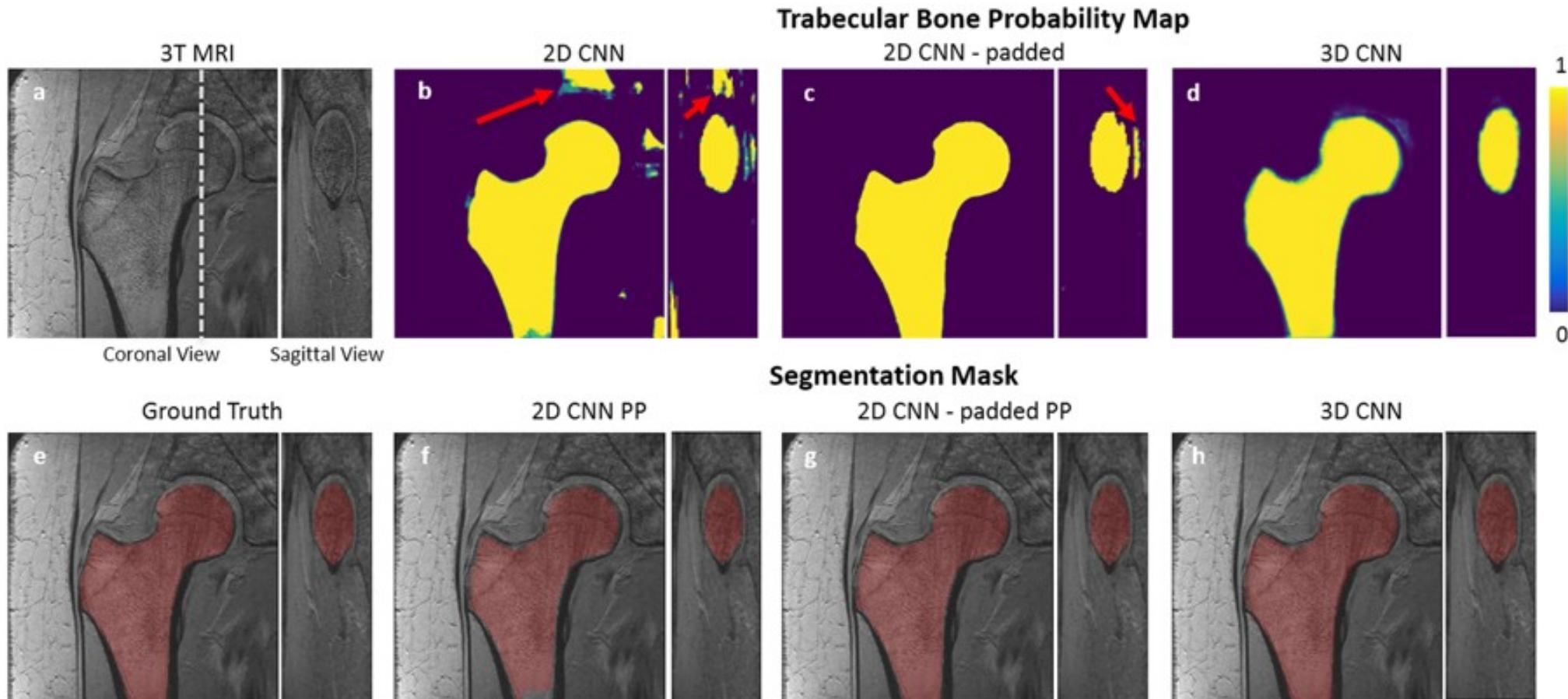


# 3D ConvNet for Medical Image Analysis (NYU)

- ▶ Segmentation Femur from MR Images
  - ▶ [Deniz et al. Nature 2018]

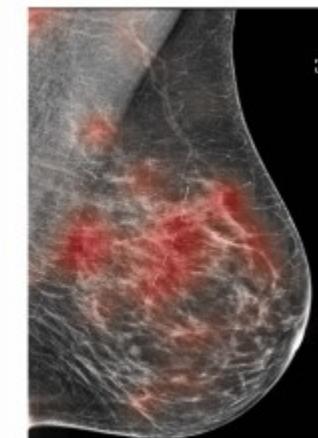
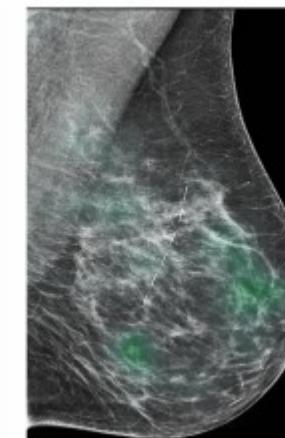
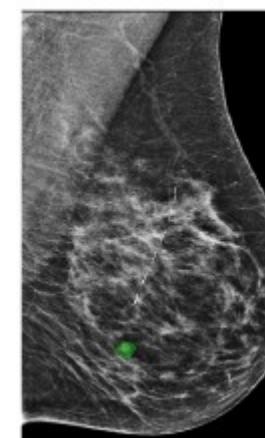
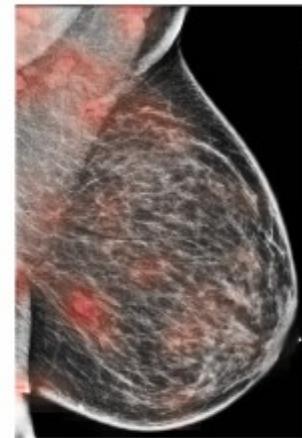
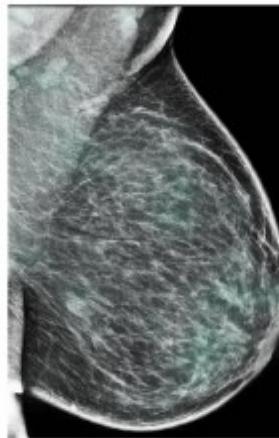
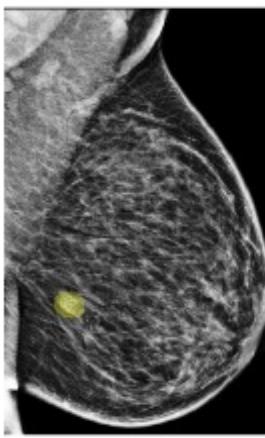
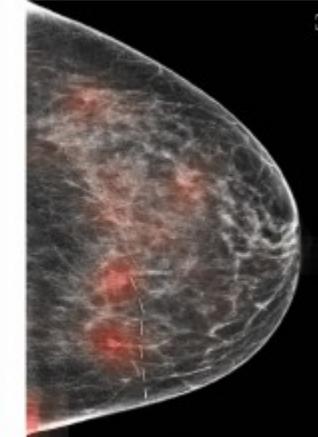
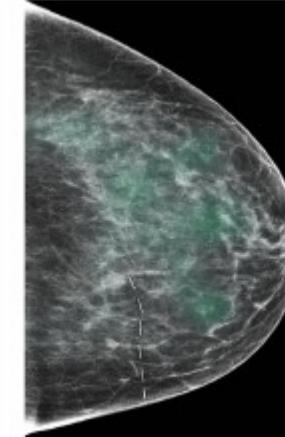
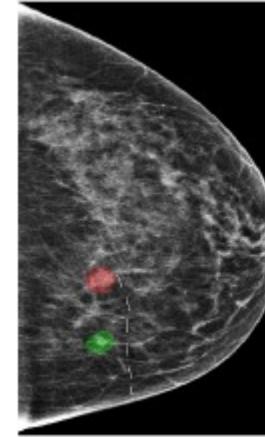
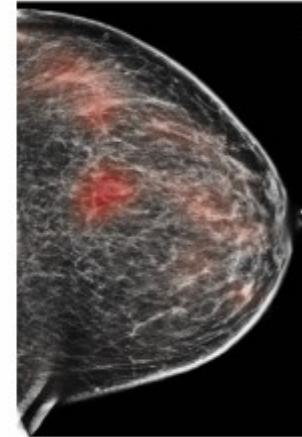
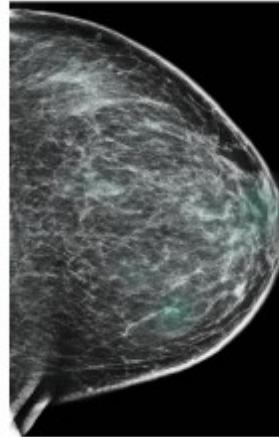
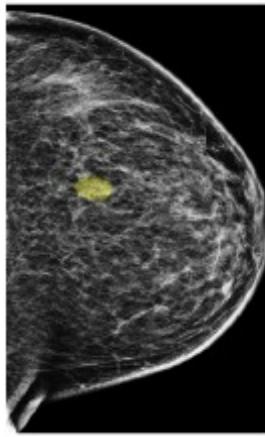


# 3D ConvNet for Medical Image Analysis



# Breast Cancer Detection (NYU)

► [Wu et al. ArXiv:1903.08297] [https://github.com/nyukat/breast\\_cancer\\_classifier](https://github.com/nyukat/breast_cancer_classifier)

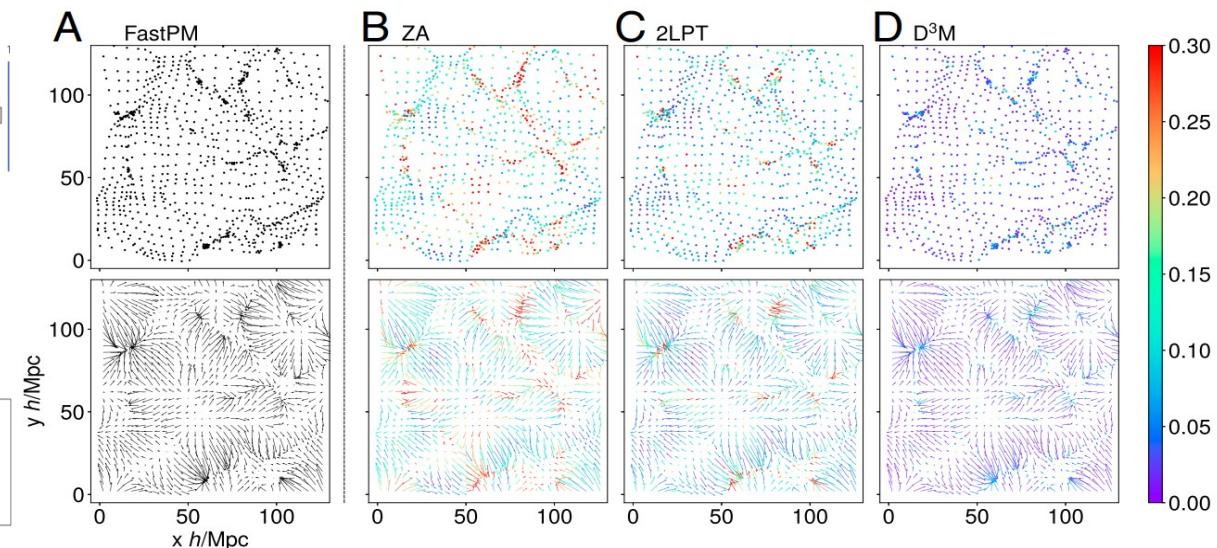
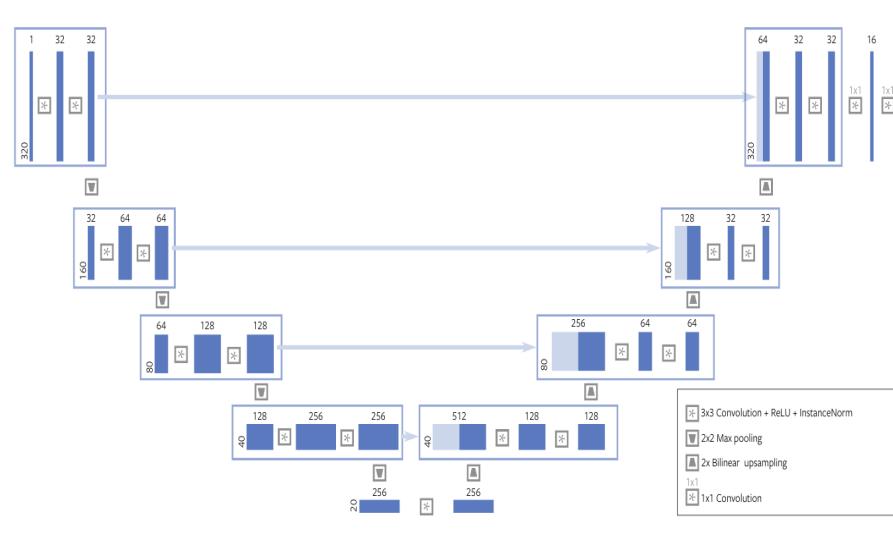


# ConvNets in Astrophysics [He et al. PNAS 07/2019]

## Learning to predict the cosmological structure formation

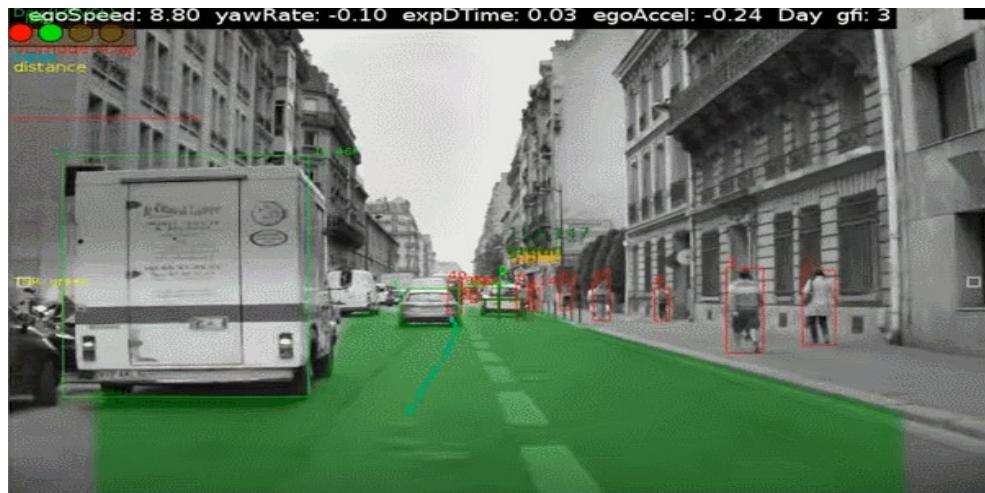
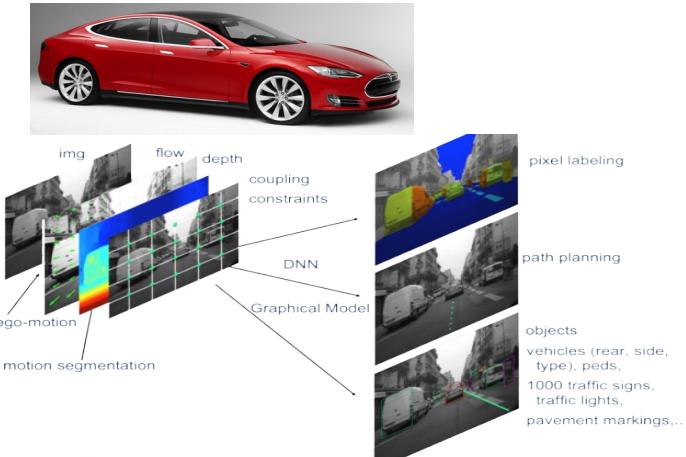
Siyu He<sup>a,b,c,1</sup>, Yin Li<sup>d,e,f</sup>, Yu Feng<sup>d,e</sup>, Shirley Ho<sup>a,b,c,d,e,1</sup>, Siamak Ravanbakhsh<sup>g</sup>, Wei Chen<sup>c</sup>, and Barnabás Póczos<sup>h</sup>

- ▶ 1. Train a coarse-grained 3D U-Net to approximate a fine-grained simulation on a small volume
- ▶ 2. Use it for a simulation on a large volume (the early universe)



# Driving Cars with Convolutional Nets

## ► MobilEye (2015)



## ► NVIDIA



# What are we missing to get to “real” AI?

- ▶ **What we can have**
  - ▶ Safer cars, autonomous cars
  - ▶ Better medical image analysis
  - ▶ Personalized medicine
  - ▶ Adequate language translation
  - ▶ Useful but stupid chatbots
  - ▶ Information search, retrieval, filtering
  - ▶ Numerous applications in energy, finance, manufacturing, environmental protection, commerce, law, artistic creation, games,.....
- ▶ **What we cannot have (yet)**
  - ▶ Machines with common sense
  - ▶ Intelligent personal assistants
  - ▶ “Smart” chatbots”
  - ▶ Household robots
  - ▶ Agile and dexterous robots
  - ▶ Artificial General Intelligence (AGI)



# How do Humans and Animal Learn?

So quickly

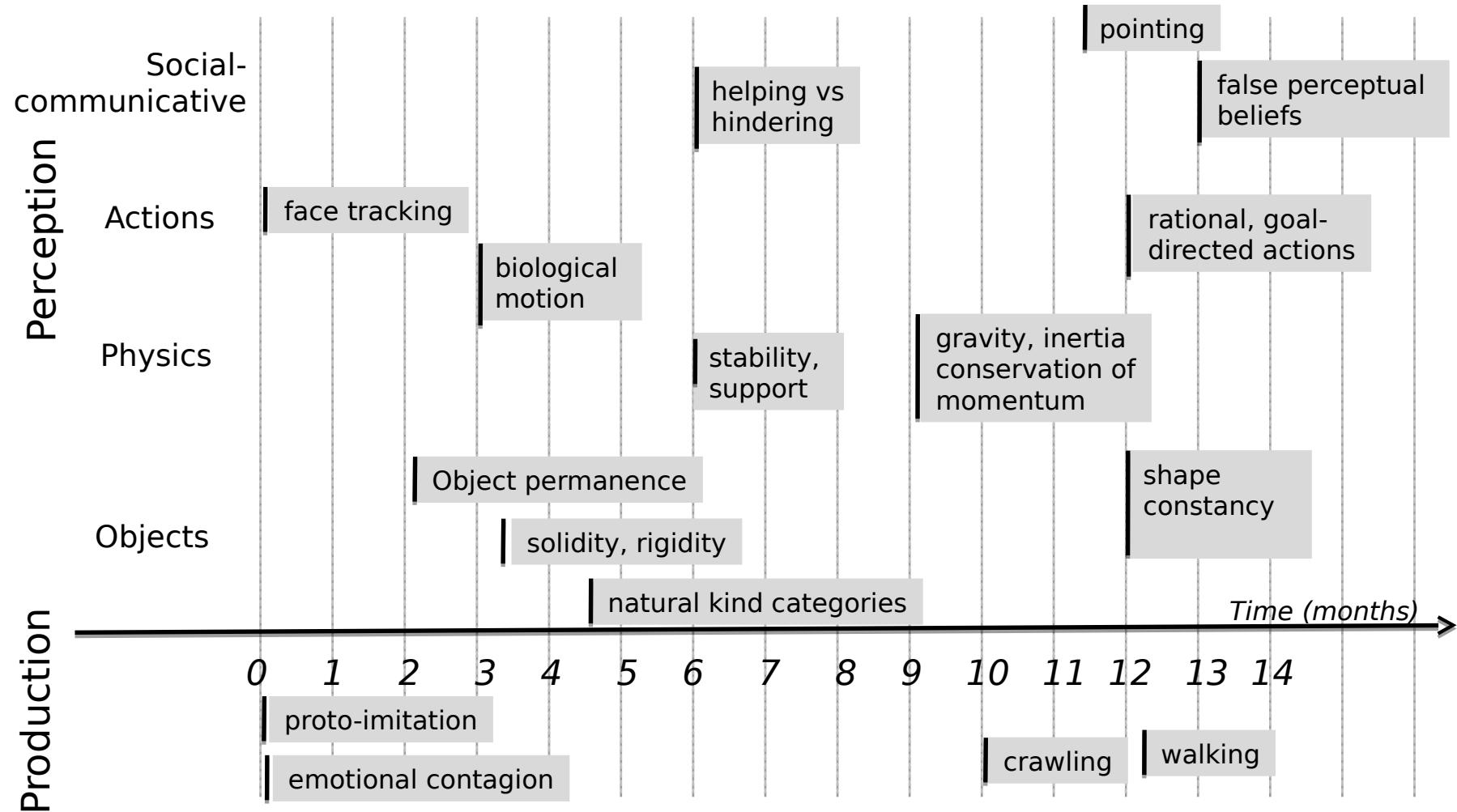
# Babies learn how the world works by observation

- ▶ Largely by observation, with remarkably little interaction.



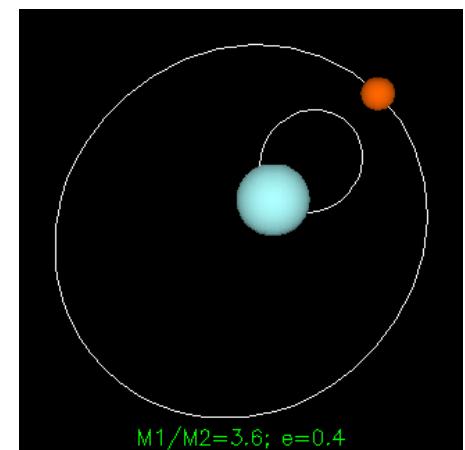
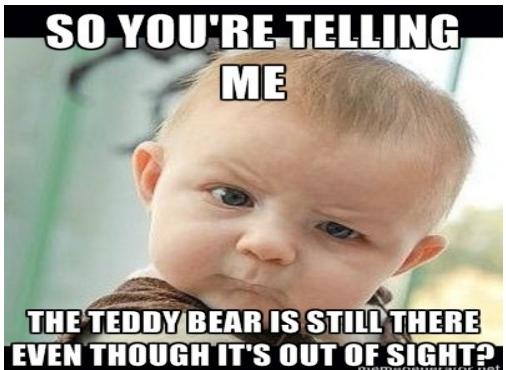
Photos courtesy of  
Emmanuel Dupoux

# Early Conceptual Acquisition in Infants [from Emmanuel Dupoux]



# Prediction is the essence of Intelligence

- We learn models of the world by predicting



# The Next AI Revolution



**THE REVOLUTION  
WILL NOT BE SUPERVISED  
(nor purely reinforced)**



Get the T-shirt!

With thanks to Alyosha Efros and Gil Scott Heron

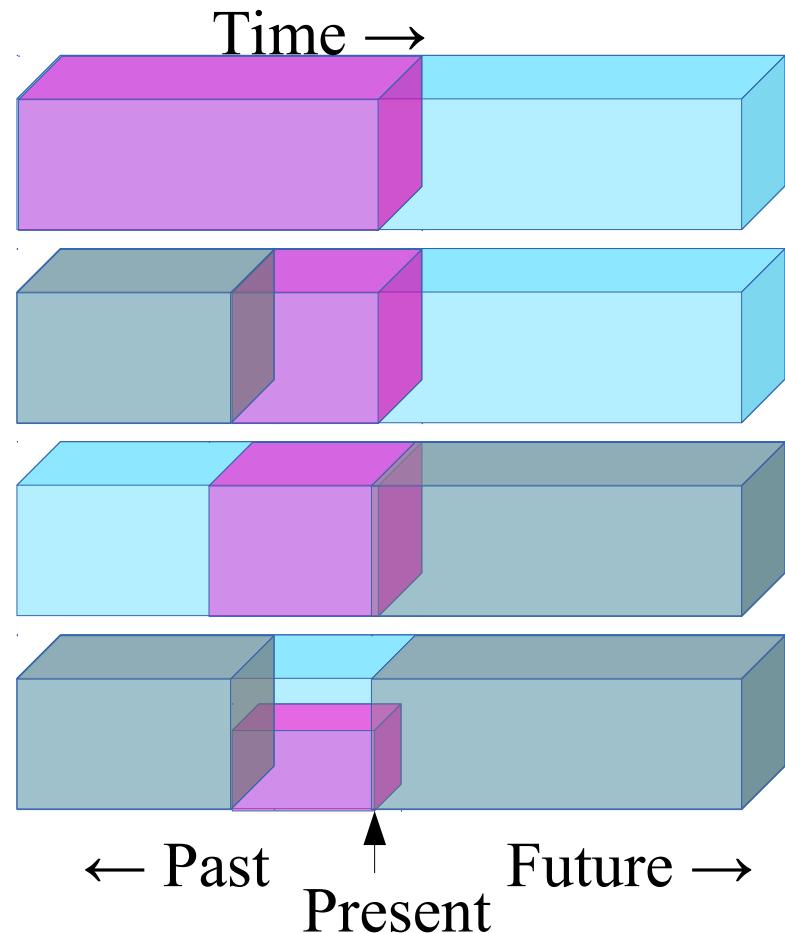


# The Salvation? Self-Supervised Learning

Training very large networks to  
Understand the world through prediction

# Self-Supervised Learning: Prediction & Reconstruction

- ▶ Predict any part of the input from any other part.
- ▶ Predict the **future** from the **past**.
- ▶ Predict the **future** from the **recent past**.
- ▶ Predict the **past** from the **present**.
- ▶ Predict the **top** from the **bottom**.
- ▶ Predict the **occluded** from the **visible**
- ▶ Pretend there is a part of the input you don't know and predict that.



# Three Types of Learning

## ► Reinforcement Learning

- The machine predicts a scalar reward given once in a while.

## ► weak feedback

## ► Supervised Learning

- The machine predicts a category or a few numbers for each input

## ► medium feedback

## ► Self-supervised Predictive Learning

- The machine predicts any part of its input for any observed part.

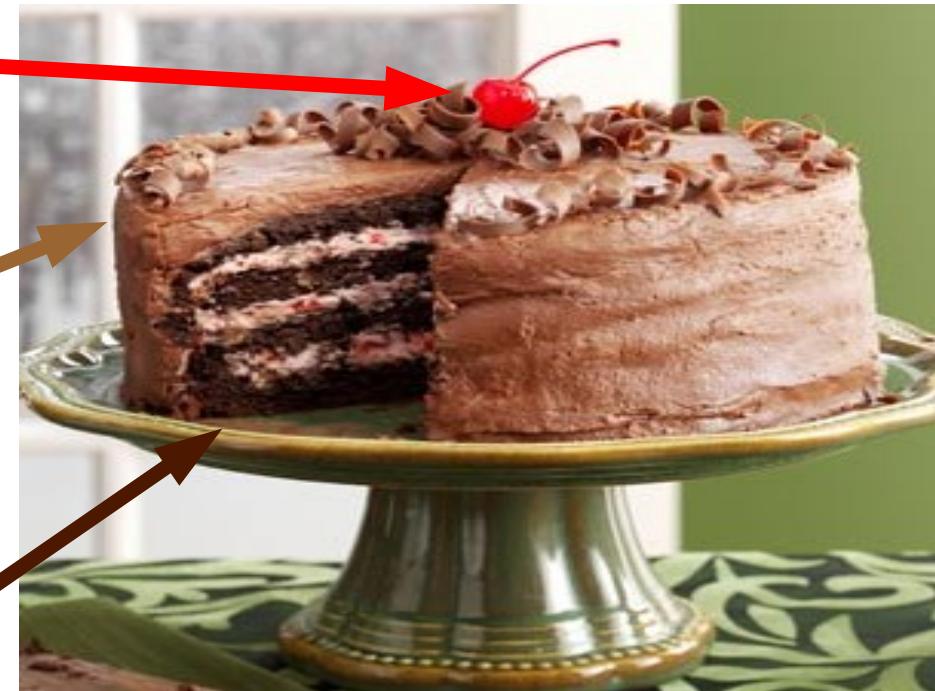
- Predicts future frames in videos

## ► A lot of feedback



# How Much Information is the Machine Given during Learning?

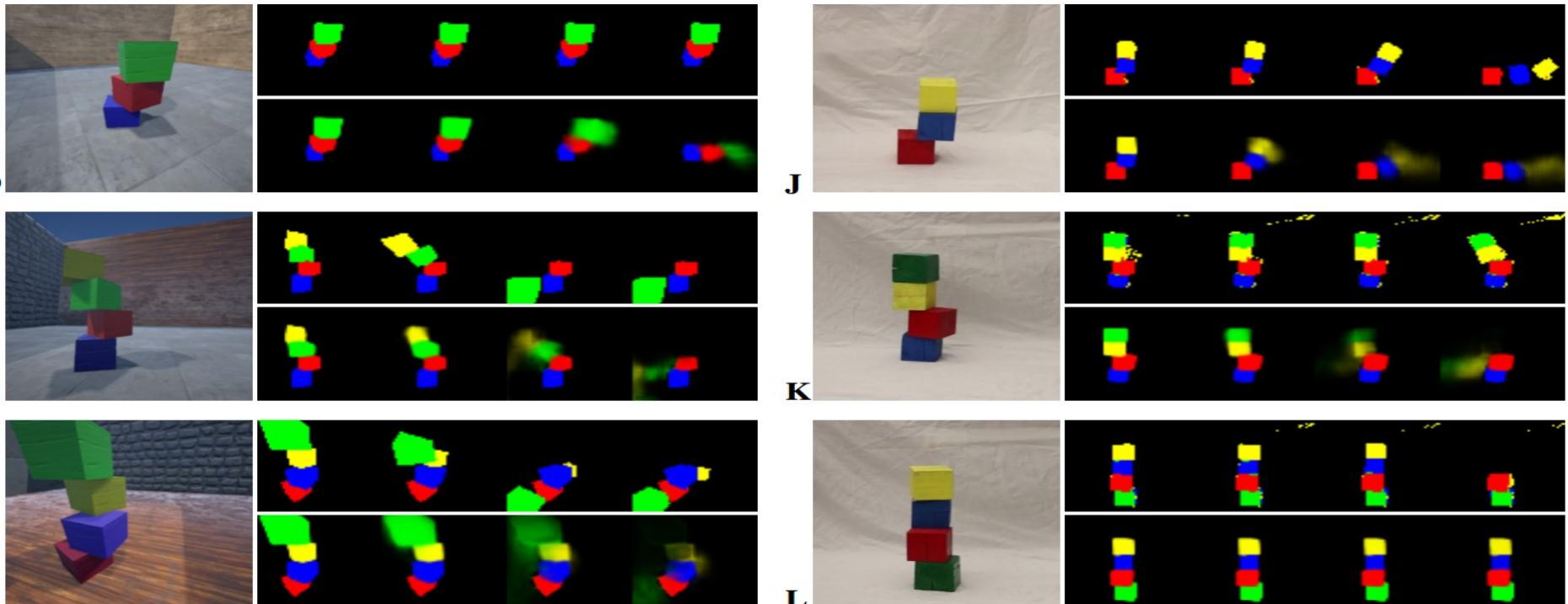
- ▶ “Pure” Reinforcement Learning (**cherry**)
  - ▶ The machine predicts a scalar reward given once in a while.
  - ▶ **A few bits for some samples**
  
- ▶ Supervised Learning (**icing**)
  - ▶ The machine predicts a category or a few numbers for each input
  - ▶ Predicting human-supplied data
  - ▶ **10→10,000 bits per sample**
  
- ▶ Self-Supervised Learning (**cake génoise**)
  - ▶ The machine predicts any part of its input for any observed part.
  - ▶ Predicts future frames in videos
  - ▶ **Millions of bits per sample**



# PhysNet: tell me what happens next

[Lerer, Gross, Fergus ICML 2016, arxiv:1603.01312]

- ▶ ConvNet produces object masks that predict the trajectories of falling blocks. **Blurry predictions when uncertain**



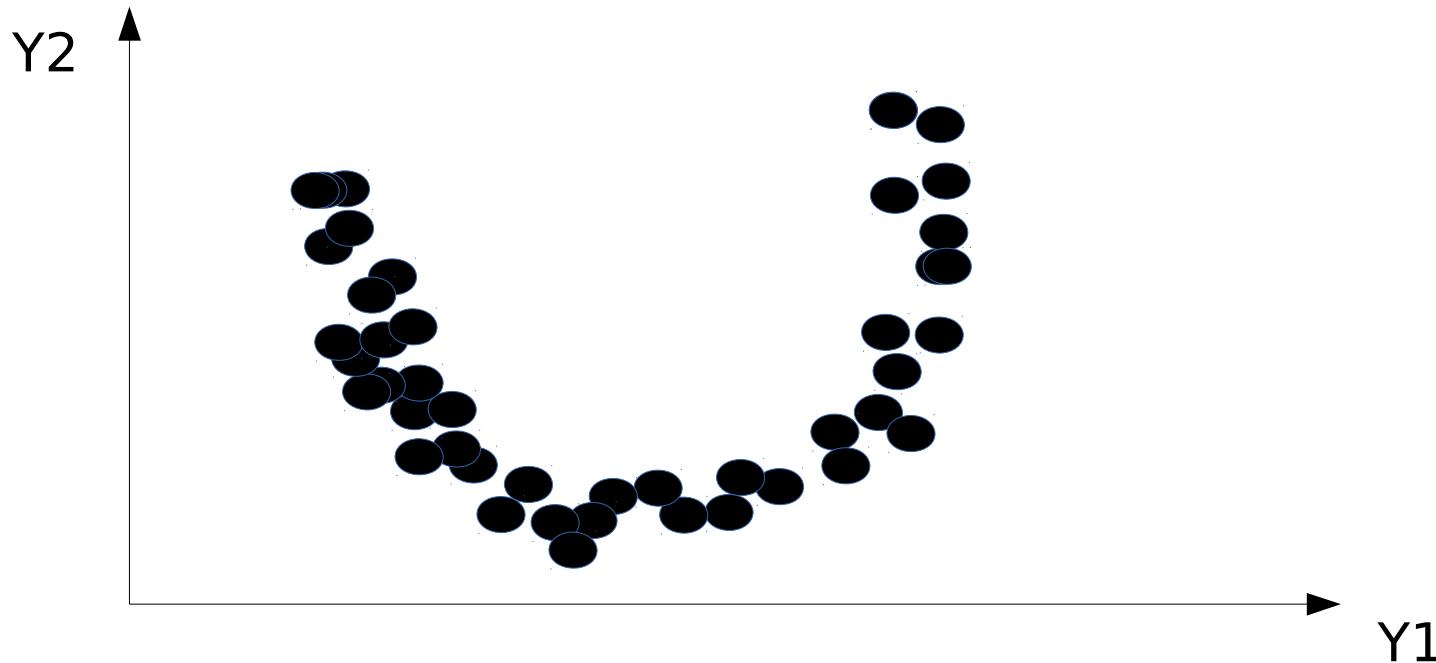


# Energy-Based Learning

Energy Shaping  
Regularized Auto-Encoders

# Energy-Based Unsupervised Learning

- Learning an **energy function** (or contrast function)  $F(Y)$ , that takes
  - ▶ Low values on the data manifold
  - ▶ Higher values everywhere else

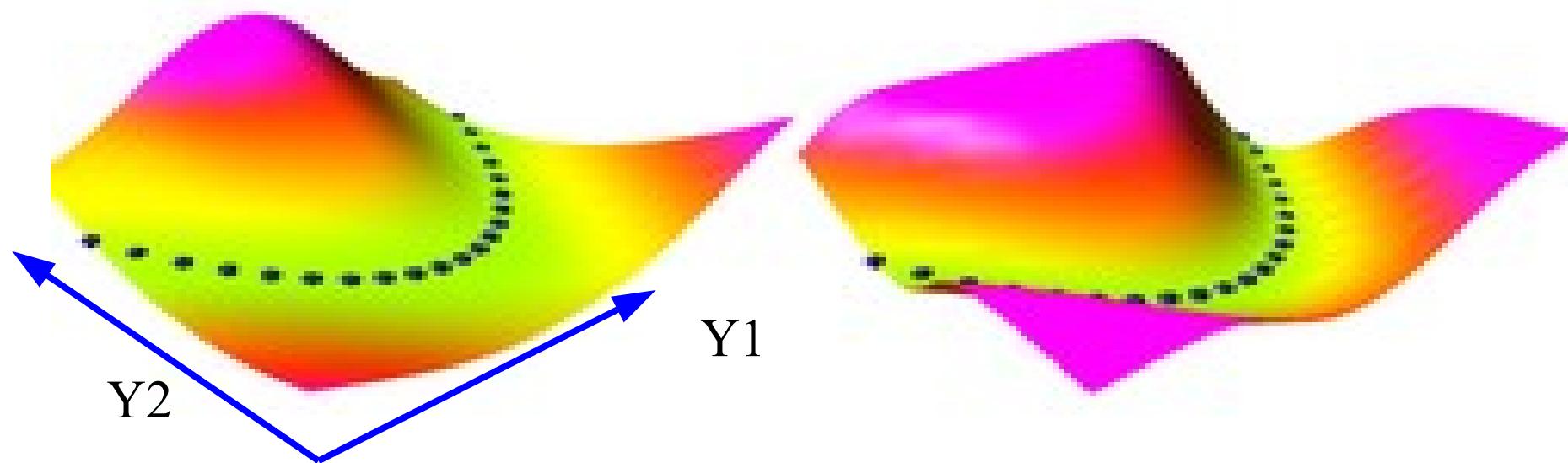




# Capturing Dependencies Between Variables with an Energy Function

- 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

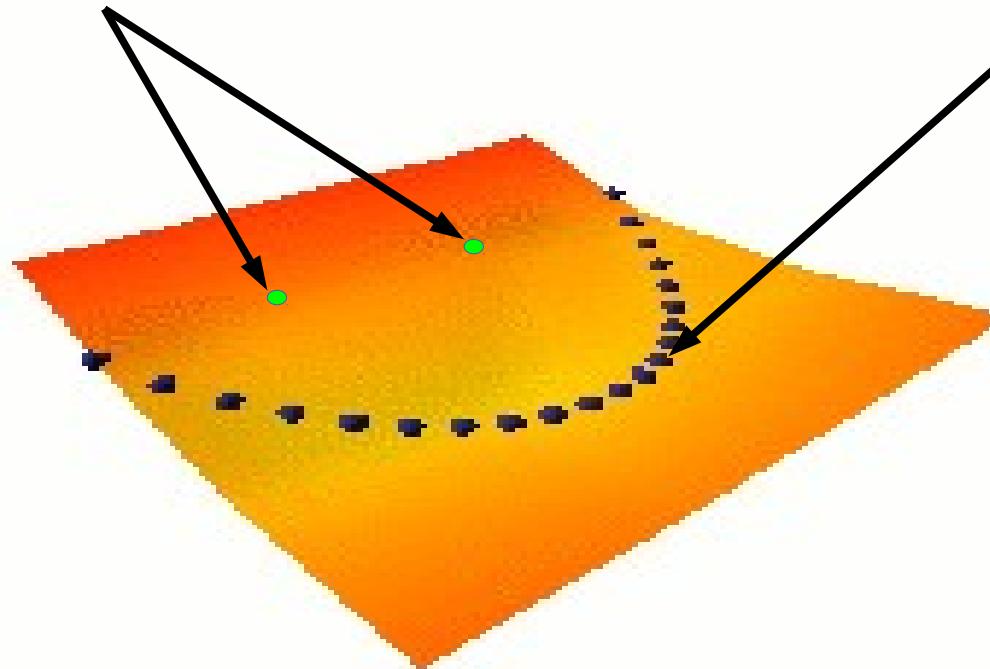
$$Y_2 = (Y_1)^2$$



# 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?**

Implausible  
futures  
(high energy)



Plausible  
futures  
(low energy)

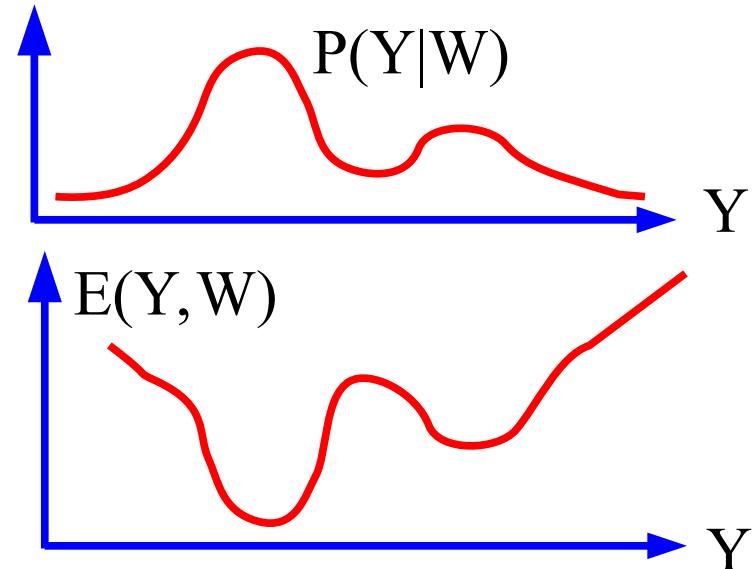


# Transforming Energies into Probabilities (if necessary)

- The energy can be interpreted as an unnormalized negative log density
- Gibbs distribution: Probability proportional to  $\exp(-\text{energy})$ 
  - ▶ Beta parameter is akin to an inverse temperature
- Don't compute probabilities unless you absolutely have to
  - ▶ Because the denominator is often intractable

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

$$E(Y, W) \propto -\log P(Y|W)$$

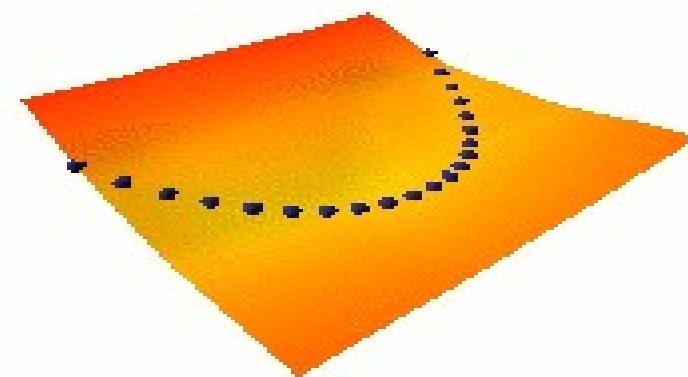
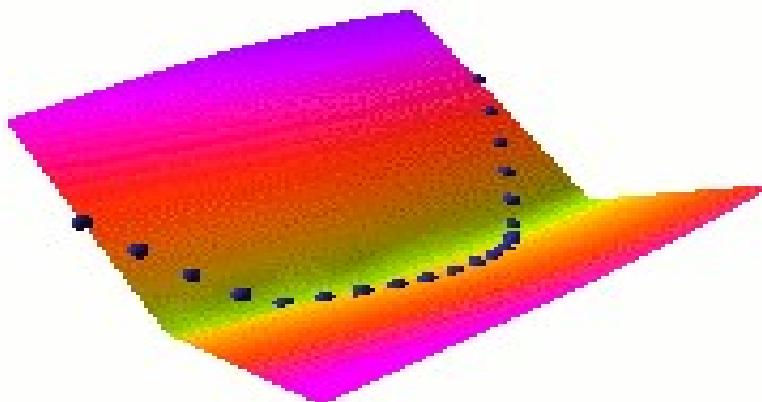




# Learning the Energy Function

## ■ parameterized energy function $F(Y, W)$

- ▶ 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?

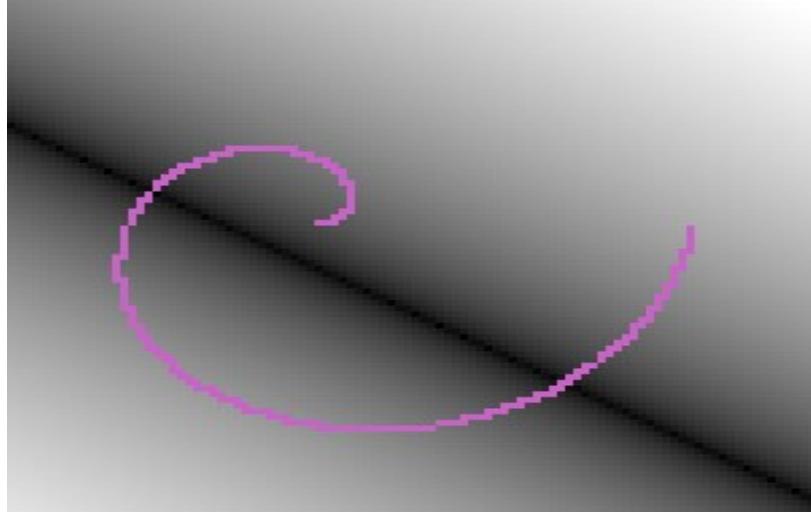


# Energy surface for PCA and K-means

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

PCA

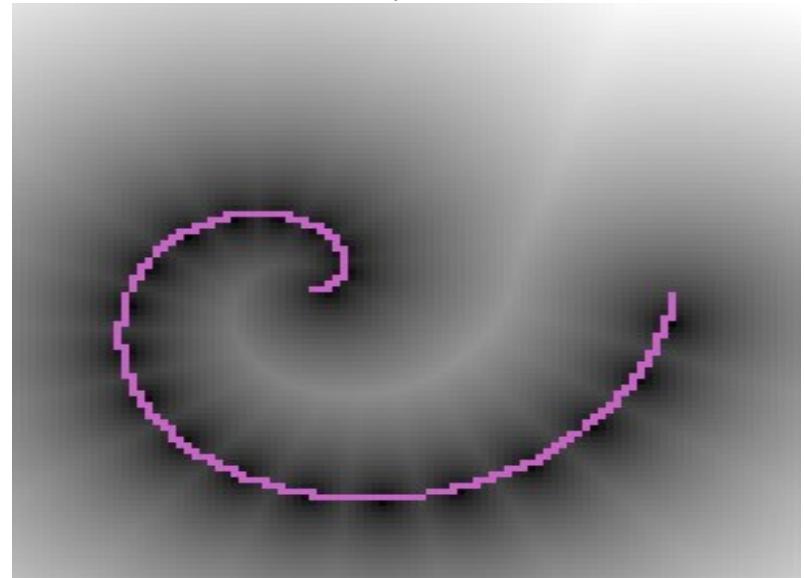
$$F(Y) = \|W^T W Y - Y\|^2$$



K-Means,

Z constrained to 1-of-K code

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



# Seven Strategies to Shape the Energy Function

MOST IMPORTANT SLIDE

- ▶ 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 or variational approximation)
- ▶ 3. push down of the energy of data points, push up on chosen locations
  - ▶ Contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Min Probability Flow, adversarial generator/GANs
- ▶ 4. minimize the gradient and maximize the curvature around data points score matching
- ▶ 5. if  $F(Y) = \|Y - G(Y)\|^2$ , make  $G(Y)$  as "constant" as possible.
  - ▶ Contracting auto-encoder, saturating auto-encoder
- ▶ 6. train a dynamical system so that the dynamics goes to the data manifold
  - ▶ denoising auto-encoder, masked auto-encoder (e.g. BERT)
- ▶ 7. use a regularizer that limits the volume of space that has low energy
  - ▶ Sparse coding, sparse auto-encoder, LISTA & PSD, Variational auto-encoders



push down of the energy of data points, push up everywhere else



## Max likelihood (requires a tractable partition function)

Maximizing  $P(Y|W)$  on training samples

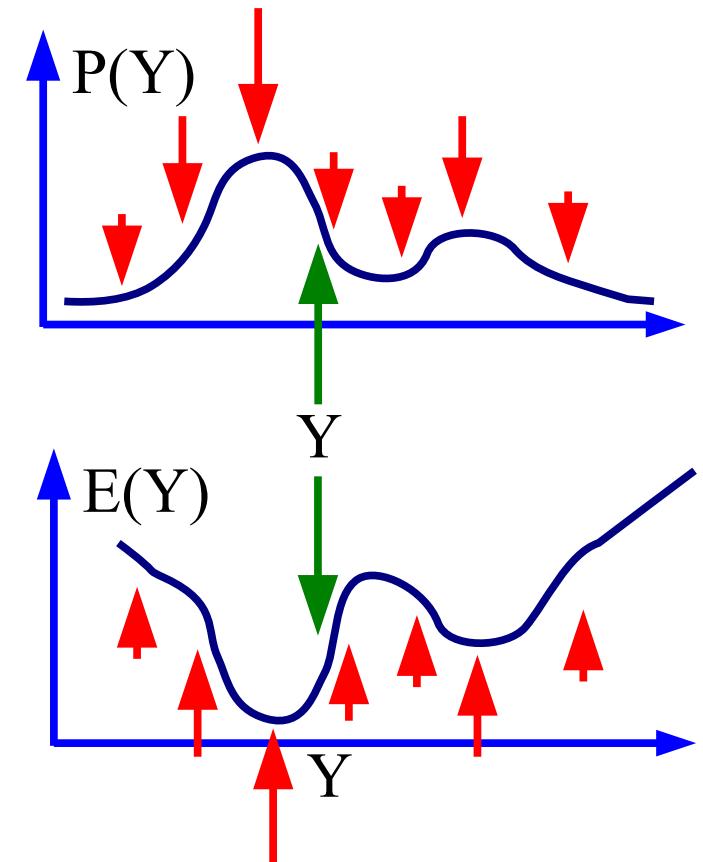
$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

make this big  
make this small

Minimizing  $-\log P(Y, W)$  on training samples

$$L(Y, W) = E(Y, W) + \frac{1}{\beta} \log \int_y e^{-\beta E(y,W)}$$

make this small  
make this big





push down of the energy of data points, push up everywhere else

Gradient of the negative log-likelihood loss for one sample Y:

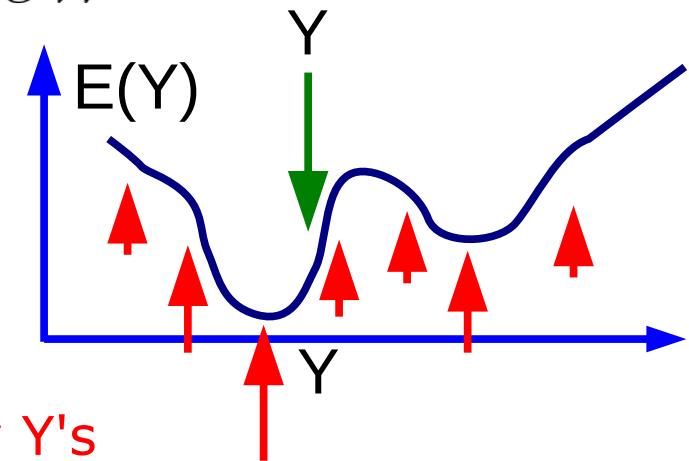
$$\frac{\partial L(Y, W)}{\partial W} = \frac{\partial E(Y, W)}{\partial W} - \int_y P(y|W) \frac{\partial E(y, W)}{\partial W}$$

Gradient descent:

$$W \leftarrow W - \eta \frac{\partial L(Y, W)}{\partial W}$$

Pushes down on the  
energy of the samples

Pulls up on the  
energy of low-energy Y's



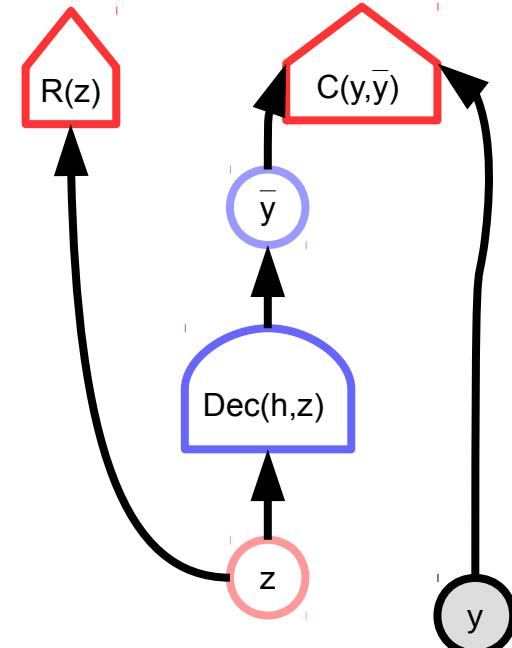
$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \int_y P(y|W) \frac{\partial E(y, W)}{\partial W}$$



# Latent Variable Models

## Sparse Modeling, Regularized Auto-Encoders

# Sparse Coding & Regularized Auto-Encoders

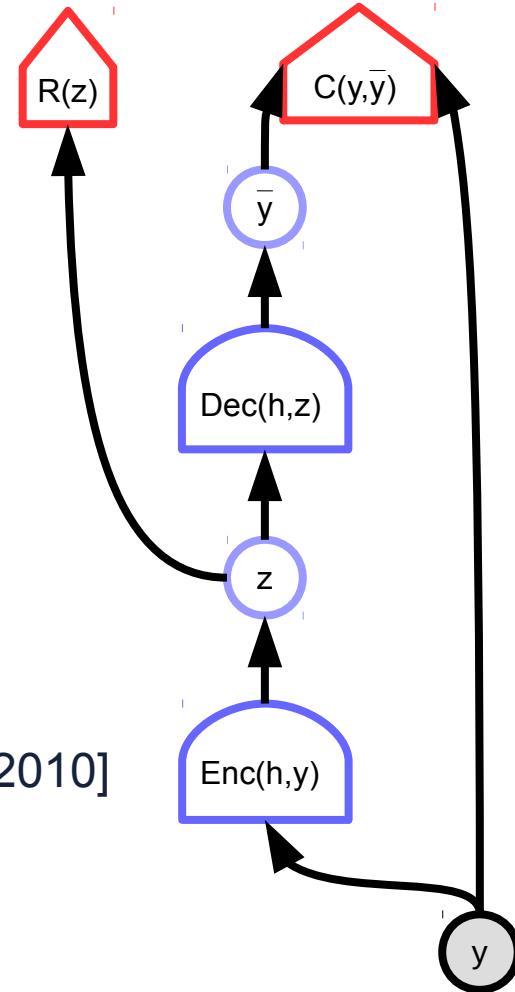


- ▶  $y$ : variable to be predicted
- ▶  $h$ : encoding of  $x$
- ▶  $z$ : latent variable
- ▶ Sparse modeling
- ▶  $E(y, z) = C(y, \text{Dec}(z)) + R(z)$
- ▶  $z^* = \text{argmin } E(y, z)$

- ▶ Sparse Auto-Encoder
- ▶  $E(y, z) = C(y, \text{Dec}(\text{Enc}(y))) + R(z)$
- ▶ [Kavukcuoglu CVPR 2009] [Gregor ICML 2010]

$$F(Y) = \min_z E(Y, Z)$$

$$E(Y, Z) = \|Y - \sum_k W_k Z_k\|^2 + \alpha \sum_k |Z_k|$$

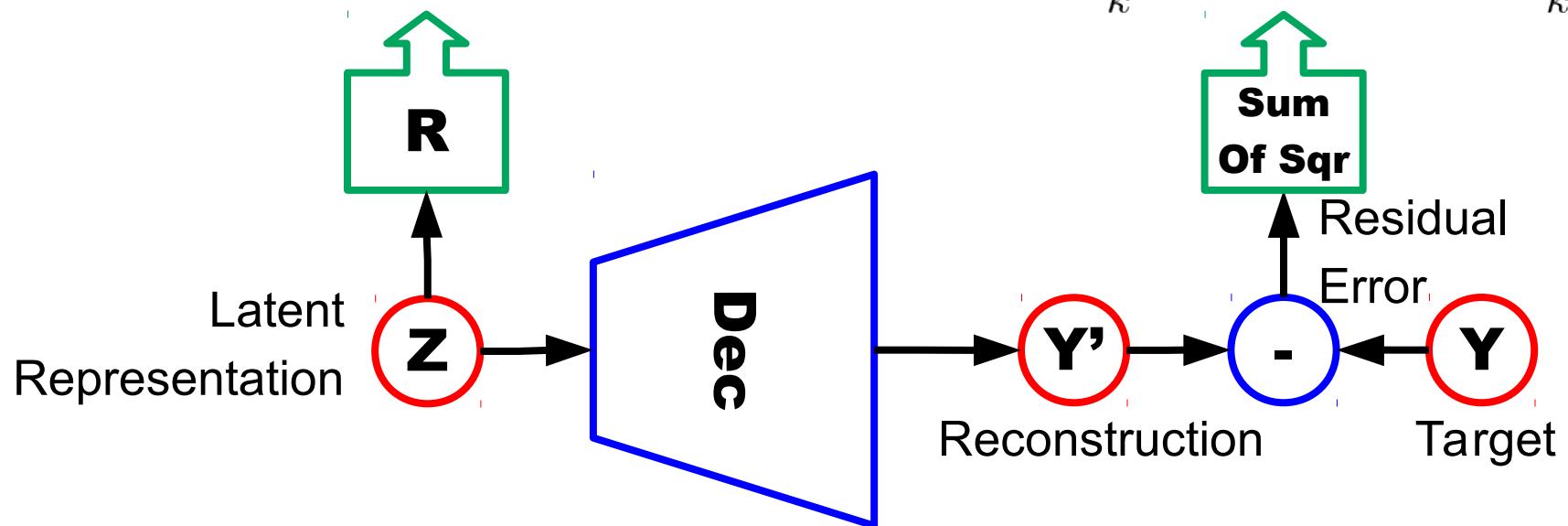


# The “Decoder with Regularized Latent Variable” Model

- ▶  $\mathbf{Y}' = \text{Dec}(\mathbf{Z})$      $\mathbf{Z}^* = \text{argmin} \|\mathbf{Y} - \text{Dec}(\mathbf{Z})\| + R(\mathbf{Z})$
- ▶ Linear decoder: K-Means, basis pursuit, K-SVD, sparse coding,....
- ▶ Sparse modeling: [Olshausen Field 1997]

$$F(Y) = \min_z E(Y, Z)$$

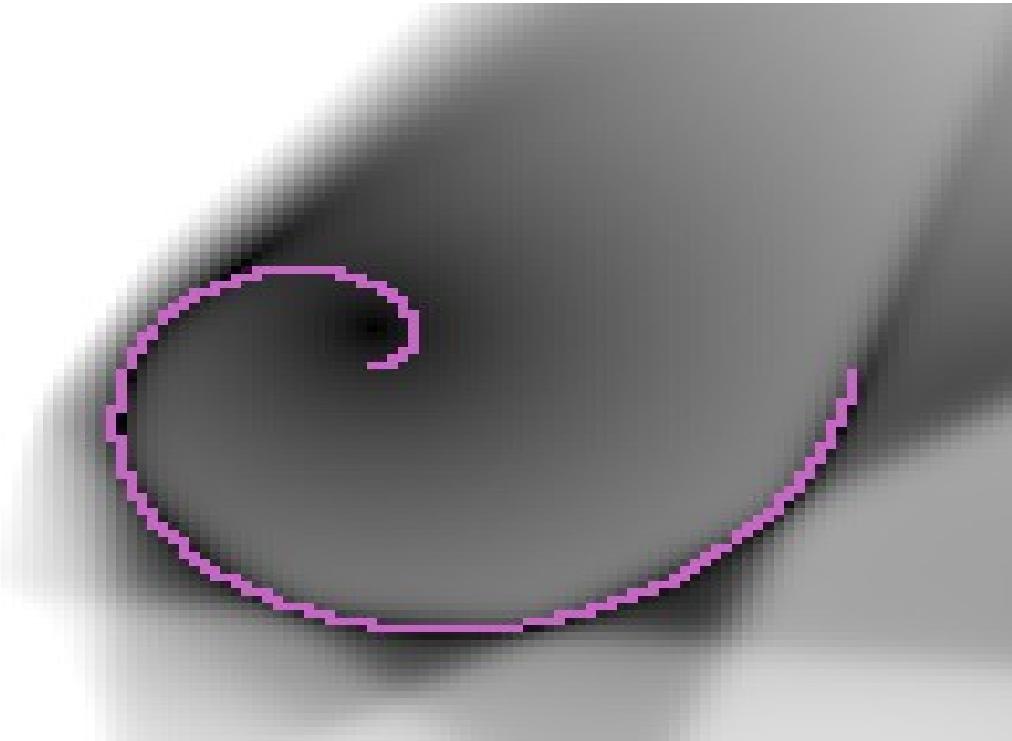
$$E(Y, Z) = \left\| Y - \sum_k W_k Z_k \right\|^2 + \alpha \sum_k |Z_k|$$



# Energy Surface for Sparse Modeling

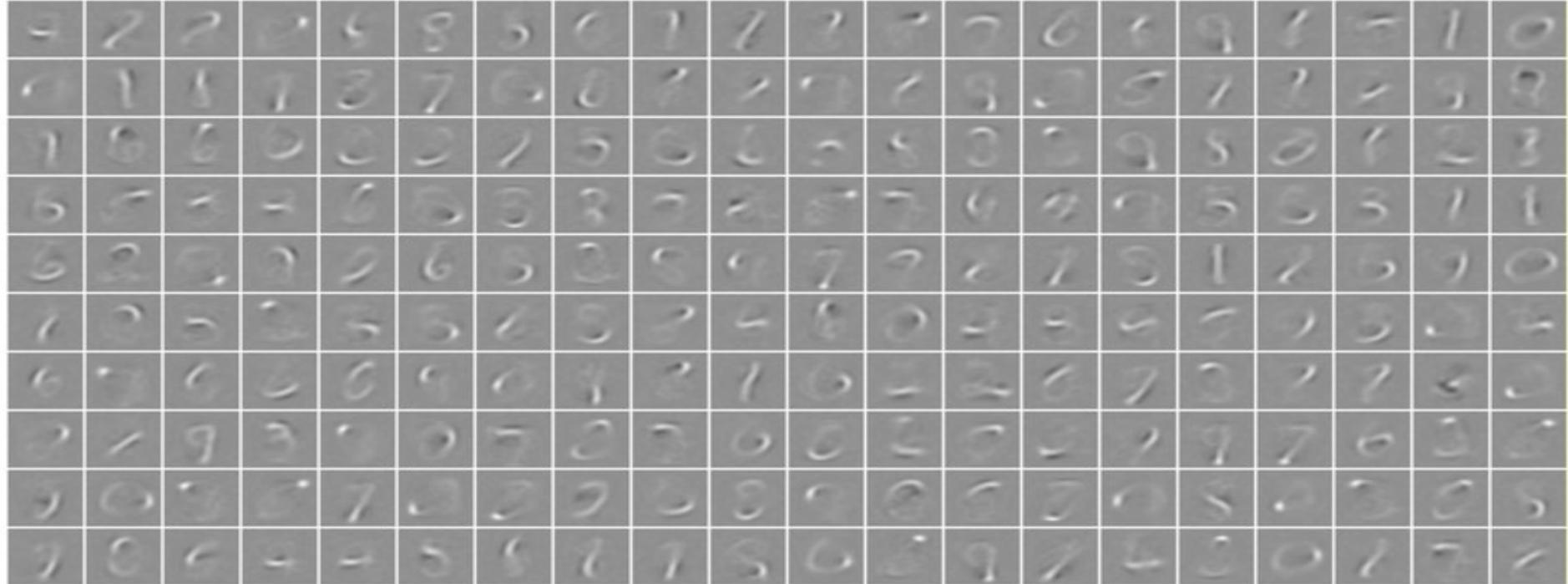


**Sparse coding, sparse auto-encoder, Predictive Sparse Decomposition**



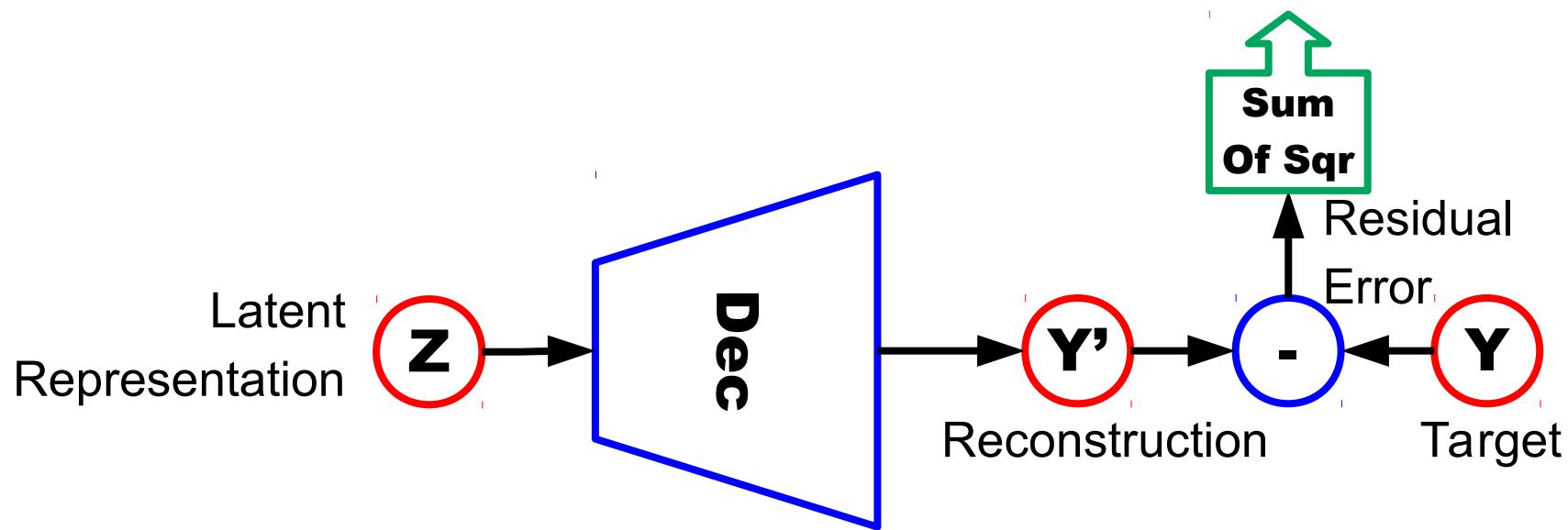
# Sparse Modeling on handwritten digits (MNIST)

- Basis functions (columns of decoder matrix) are digit parts
- All digits are a linear combination of a small number of these



# Generation through Latent Optimization

- ▶ [Bojanowski, Joulin, Lopez-Paz, Szlam arxiv:1707.05776]
- ▶  $\mathbf{Y}' = \text{Dec}(\mathbf{Z})$        $\mathbf{Z}^* = \operatorname{argmin} \|\mathbf{Y} - \text{Dec}(\mathbf{Z})\|^2$
- ▶ **Z is not explicitly regularized, just low-dimensional**

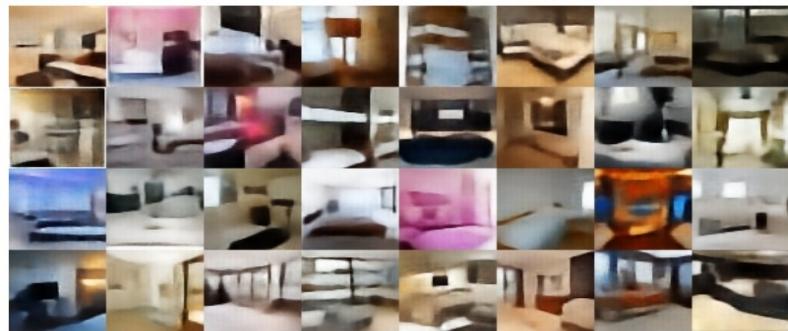


# Generation through Latent Optimization

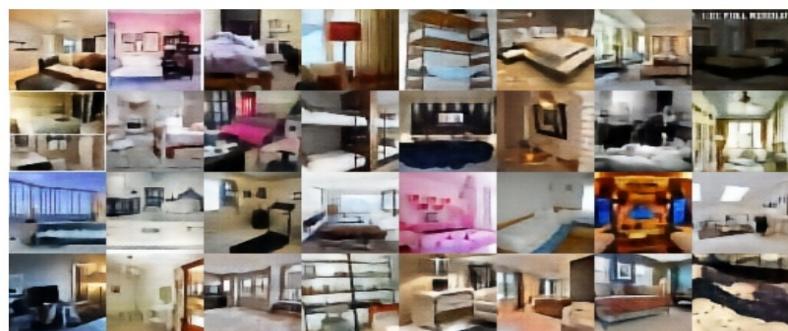
► Original



► Reconstr.  
 $D=100$



► Reconstr  
 $D=512$



# Generation through Latent Optimization

[Bojanowski, Joulin, Lopez-Paz, Szlam arxiv:1707.05776]

- ▶ **Interpolation  
in Z space**



- ▶ **Interpolation  
in pixels**



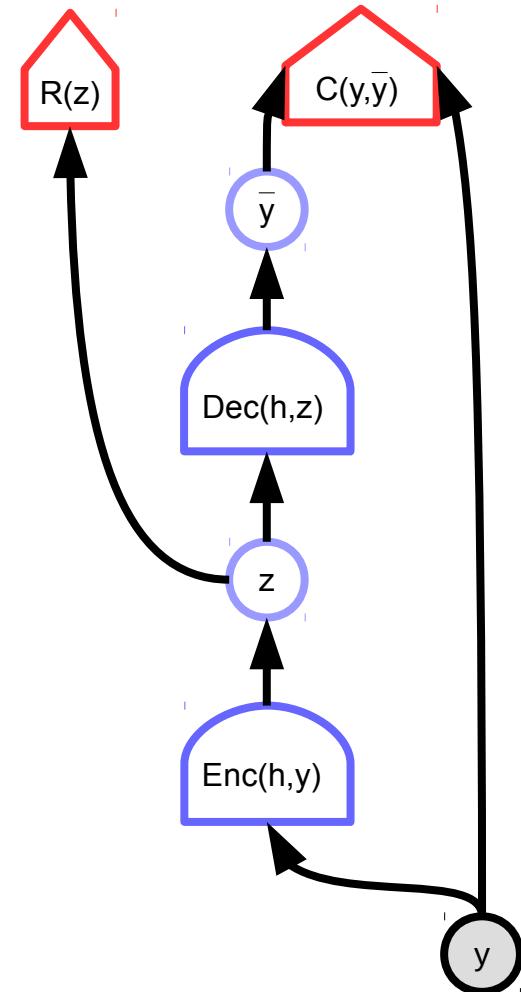


# Sparse Auto-Encoders

Computing the sparse code efficiently

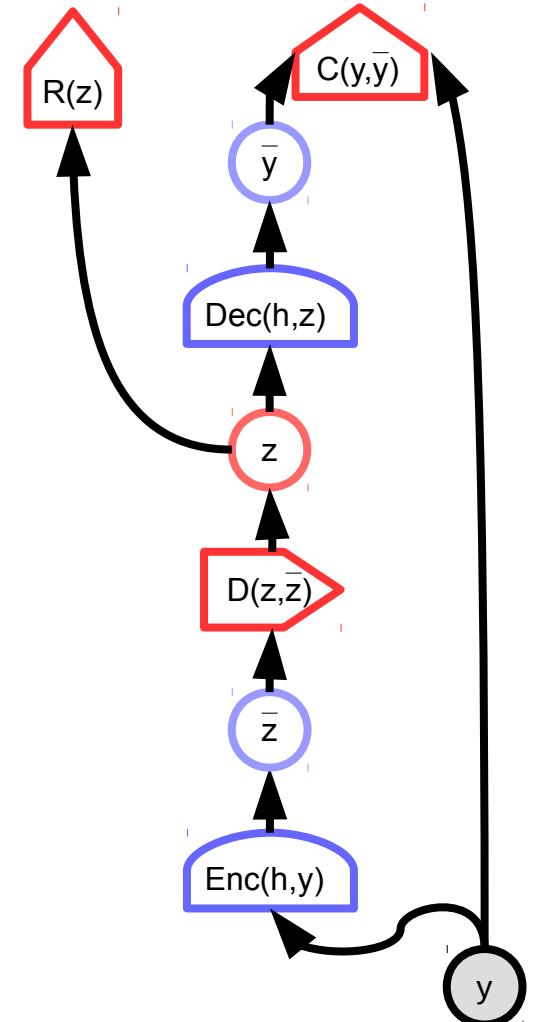
# Regularized Auto-Encoders

- ▶ **y: variable to be predicted**
  - ▶ **h: encoding of x**
  - ▶ **z: internal representation**
- 
- ▶ **Sparse Auto-Encoder**
    - ▶  $E(y,z) = C(y, \text{Dec}(\text{Enc}(y))) + R(z)$
    - ▶  $F(y) = \min\{z\} E(y,z)$



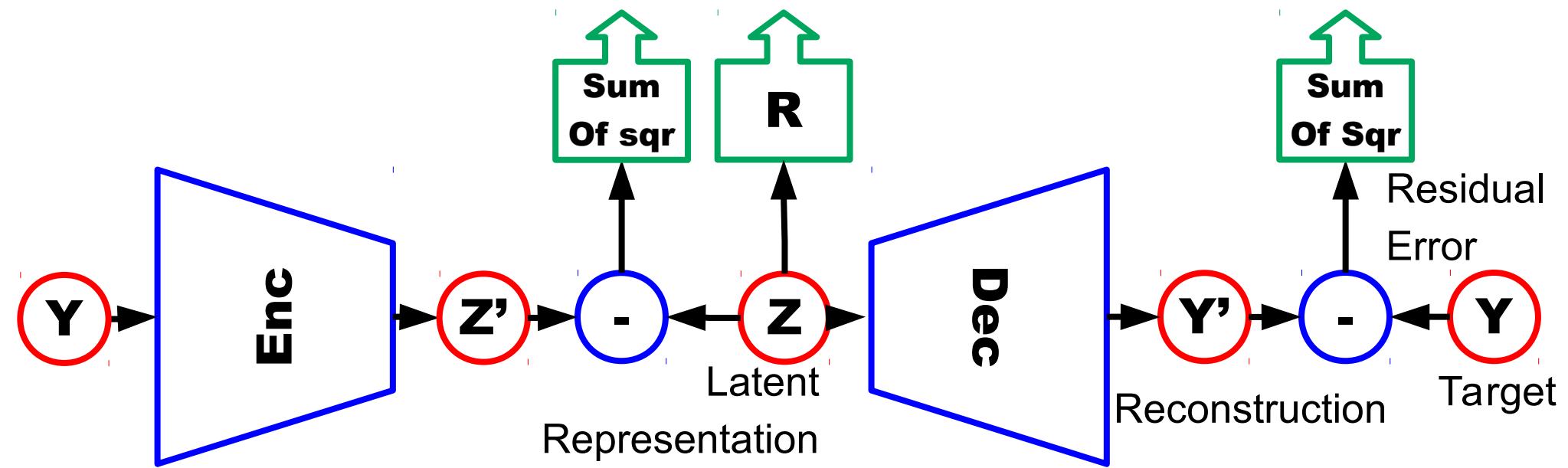
# Regularized Auto-Encoders

- ▶ **y: variable to be predicted**
- ▶ **h: encoding of x**
- ▶ **z: latent variable**
  
- ▶ **Sparse Auto-Encoder with latent variable**
  - ▶  $E(y,z) = C[y, \text{Dec}(\text{Enc}(y))] + D[z, \text{Enc}(y)] + R(z)$
  - ▶ [Kavukcuoglu CVPR 2009] [Gregor ICML 2010]



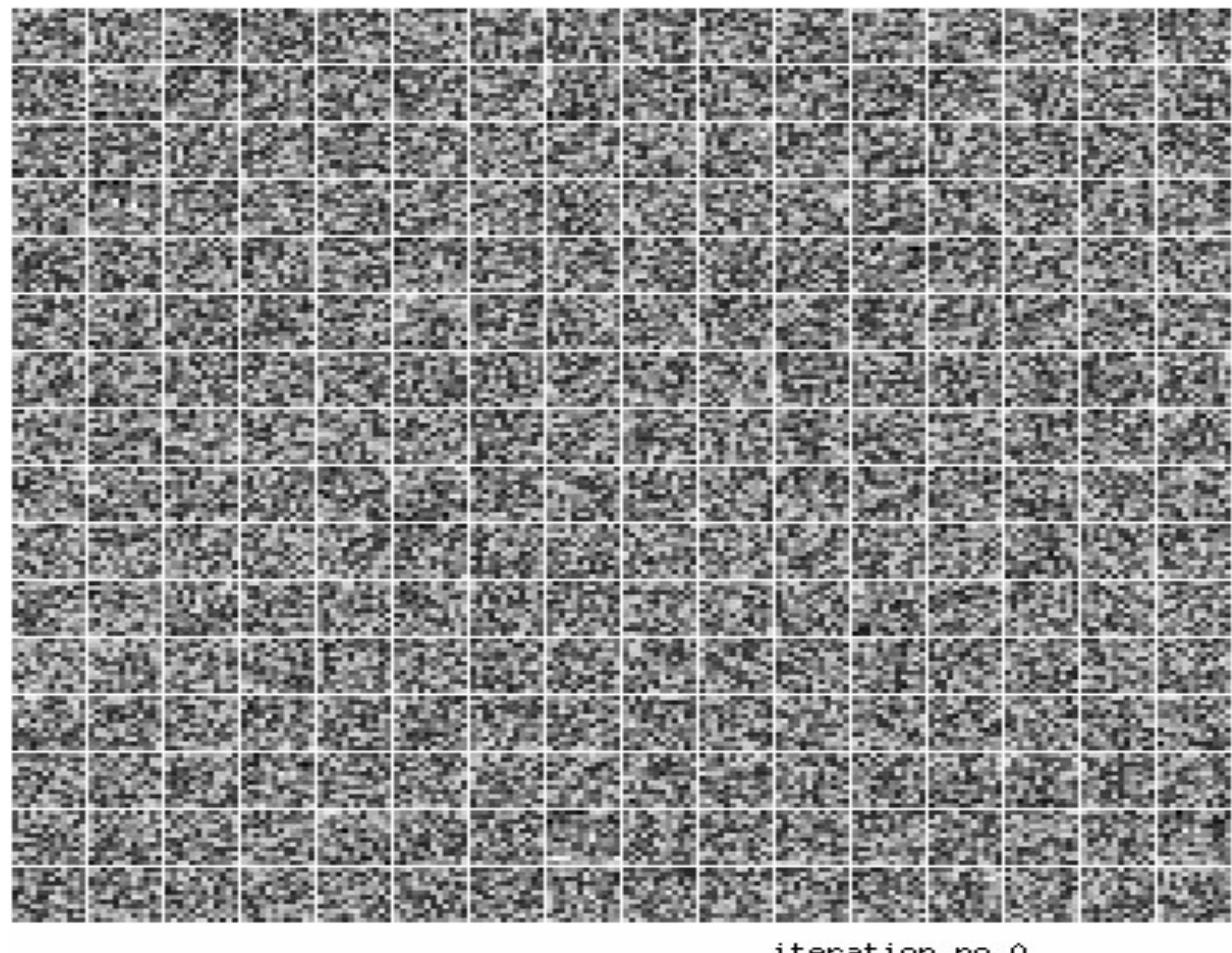
# The “Encoder-Decoder with latent vars” Model

- ▶  $E(Y, Z) = \| Y - \text{Dec}(Z) \|^2 + R(Z) + \| Z - \text{Enc}(Y) \|^2 \quad Z^* = \underset{z}{\operatorname{argmin}} E(Y, Z)$
- ▶ Linear decoder: Predictive Sparse Decomposition
- ▶ [Kavukcuoglu 2008 (arxiv:1010.3467)] [Kavukcuoglu CVPR 2009]

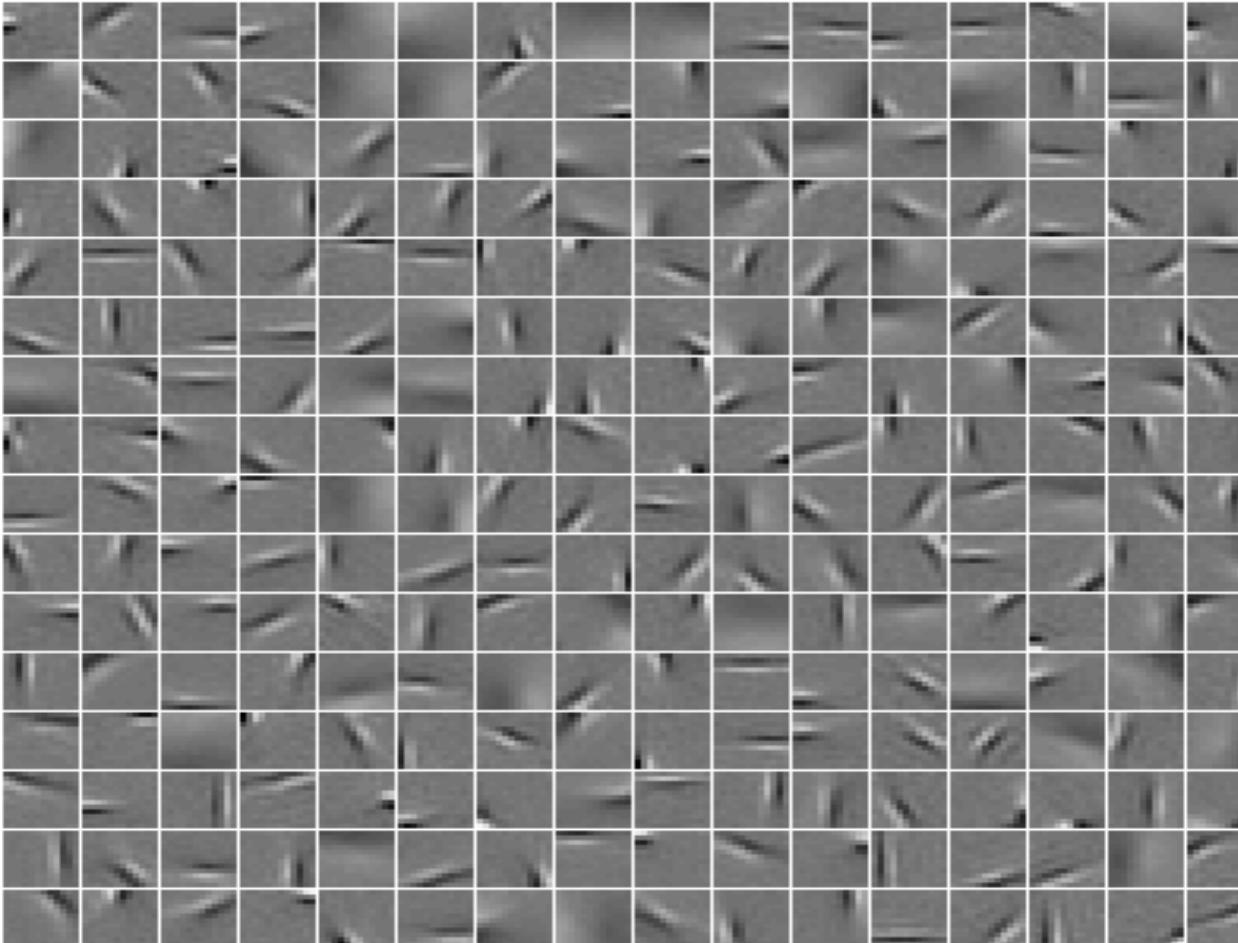


# Predictive Sparse Decomposition (PSD): Training

- ➊ Training on natural images patches.
  - ▶ 12X12
  - ▶ 256 basis functions



# Learned Features on natural patches: V1-like receptive fields



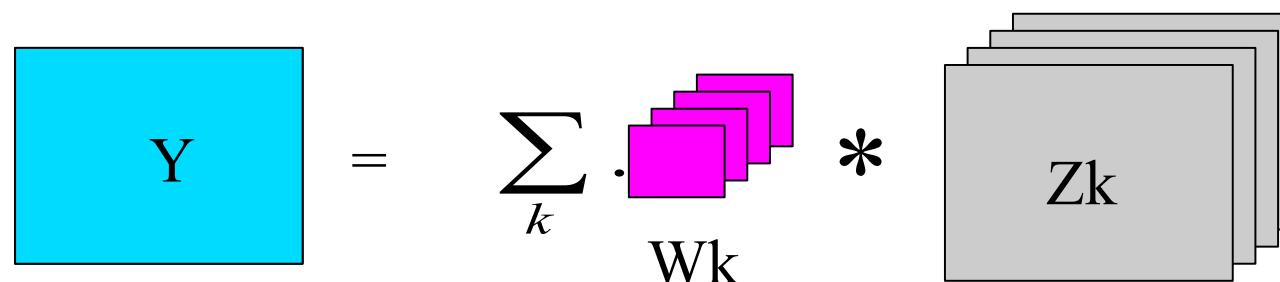
# Convolutional Sparse Coding

- Replace the dot products with dictionary element by convolutions.

- Input  $Y$  is a full image
- Each code component  $Z_k$  is a feature map (an image)
- Each dictionary element is a convolution kernel

- Regular sparse coding**  $E(Y, Z) = \sum_k \|Y - W_k Z_k\|^2 + \alpha \sum_k |Z_k|$

- Convolutional S.C.**  $E(Y, Z) = \sum_k \|Y - W_k * Z_k\|^2 + \alpha \sum_k |Z_k|$



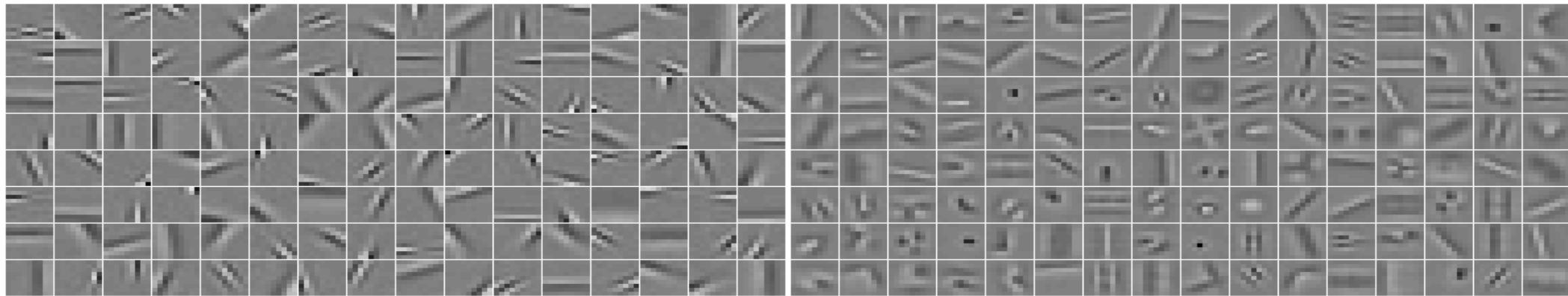
Also used in “deconvolutional networks” [Zeiler, Taylor, Fergus CVPR 2010]

# Convolutional PSD: Encoder with a soft sh() Function

## Convolutional Formulation

- Extend sparse coding from **PATCH** to **IMAGE**

$$\mathcal{L}(x, z, \mathcal{D}) = \frac{1}{2} \|x - \sum_{k=1}^K \mathcal{D}_k * z_k\|_2^2 + \sum_{k=1}^K \|z_k - f(W^k * x)\|_2^2 + |z|_1$$



► **PATCH** based learning

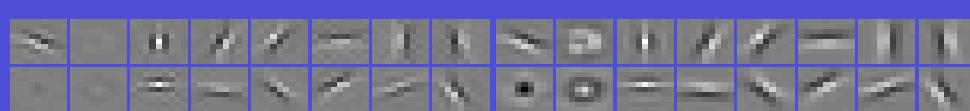
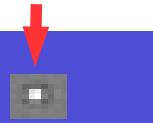
► **CONVOLUTIONAL** learning

# Convolutional Sparse Auto-Encoder on Natural Images

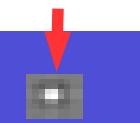
## Filters and Basis Functions obtained

- with 1, 2, 4, 8, 16, 32, and 64 filters.

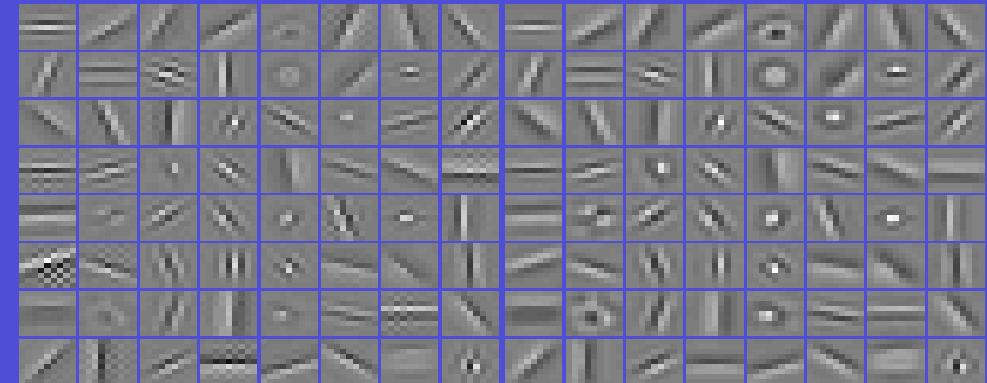
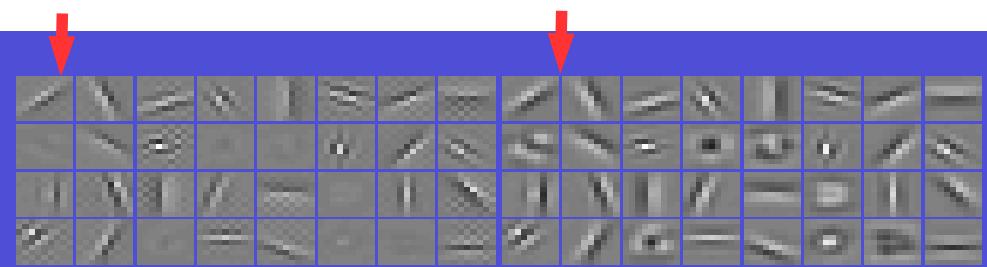
Encoder Filters



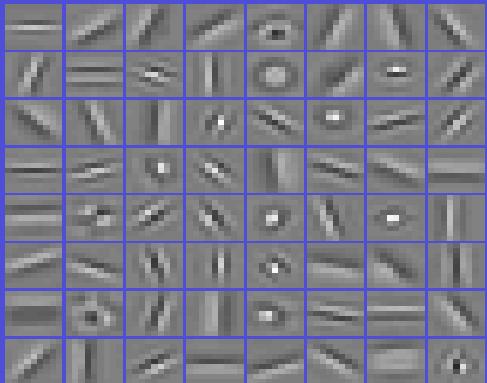
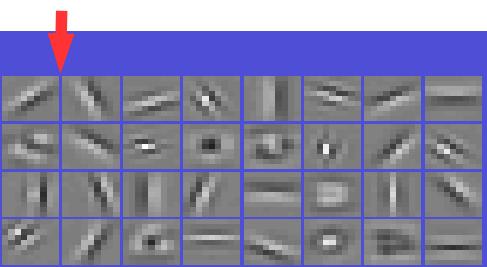
Decoder Filters



Encoder Filters

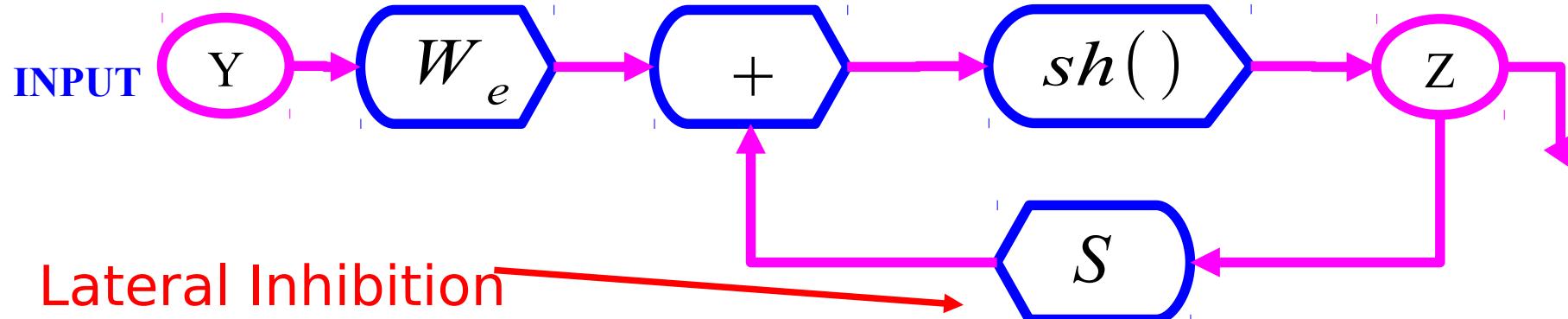


Decoder Filters



# Better Idea: Give the “right” structure to the encoder

- ISTA/FISTA: iterative algorithm that converges to optimal sparse code



$$Z(t+1) = \text{Shrinkage}_{\lambda/L} \left[ Z(t) - \frac{1}{L} W_d^T (W_d Z(t) - Y) \right]$$

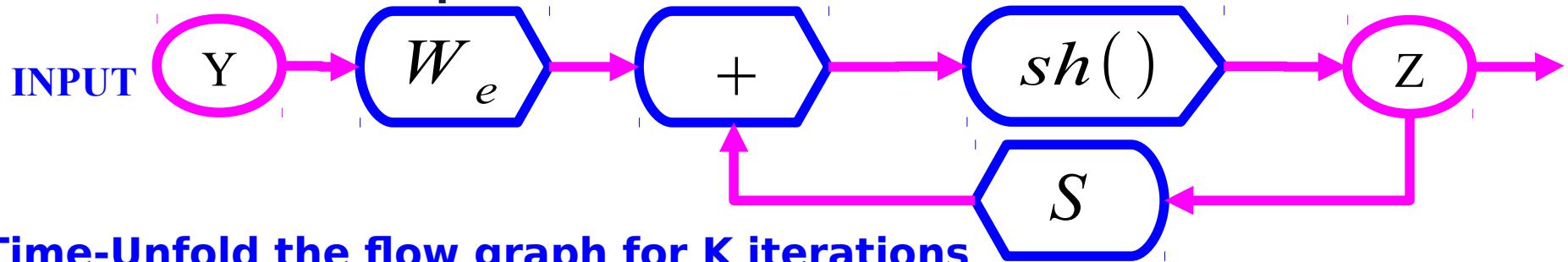
- ISTA/FastISTA reparameterized:

$$Z(t+1) = \text{Shrinkage}_{\lambda/L} [W_e^T Y + S Z(t)] ; \quad W_e = \frac{1}{L} W_d; \quad S = I - \frac{1}{L} W_d^T W_d$$

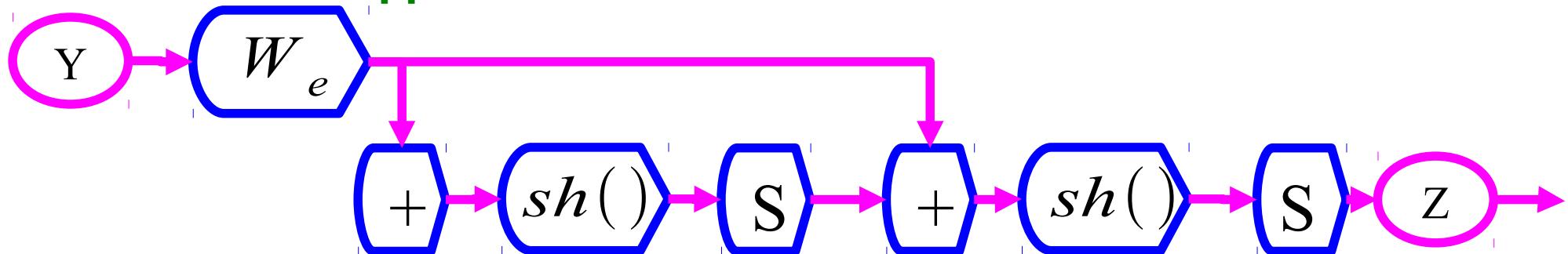
- LISTA (Learned ISTA): learn the  $W_e$  and  $S$  matrices to get fast solutions**

LISTA: Train We and S matrices  
to give a good approximation quickly

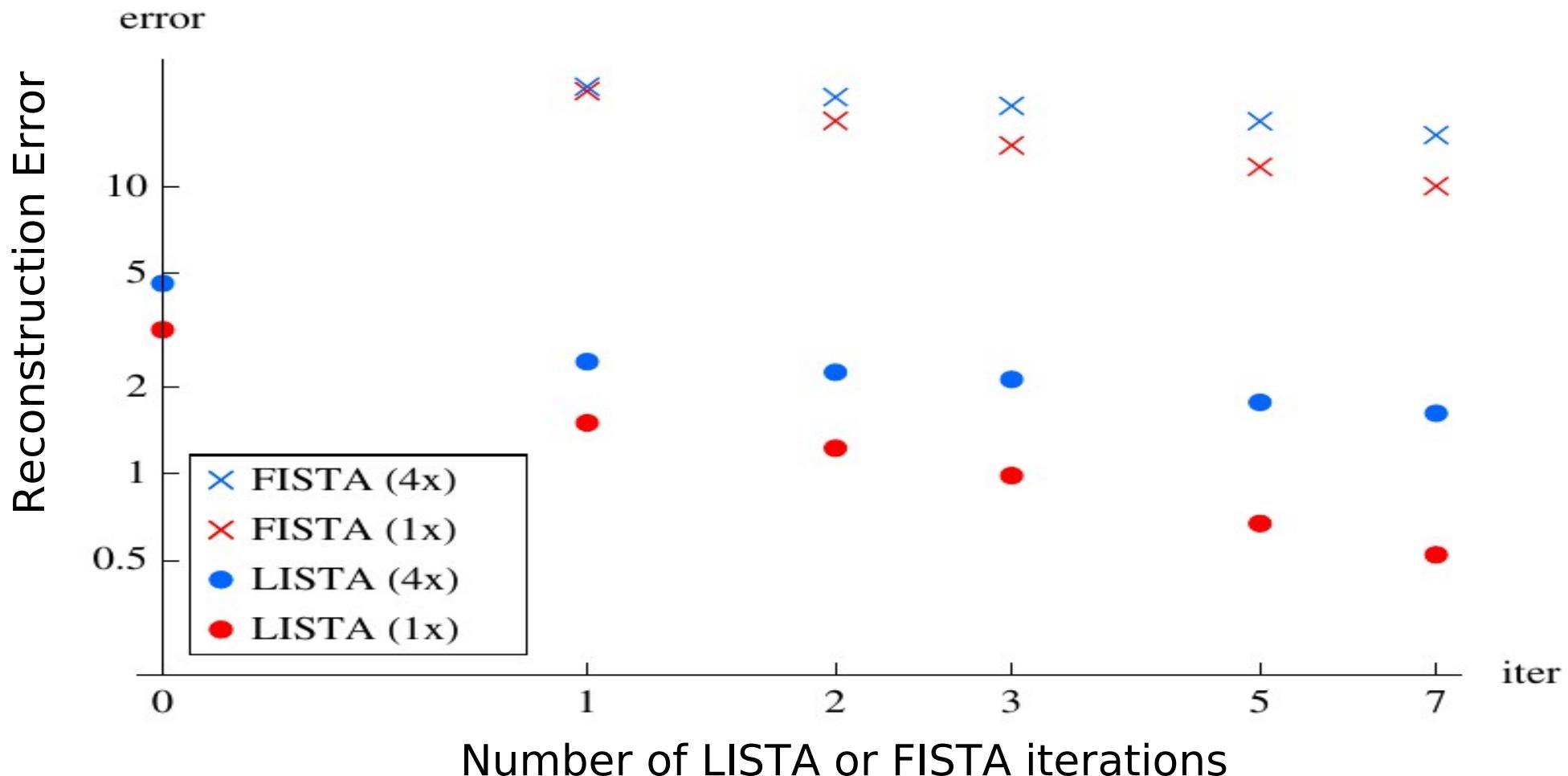
- Think of the FISTA flow graph as a recurrent neural net where We and S are trainable parameters



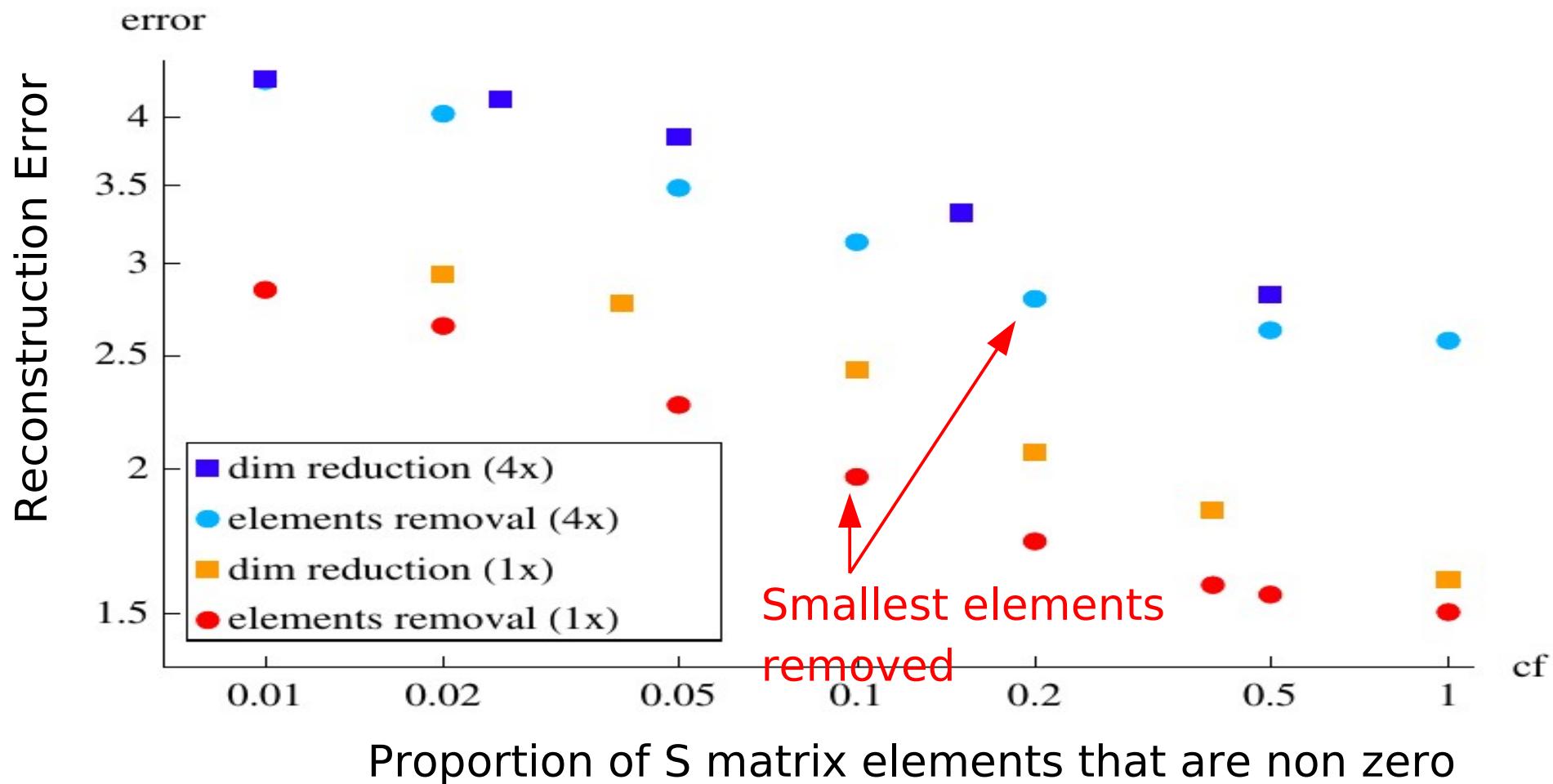
- Time-Unfold the flow graph for K iterations
- Learn the We and S matrices with “backprop-through-time”
- Get the best approximate solution within K iterations



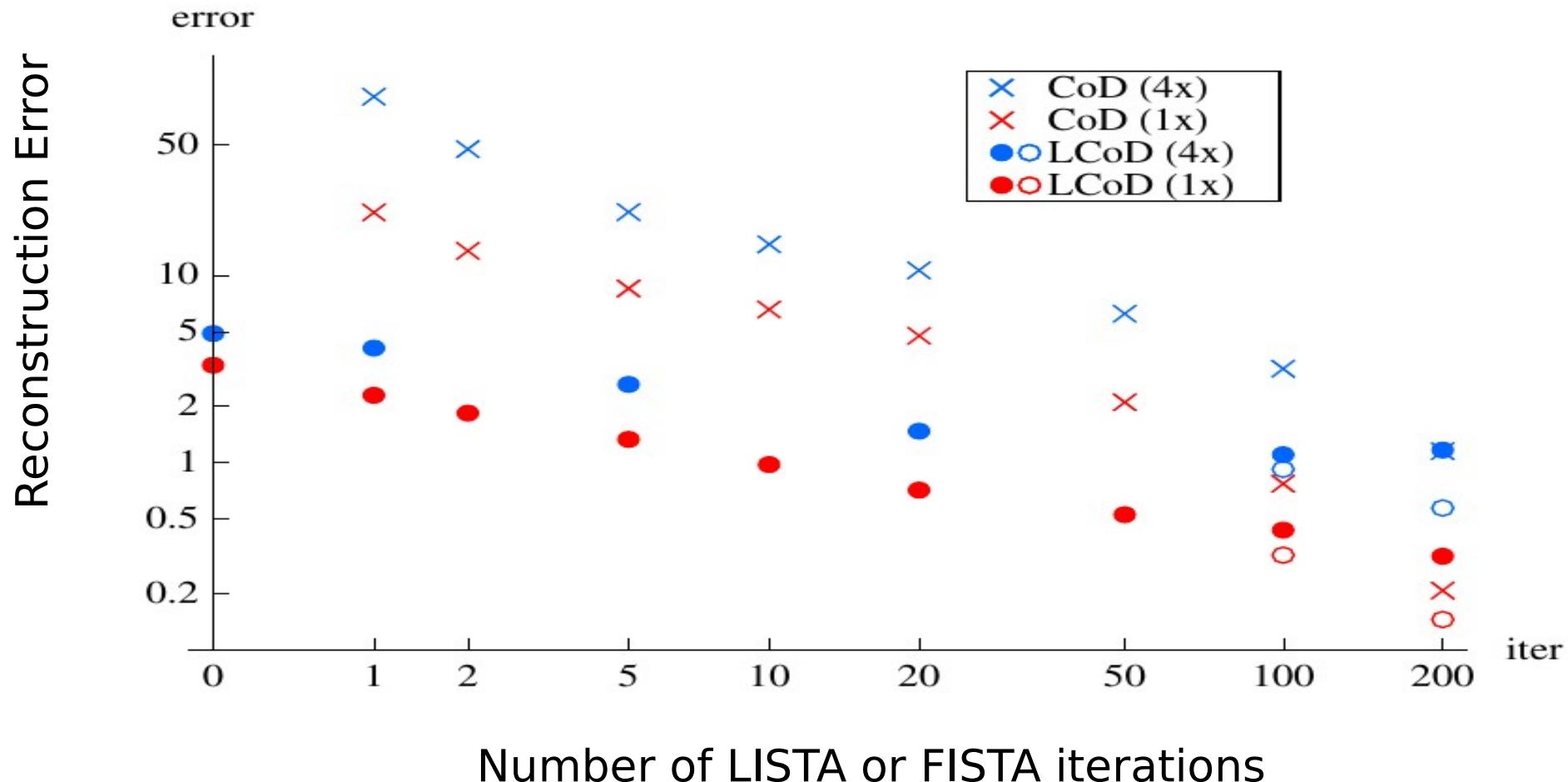
# Learning ISTA (LISTA) vs ISTA/FISTA



# LISTA with partial mutual inhibition matrix

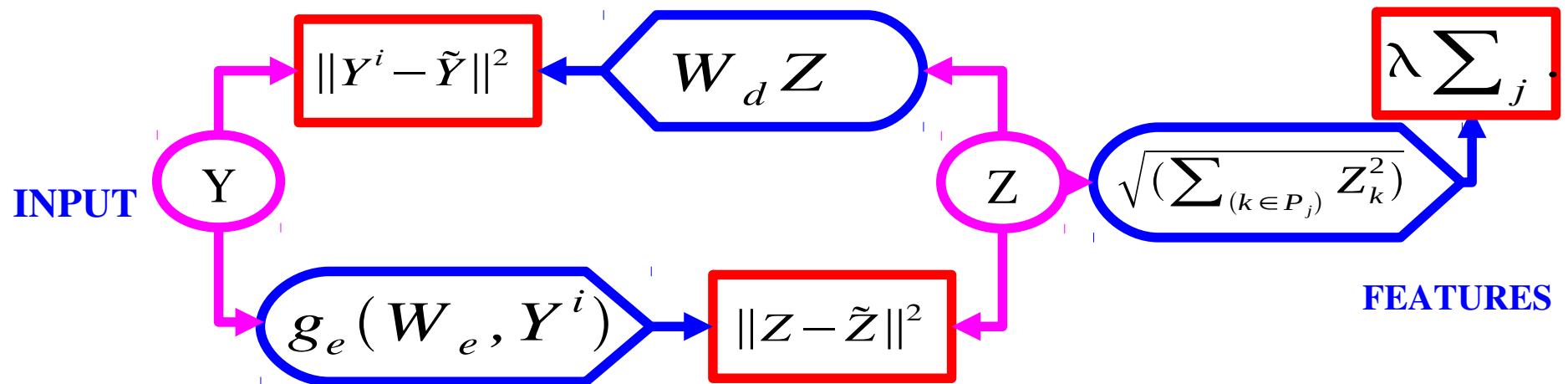


# Learning Coordinate Descent (LcoD): faster than LISTA



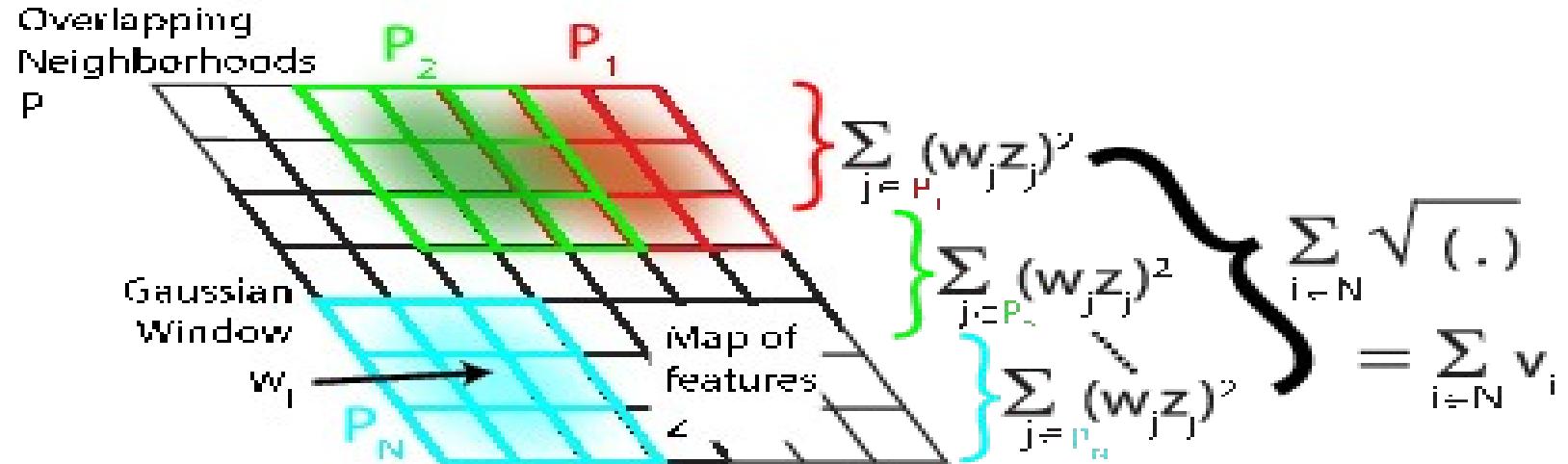
# Learning Invariant Features [Kavukcuoglu et al. CVPR 2009]

- ▶ Unsupervised PSD ignores the spatial pooling step.
- ▶ Could we devise a similar method that learns the pooling layer as well?
- ▶ Idea [Hyvarinen & Hoyer 2001]: **group sparsity** on pools of features
  - ▶ Minimum number of pools must be non-zero
  - ▶ Number of features that are on within a pool doesn't matter
  - ▶ Pools tend to regroup similar features



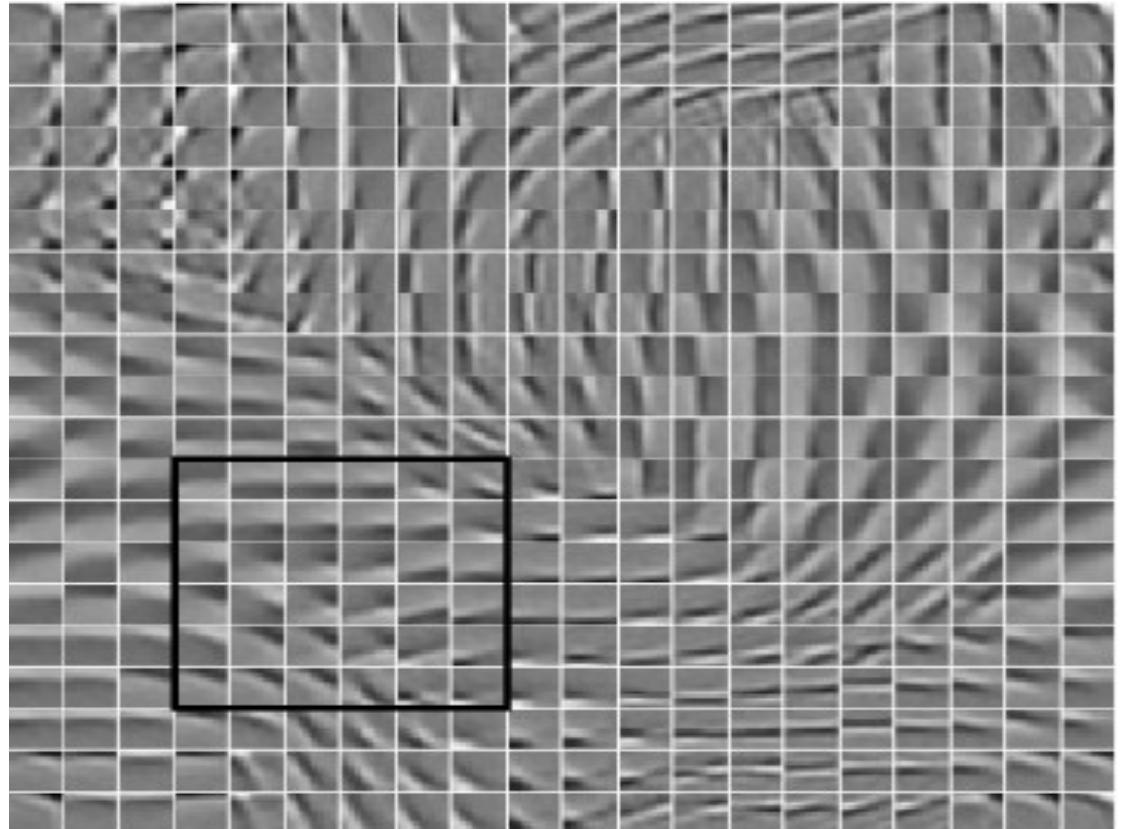
# Why just pool over space? Why not over orientation?

- ▶ Using an idea from Hyvarinen: topographic square pooling (subspace ICA)
- ▶ 1. Apply filters on a patch (with suitable non-linearity)
- ▶ 2. Arrange filter outputs on a 2D plane
- ▶ 3. square filter outputs
- ▶ 4. minimize sqrt of sum of blocks of squared filter outputs



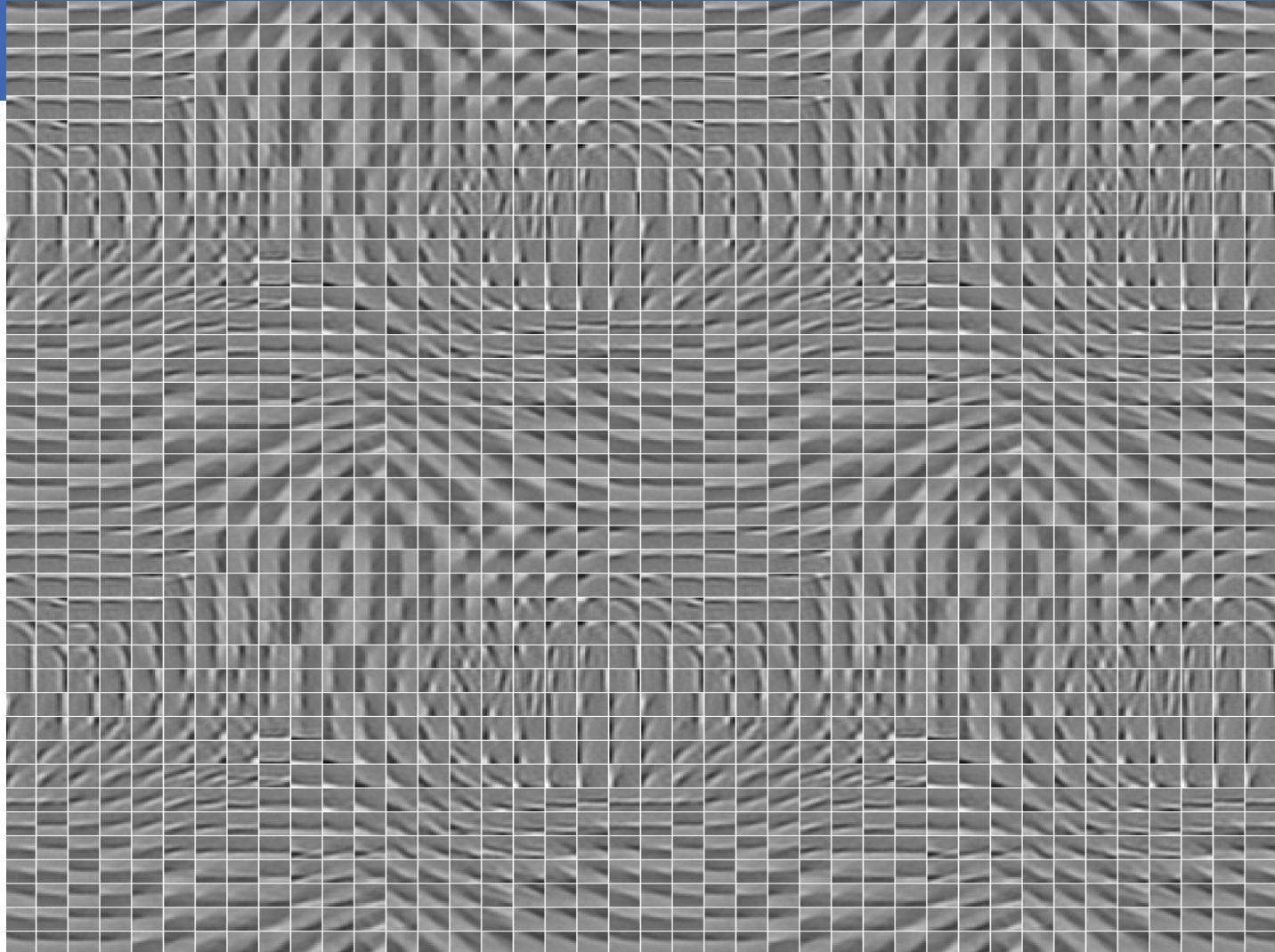
# Why just pool over space? Why not over orientation?

- ▶ The filters arrange themselves spontaneously so that similar filters enter the same pool.
- ▶ The pooling units can be seen as complex cells
- ▶ They are invariant to local transformations of the input
- ▶ For some it's translations, for others rotations, or other transformations.



# Pinwheels!

- ▶ Does that look pinwheely to you?

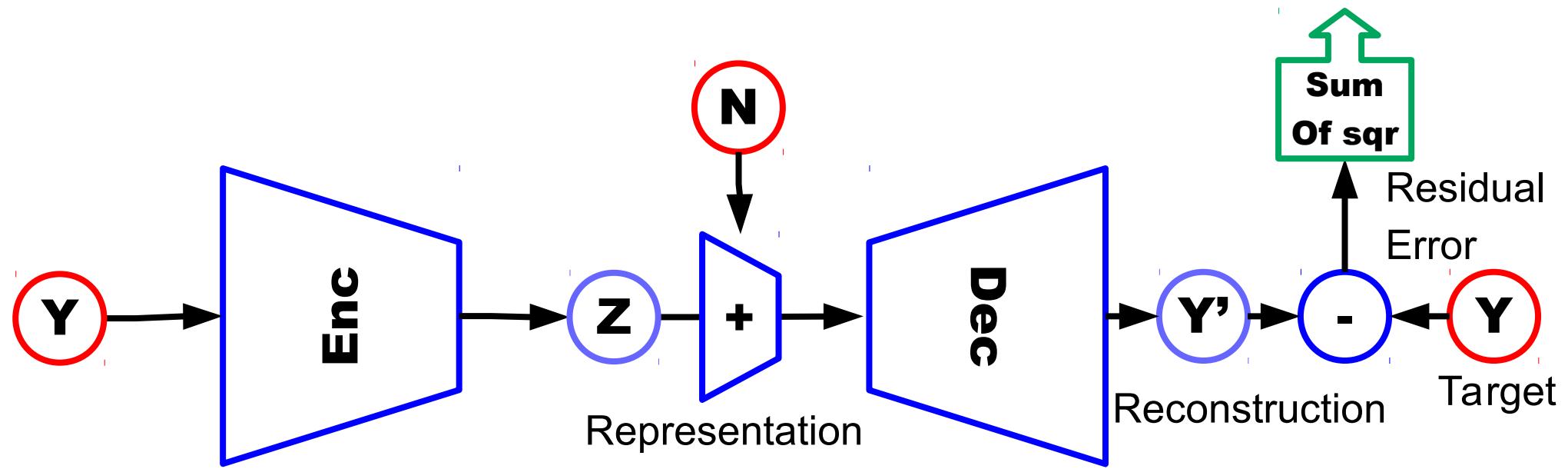


# Variational Auto-Encoder

Limiting the information content  
of the code by adding noise to it

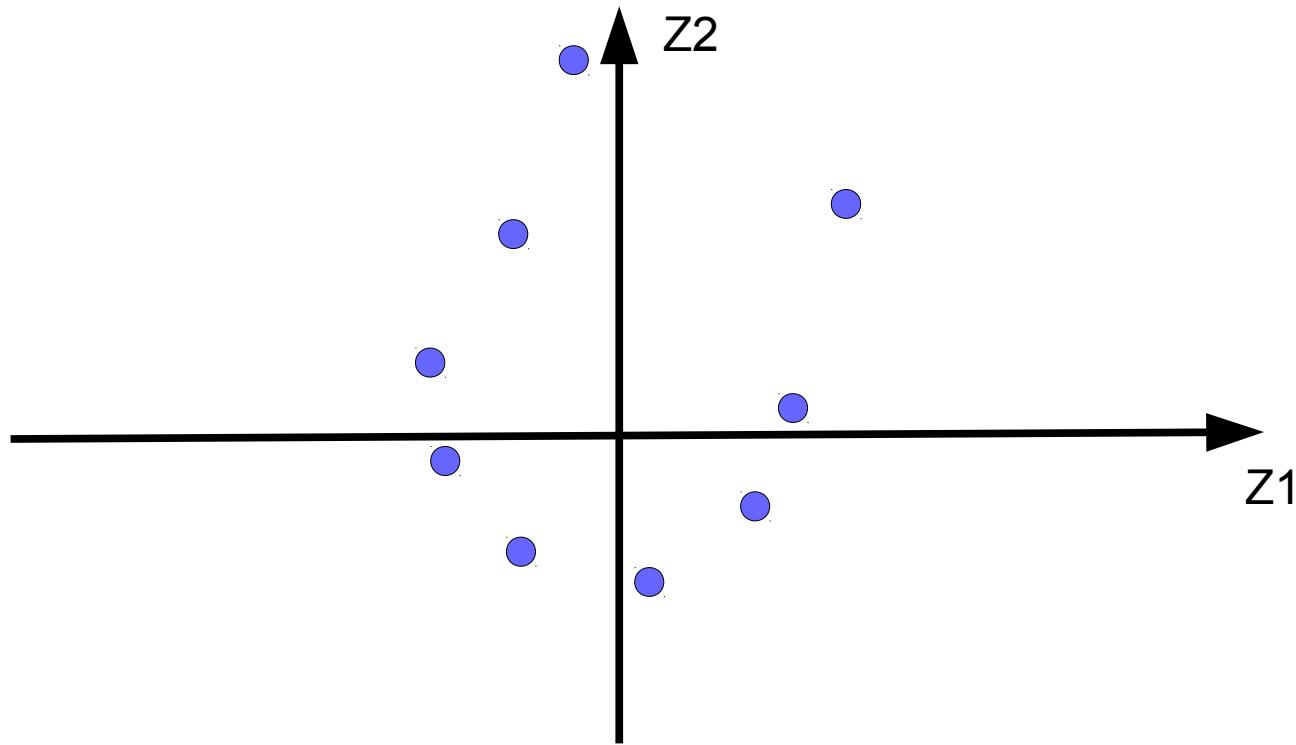
# Variational Auto-Encoder

- ▶ [Kingma & Welling ICLR 2014] arXiv:1312.6114
- ▶ Corrupt the code to limit its information content



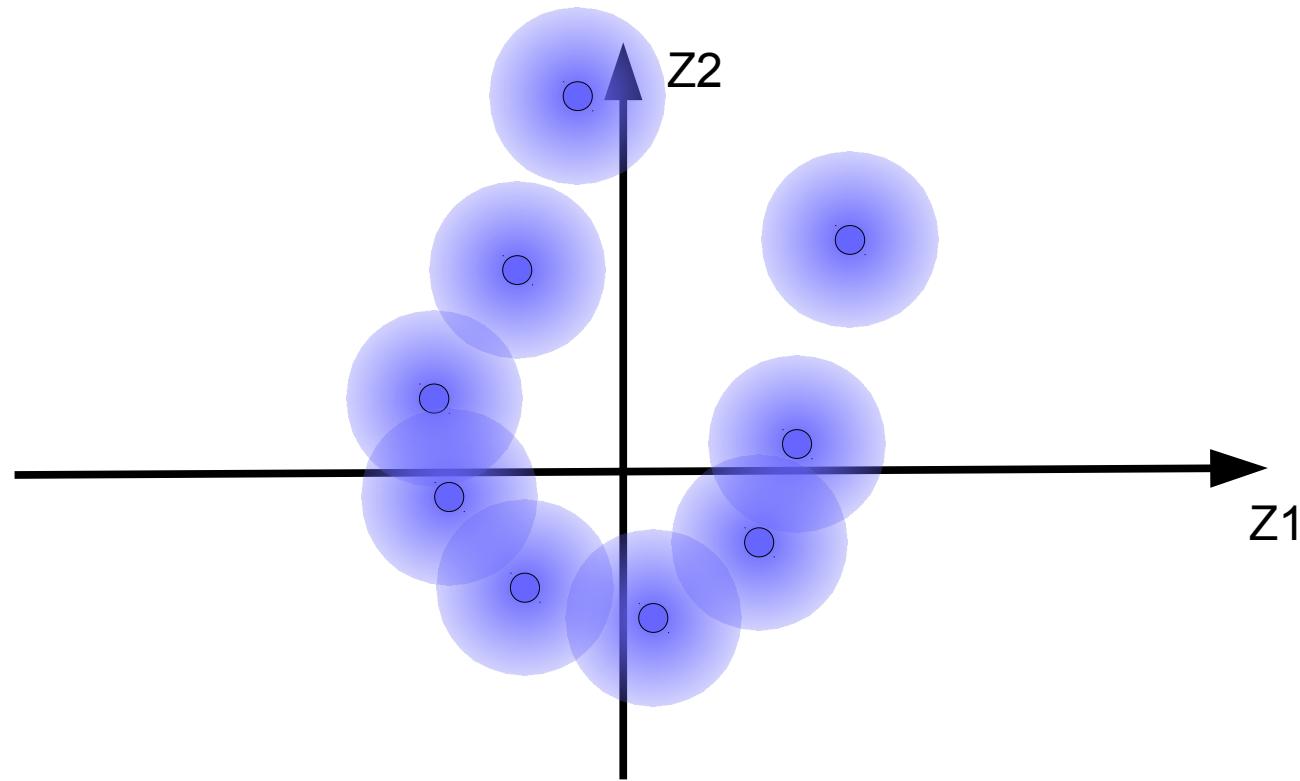
# Variational Auto-Encoder

- ▶ Code vectors for training samples



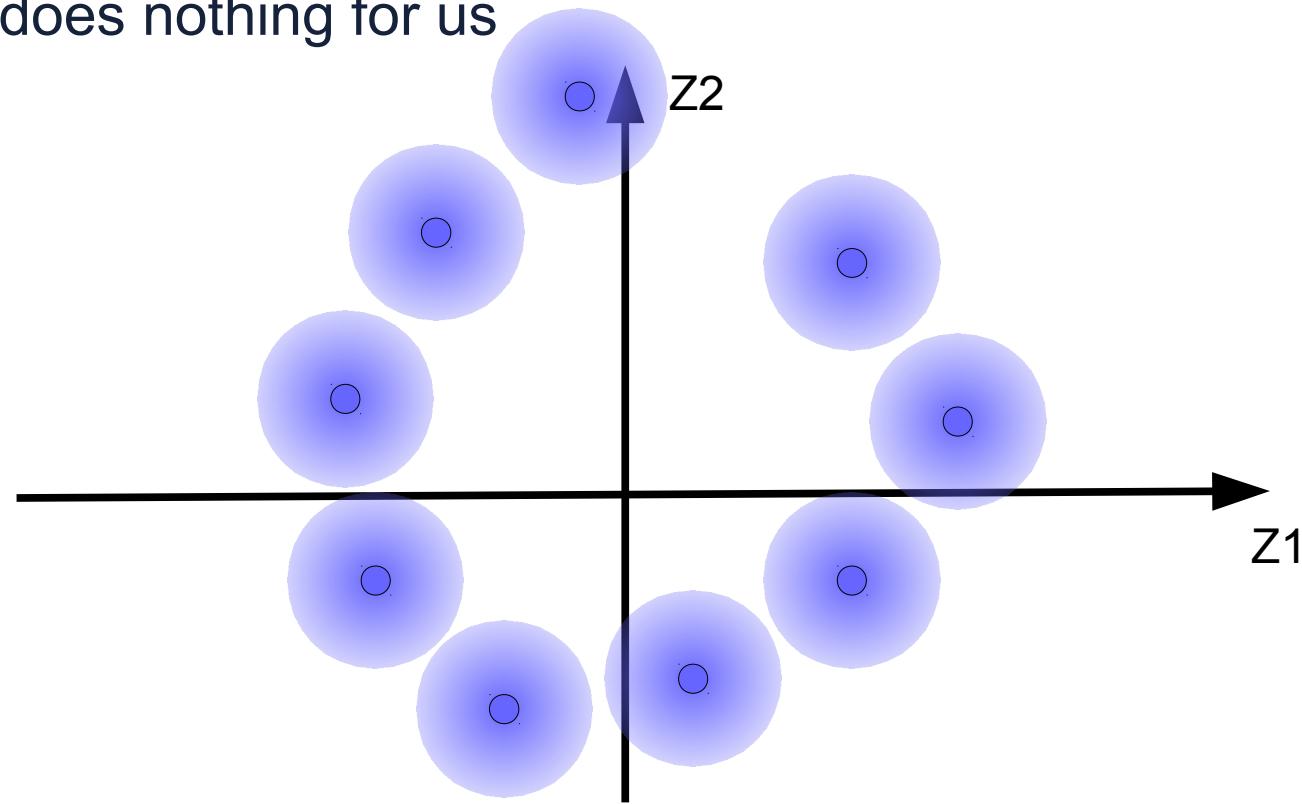
# Variational Auto-Encoder

- ▶ **Code vectors for training sample with Gaussian noise**
- ▶ Some fuzzy balls overlap, causing bad reconstructions



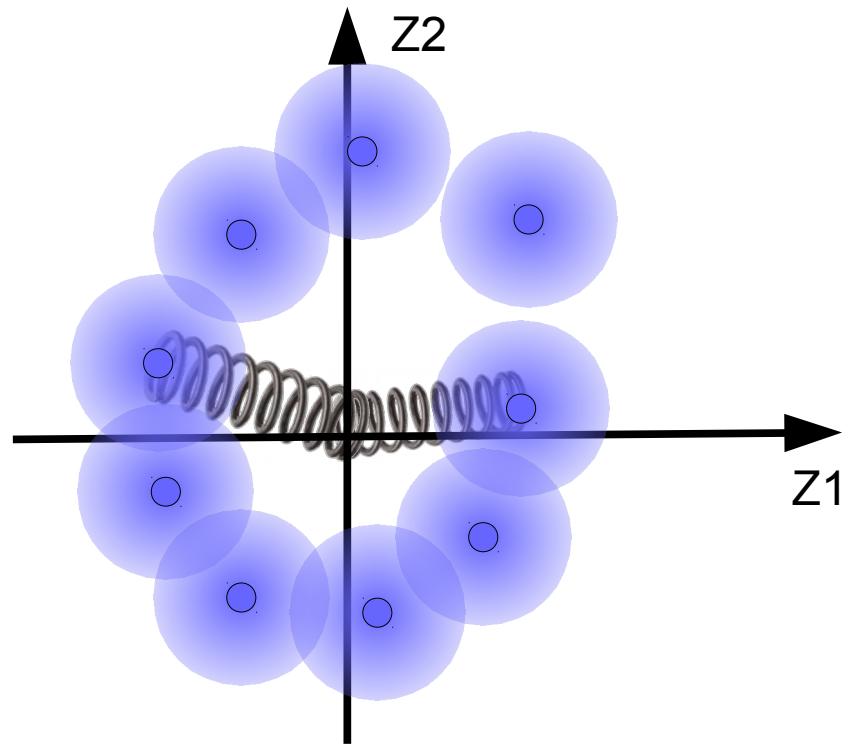
# Variational Auto-Encoder

- ▶ The code vectors want to move away from each other to minimize reconstruction error
- ▶ But that does nothing for us



# Variational Auto-Encoder

- ▶ Attach the balls to the center with a spring, so they don't fly away
- ▶ Minimize the square distances of the balls to the origin
- ▶ Center the balls around the origin
  - ▶ Make the center of mass zero
- ▶ Make the sizes of the balls close to 1 in each dimension
  - ▶ Through a so-called KL term



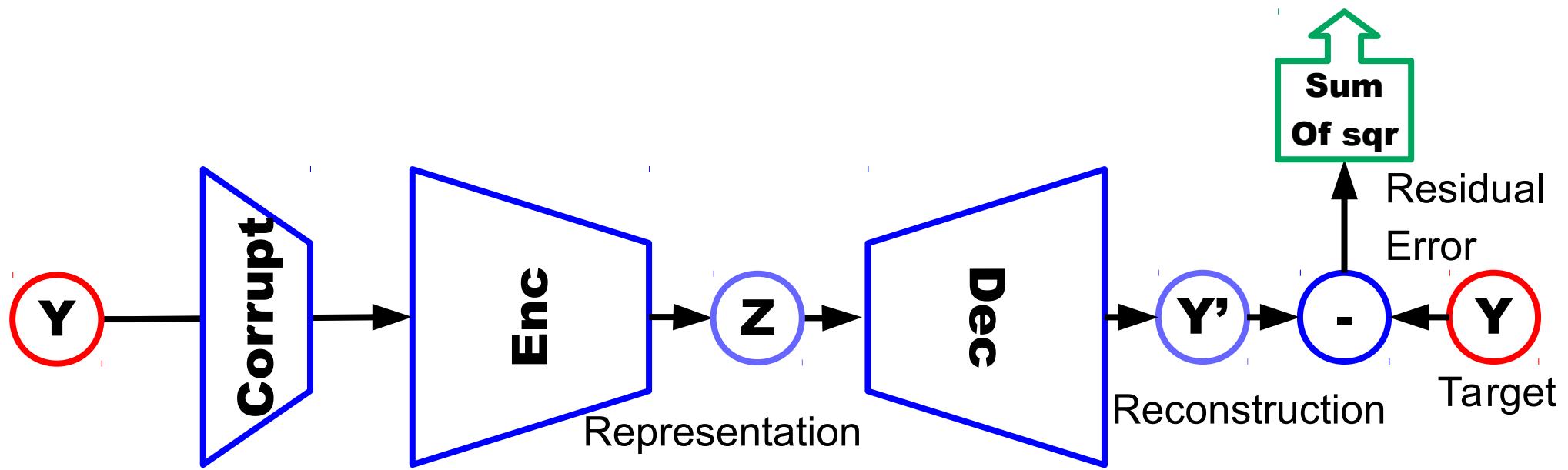


# Denoising Auto-Encoders

Masked AE, BERT...

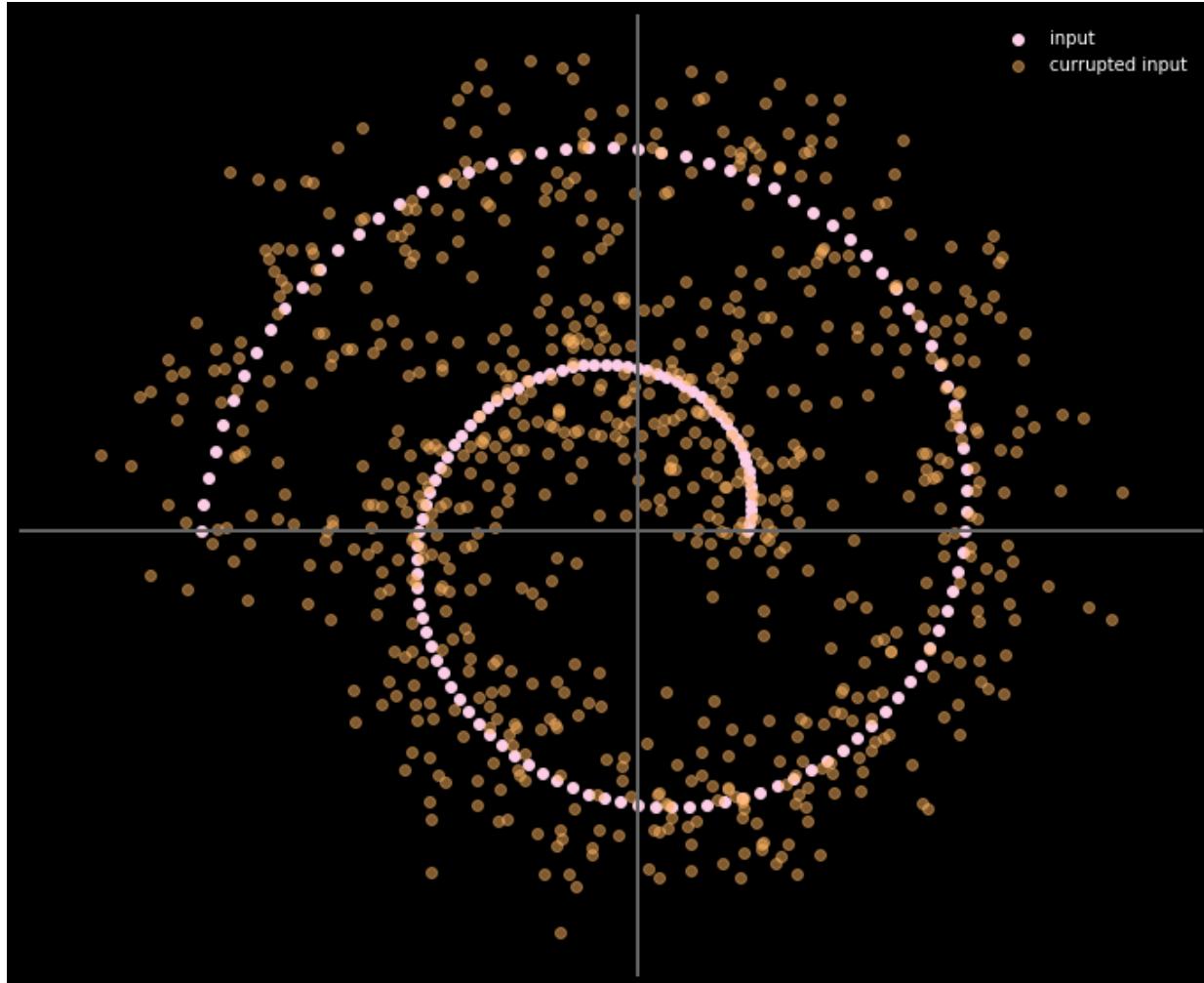
# Denoising Auto-Encoder

- ▶ Corrupt the input, train the network to recover the original input



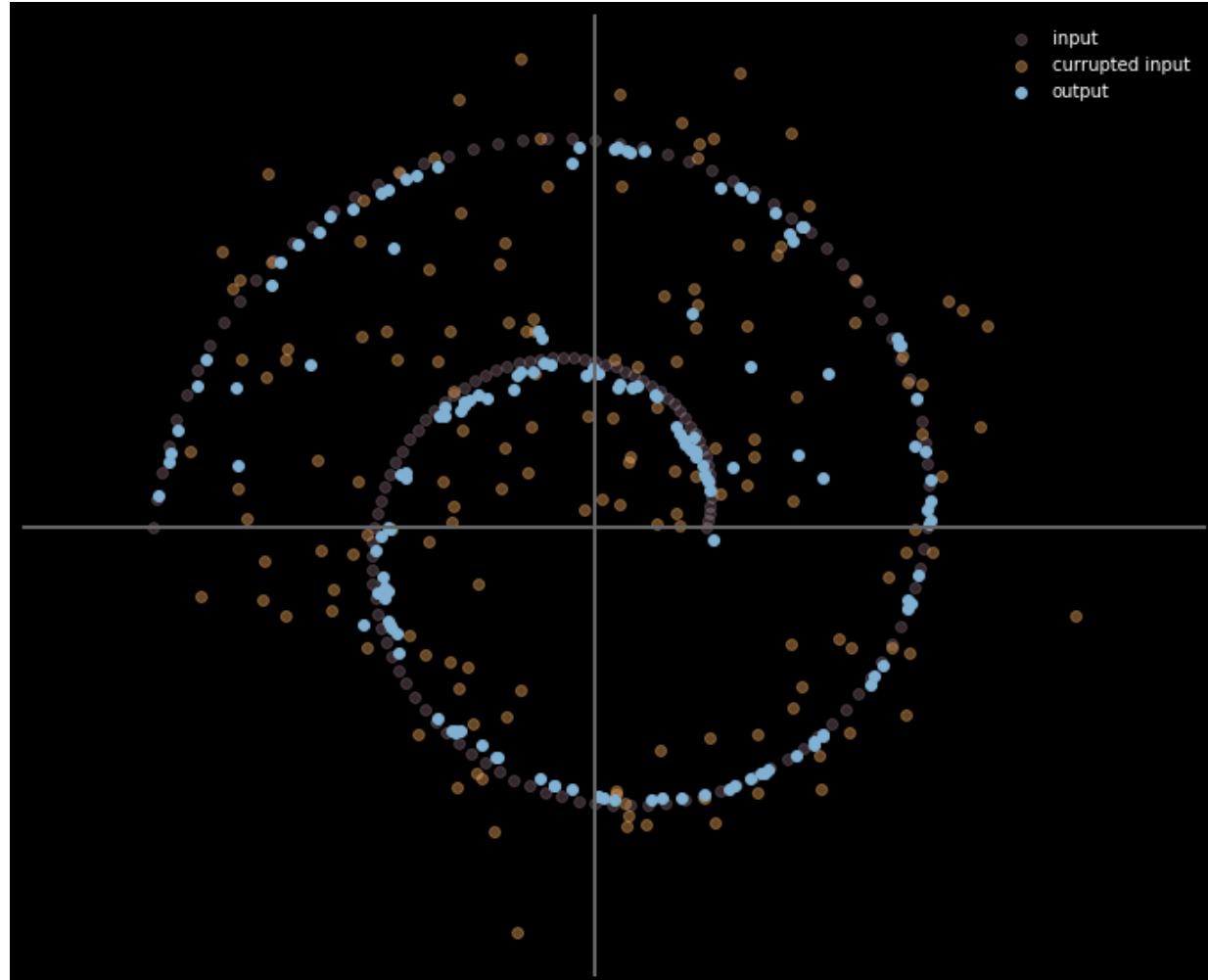
# Denoising Auto-Encoder

- ▶ **Pink points:**
  - ▶ training samples
- ▶ **Orange points:**
  - ▶ corrupted training samples
  - ▶ Additive Gaussian noise
- ▶ **Figures credit:**  
**Alfredo Canziani**



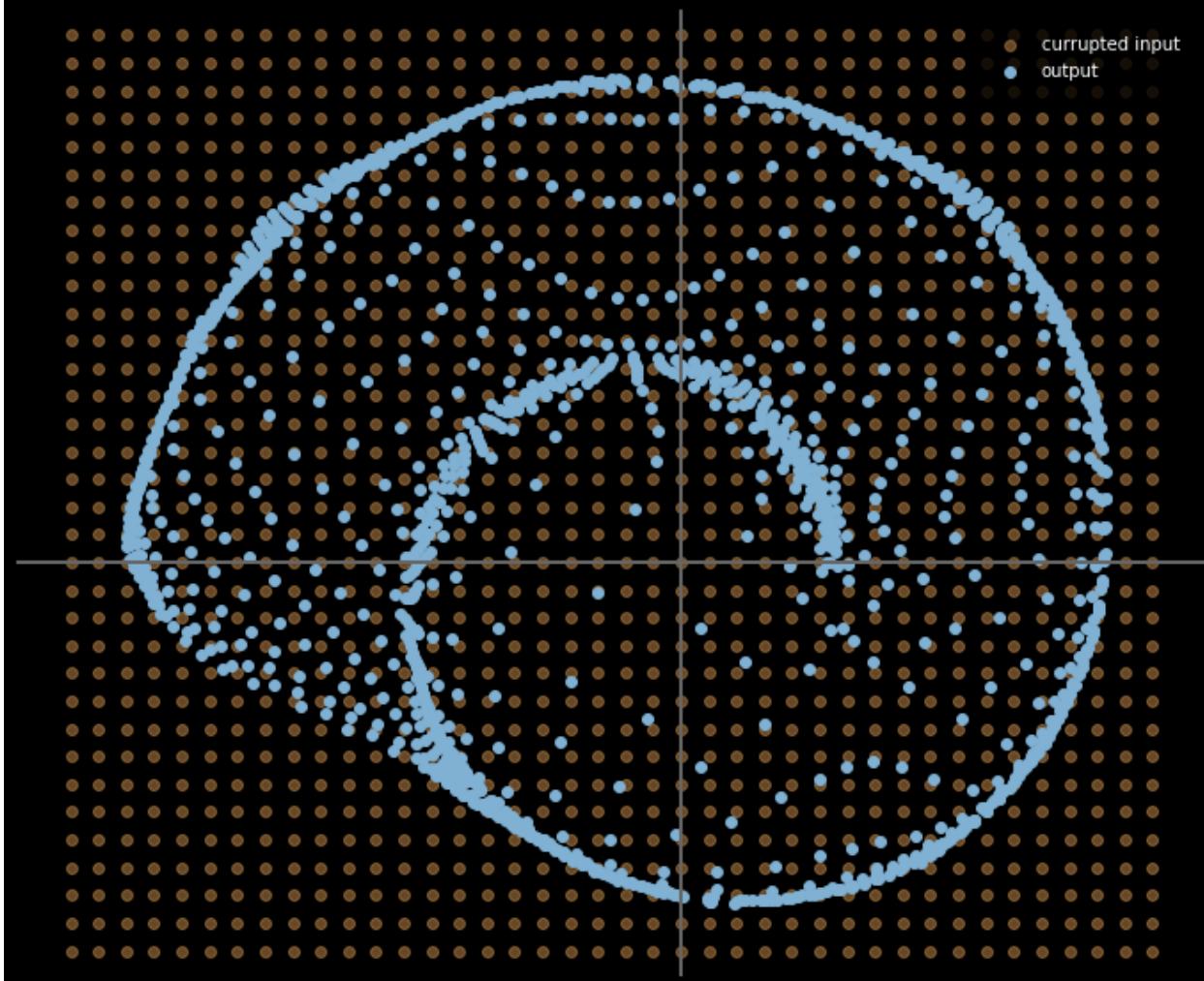
# Data

- ▶ **Pink points:**
  - ▶ training samples
  
- ▶ **Orange points:**
  - ▶ corrupted training samples
  - ▶ Additive Gaussian noise
  
- ▶ **Blue points:**
  - ▶ Network outputs
  - ▶ Denoised samples



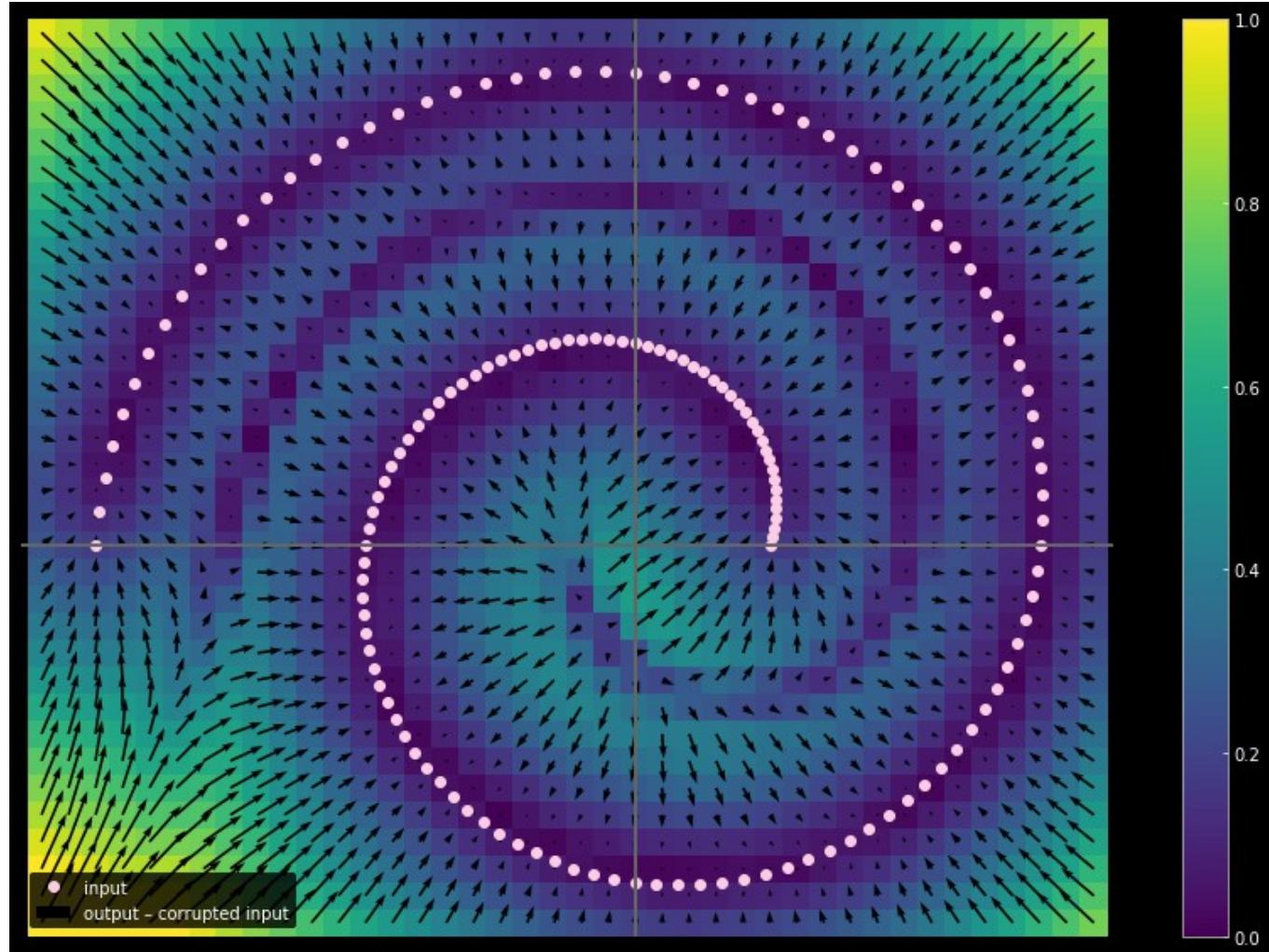
# Data

- ▶ **Orange points:**
  - ▶ Test samples on a grid
  
- ▶ **Blue points:**
  - ▶ Network outputs



# Data

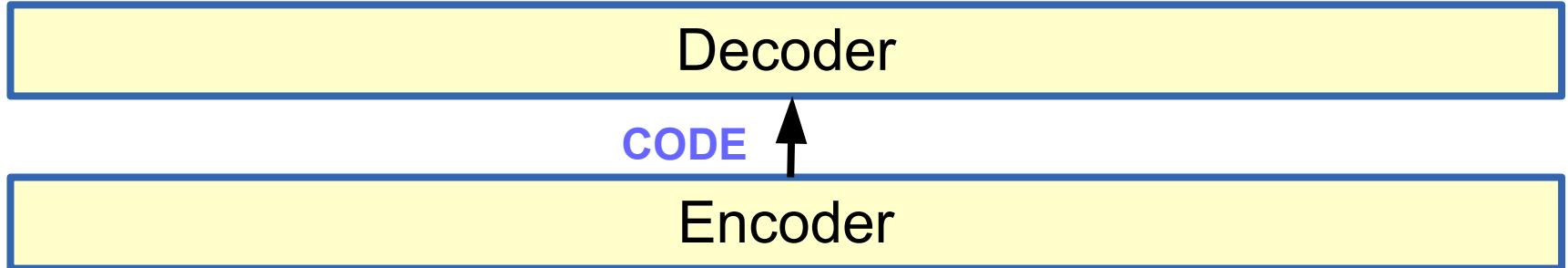
- ▶ **Pink points:**
  - ▶ Training samples
- ▶ **Color:**
  - ▶ Reconstruction energy
- ▶ **Vector field:**
  - ▶ Displacement from network input to network output (scaled down)



# Self-Supervised Learning: filling in the bl\_nks

## ► Natural Language Processing: works great!

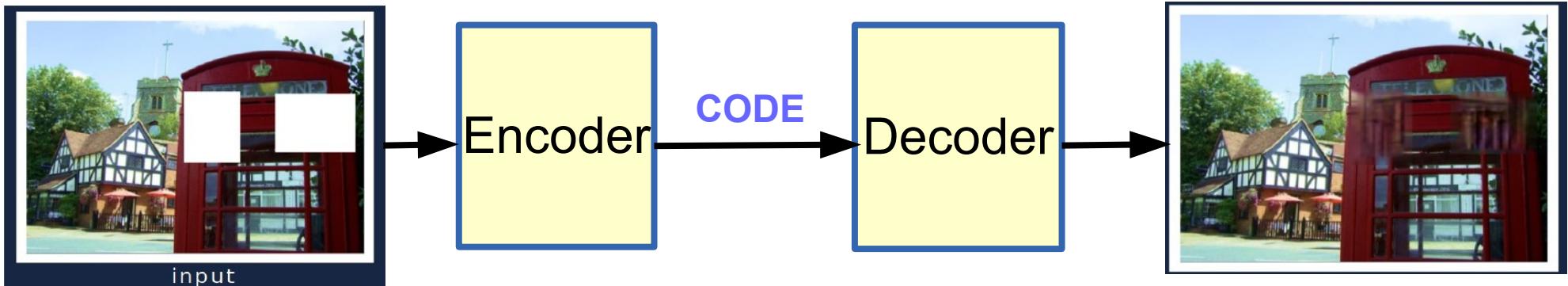
**OUTPUT:** This is a piece of text extracted from a large set of news articles



**INPUT:** This is a [.....] of text extracted [.....] a large set of [.....] articles

## ► Image Recognition / Understanding: works so-so

[Pathak et al 2014]



# Self-Supervised Learning works well for text

- ▶ **Word2vec**
- ▶ [Mikolov 2013]
- ▶ **FastText**
- ▶ [Joulin 2016] (FAIR)
- ▶ **BERT**
  - ▶ Bidirectional Encoder Representations from Transformers
  - ▶ [Devlin 2018]
  - ▶ **Cloze-Driven Auto-Encoder**
  - ▶ [Baevski 2019] (FAIR)

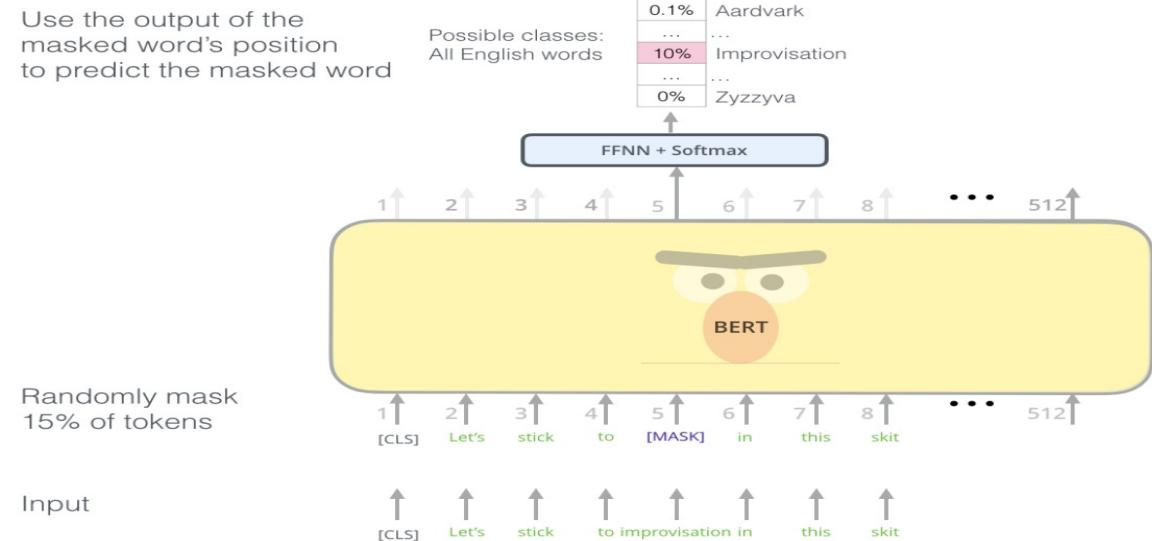
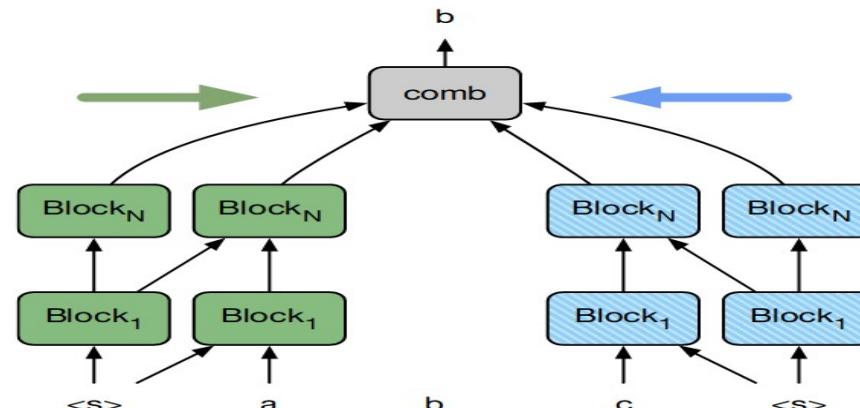


Figure credit: Jay Alammar <http://jalammar.github.io/illustrated-bert/>



# Self-Supervised Learning: Filling in the Blanks



input



Barnes et al. | 2009



Darabi et al. | 2012



Huang et al. | 2014



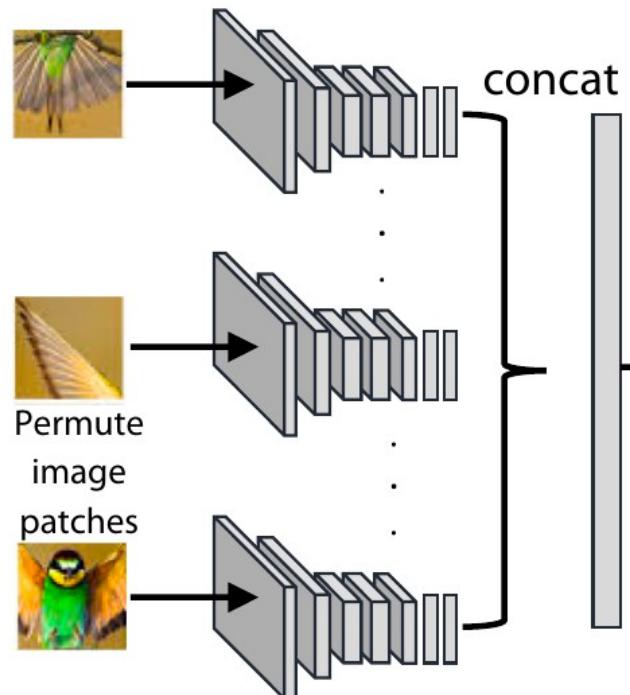
Pathak et al. | 2016



Iizuka et al. | 2017

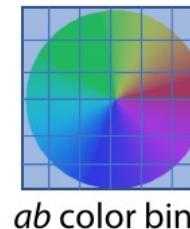
# SSL for Vision [Goyal, Mahavan, Misra, Gupta 2019]

## ► Self-supervised: jigsaw problem and colorization problem



Jigsaw Self-supervision

Quantize color  
space to create  
bins

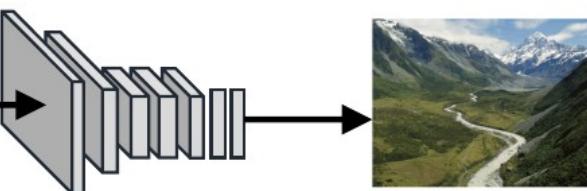


$ab$  color bins



Input grayscale  
image

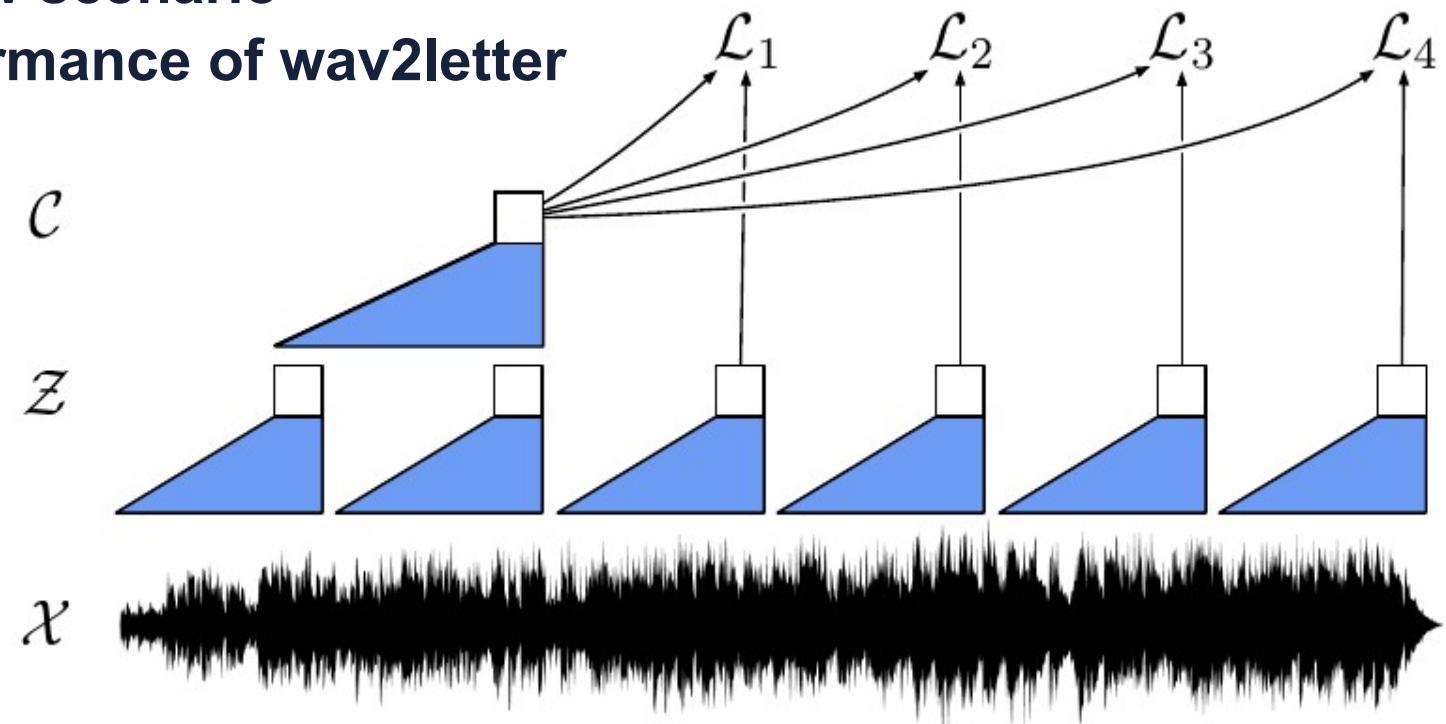
Colorization Self-supervision



Classify  $ab$  color bins

# Wav2vec [Schneider et al. 2019]

- ▶ Self-supervised speech features
- ▶ Classify a future audio segment as compatible or not
- ▶ Similar to jigsaw scenario
- ▶ Improves performance of wav2letter

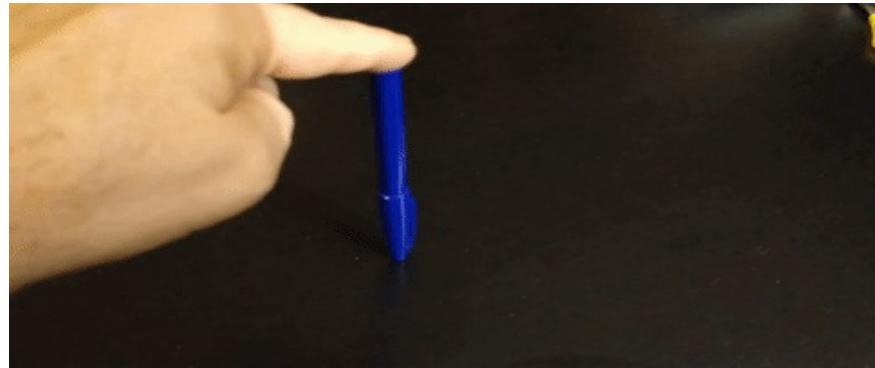


# SSL works with discrete data

- ▶ **BERT / LM: discrete distribution on words**
- ▶ **Colorization: discrete distribution on quantized color bins**
- ▶ **Jigsaw: fundamentally a classification problem**
- ▶ **Wav2vec: fundamentally a 2-class classification problem**
  
- ▶ **SSL works because we know how to represent uncertainty with discrete distributions (see also [Ranzato 2014] on video prediction)**
  
- ▶ **But how do we make it work with high-dimensional continuous data?**
- ▶ Video prediction, audio prediction....

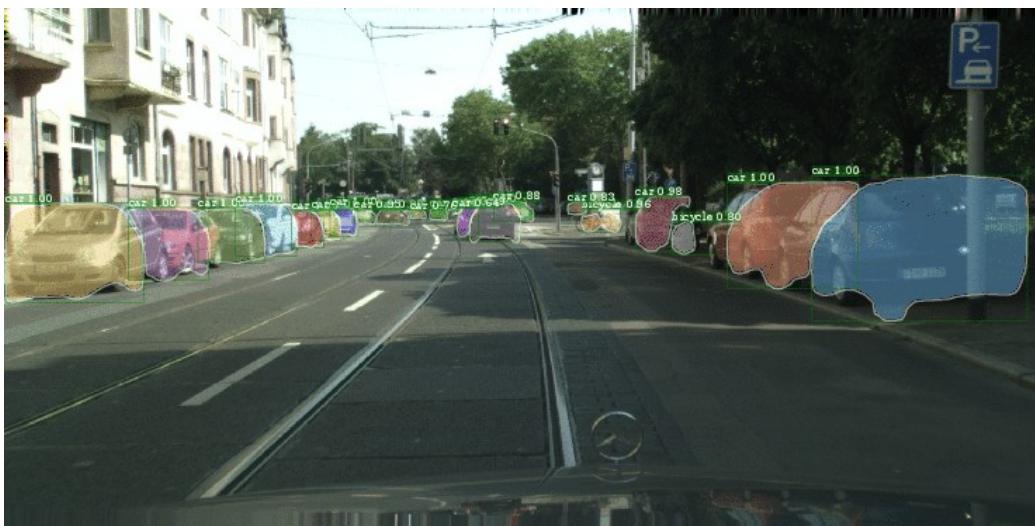
# The world is not entirely predictable / stochastic

- ▶ **Video prediction:**
- ▶ Multiple futures are possible.
- ▶ Training a system to make a single prediction results in “blurry” results
- ▶ the average of all the possible futures



# SSL through Video Prediction

- ▶ Some success
- ▶ [Mathieu, Couprie, YLC ICLR'16 arXiv:1511:05440]
- ▶ [Luc, Couprie, LeCun, Verbeek ECCV 2018]
- ▶ But we are far from a complete solution

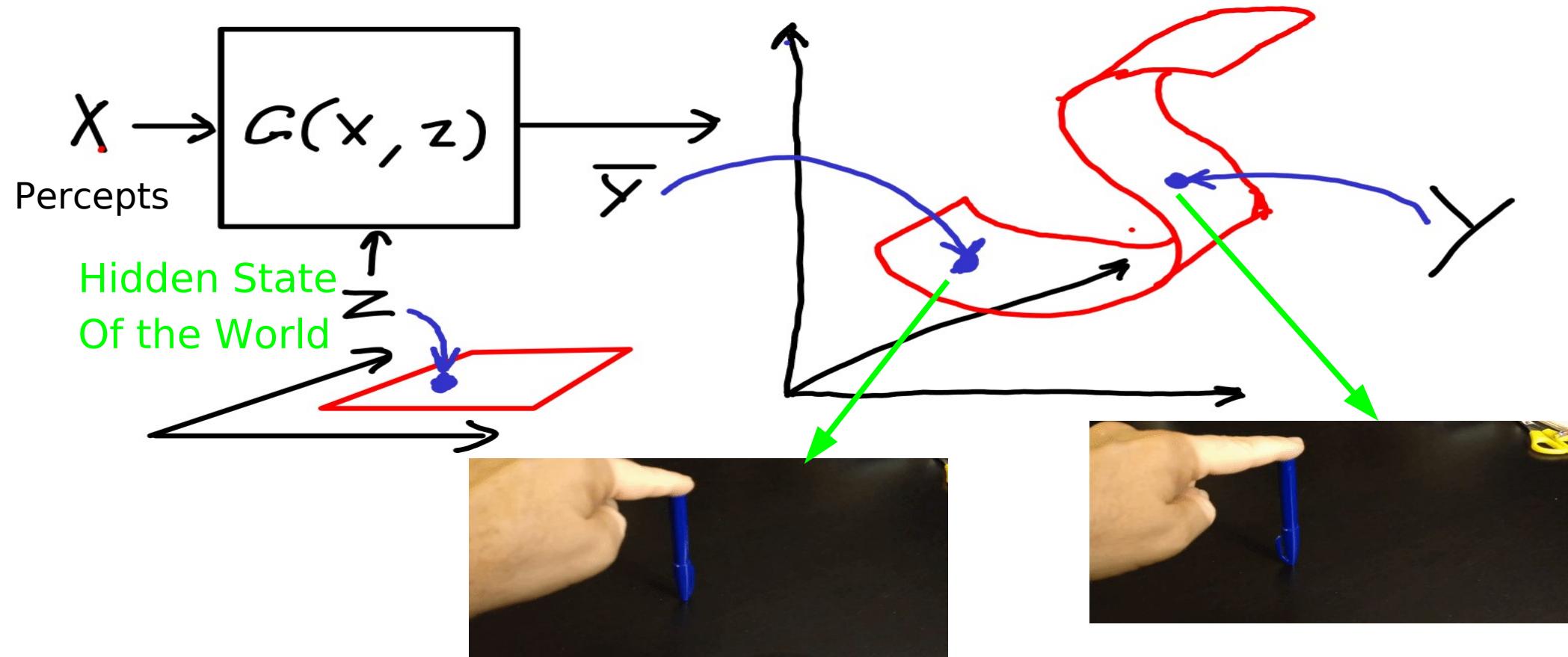




# Adversarial Training & Video Prediction

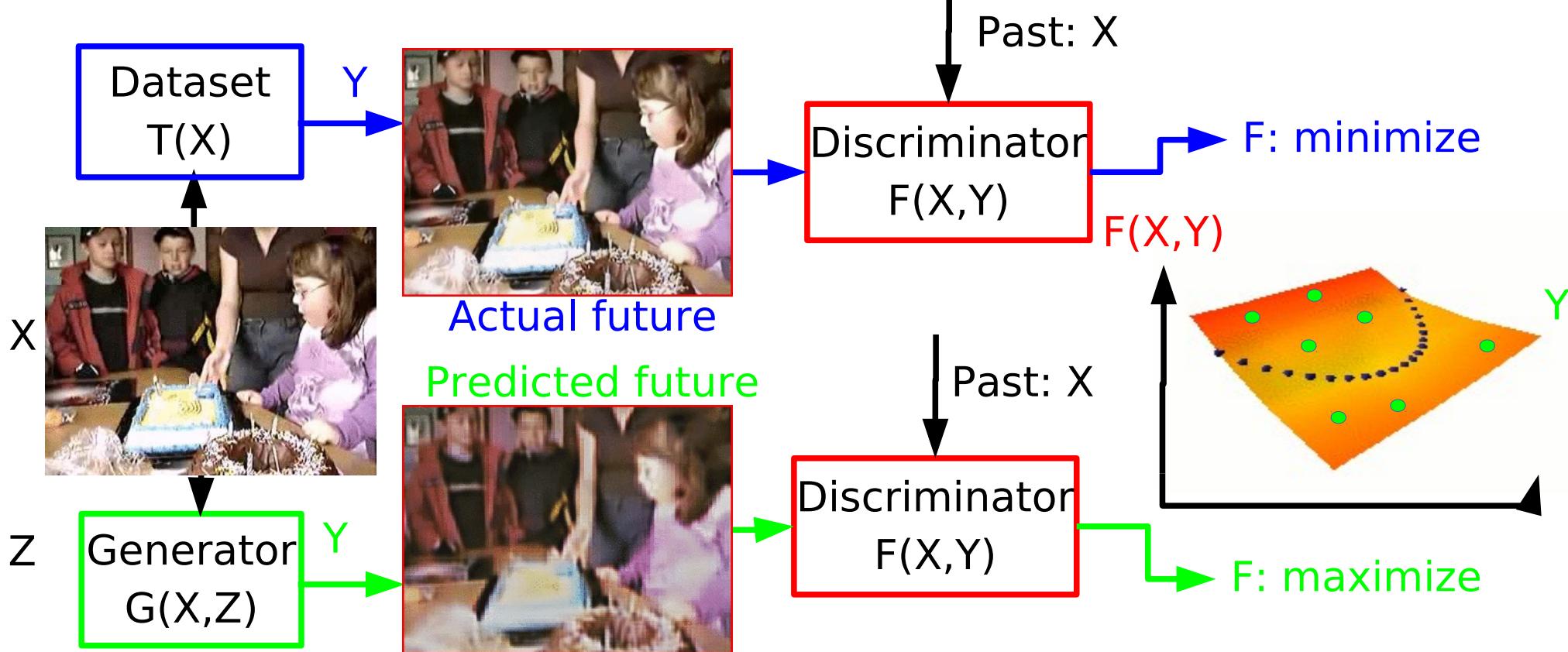
# The Hard Part: Prediction Under Uncertainty

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



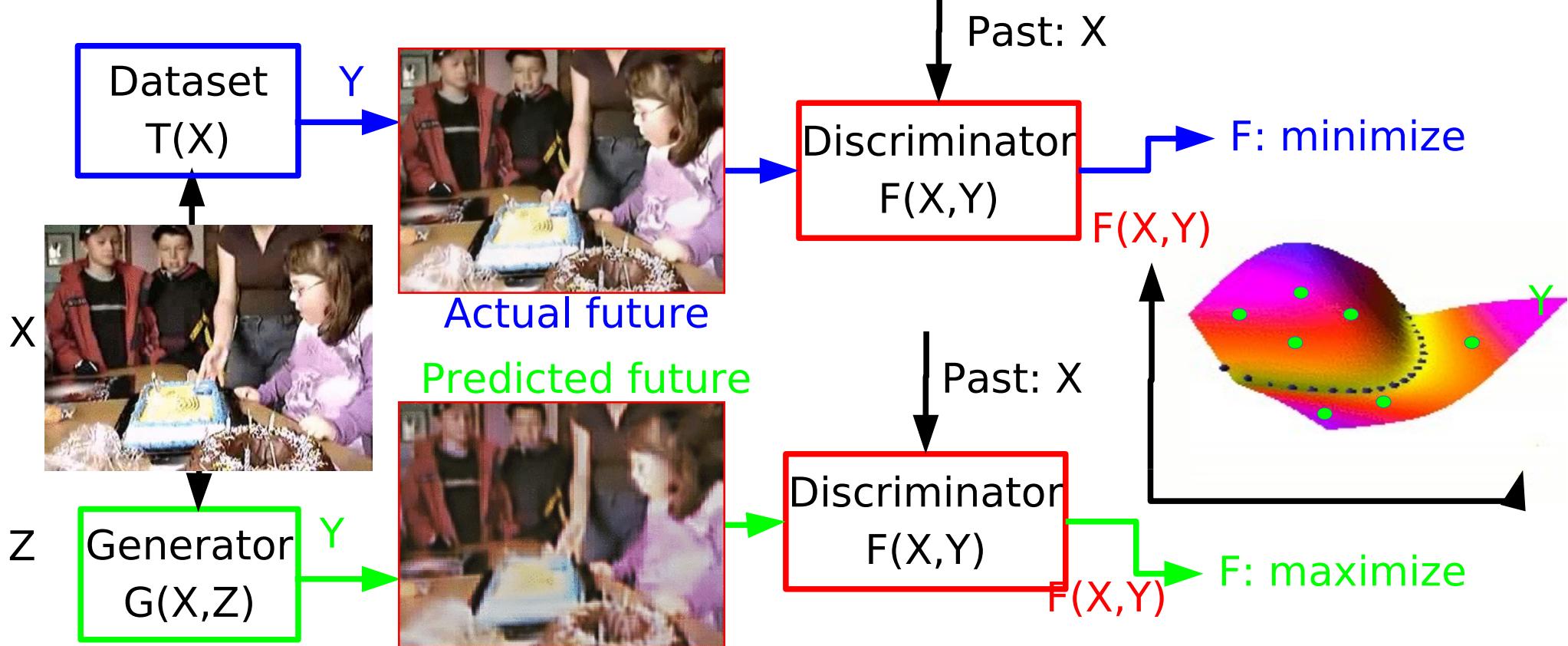
# Adversarial Training: the key to prediction under uncertainty?

- ▶ Generative Adversarial Networks (GAN) [Goodfellow et al. NIPS 2014],
- ▶ Energy-Based GAN [Zhao, Mathieu, LeCun ICLR 2017 & arXiv:1609.03126]



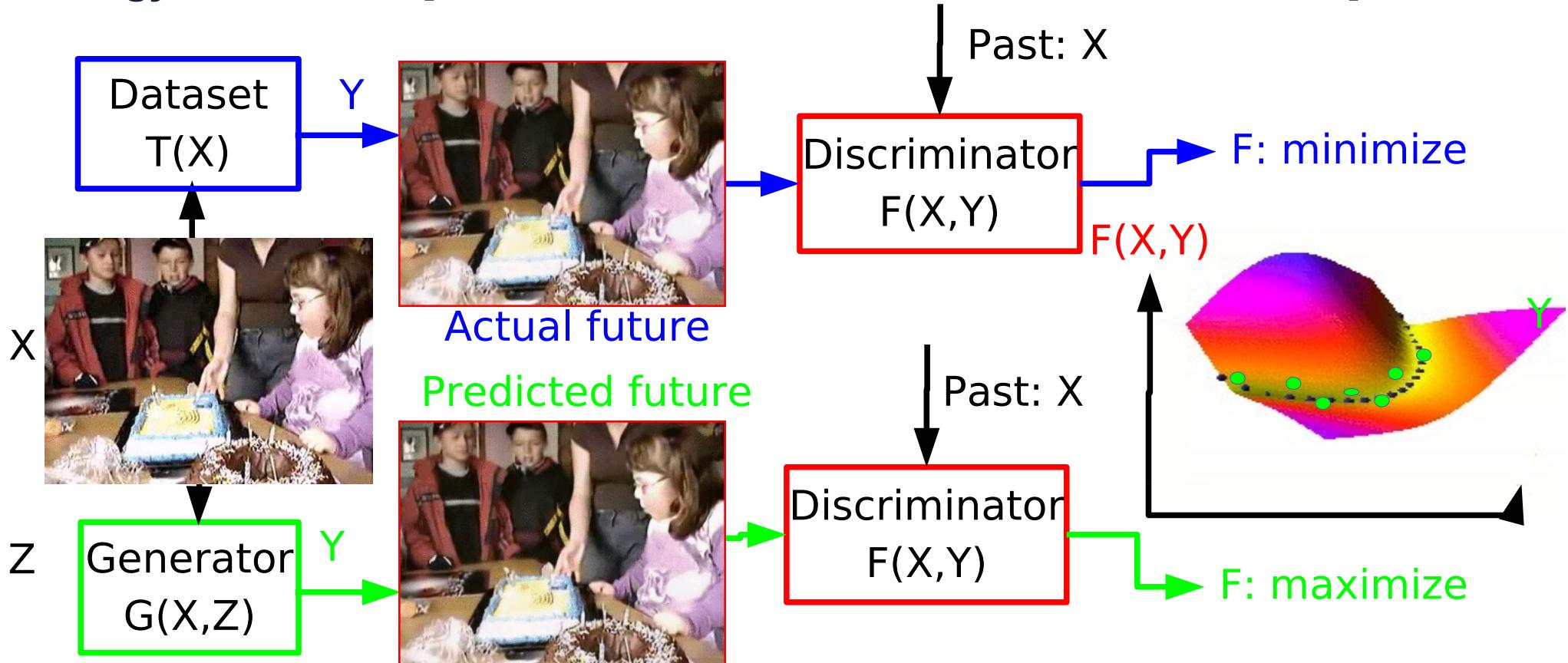
# Adversarial Training: the key to prediction under uncertainty?

- ▶ Generative Adversarial Networks (GAN) [Goodfellow et al. NIPS 2014],
- ▶ Energy-Based GAN [Zhao, Mathieu, LeCun ICLR 2017 & arXiv:1609.03126]



# Adversarial Training: the key to prediction under uncertainty?

- ▶ Generative Adversarial Networks (GAN) [Goodfellow et al. NIPS 2014],
- ▶ Energy-Based GAN [Zhao, Mathieu, LeCun ICLR 2017 & arXiv:1609.03126]



# Faces “invented” by a GAN (Generative Adversarial Network)

- ▶ Random vector → Generator Network → output image [Goodfellow NIPS 2014]  
[Karras et al. ICLR 2018] (from NVIDIA)



# Generative Adversarial Networks for Creation

► [Sbai 2017]



# Self-supervised Adversarial Learning for Video Prediction

- ▶ 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.





# Learning Forward Models Through Video Prediction

[Lerer, Gross, Fergus ICML 2016, arxiv:1603.01312]

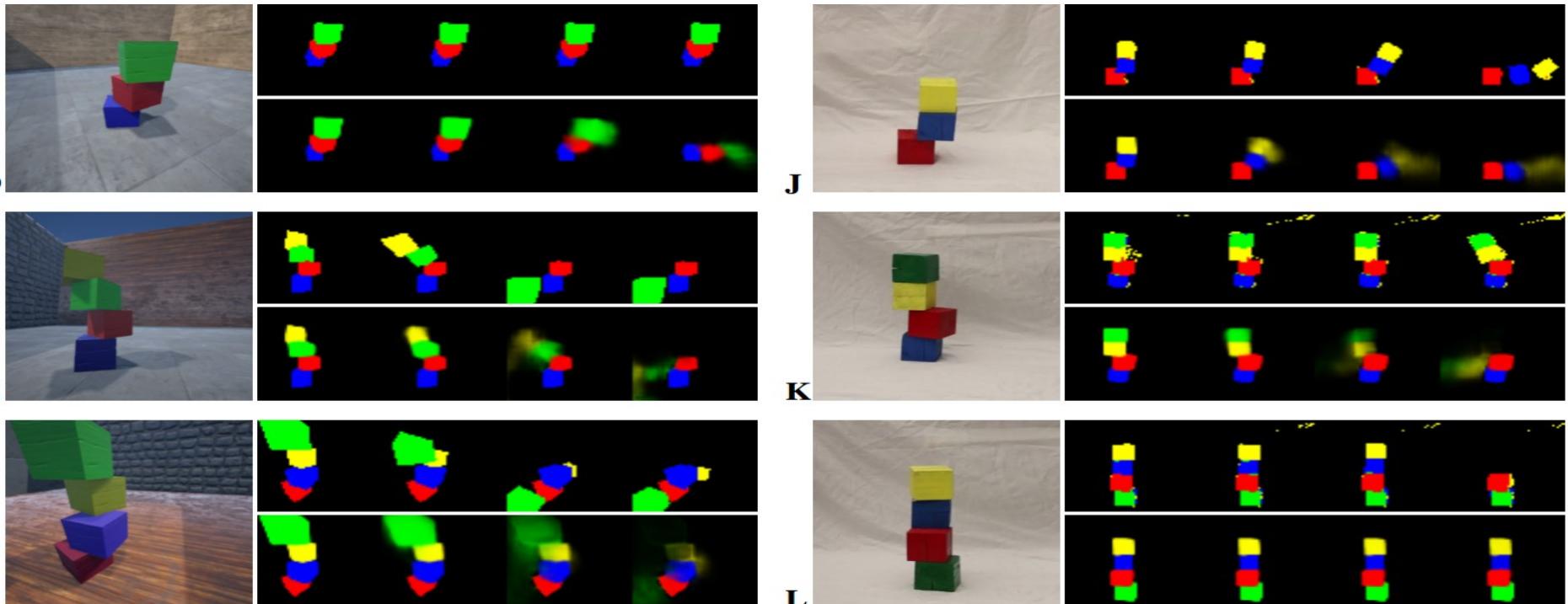
[Mathieu, Couprie, LeCun ICLR 2014]

[Luc, Couprie, LeCun, Verbeek ECCV 2018]

# PhysNet: tell me what happens next

[Lerer, Gross, Fergus ICML 2016, arxiv:1603.01312]

- ▶ ConvNet produces object masks that predict the trajectories of falling blocks. **Blurry predictions when uncertain**



# Self-supervised Adversarial Learning for Video Prediction

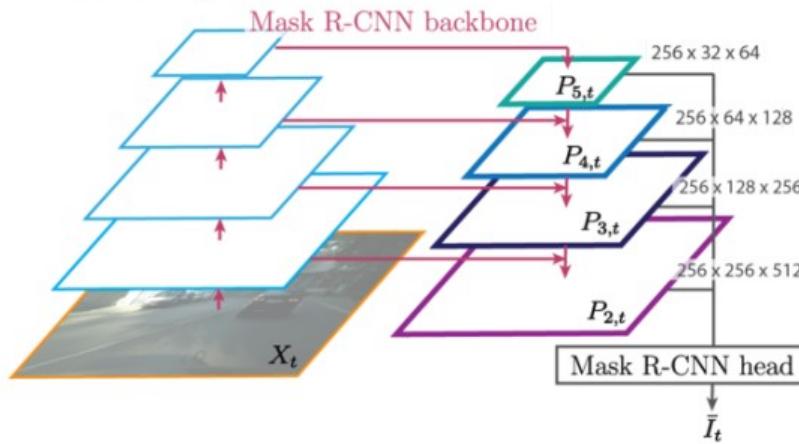
- ▶ Our brains are “prediction machines”
- ▶ Can we train machines to predict the future?
- ▶ Some success with “adversarial training”
  - ▶ [Mathieu, Couprie, LeCun ICLR 2016, arXiv:1511:05440]
- ▶ But we are far from a complete solution.



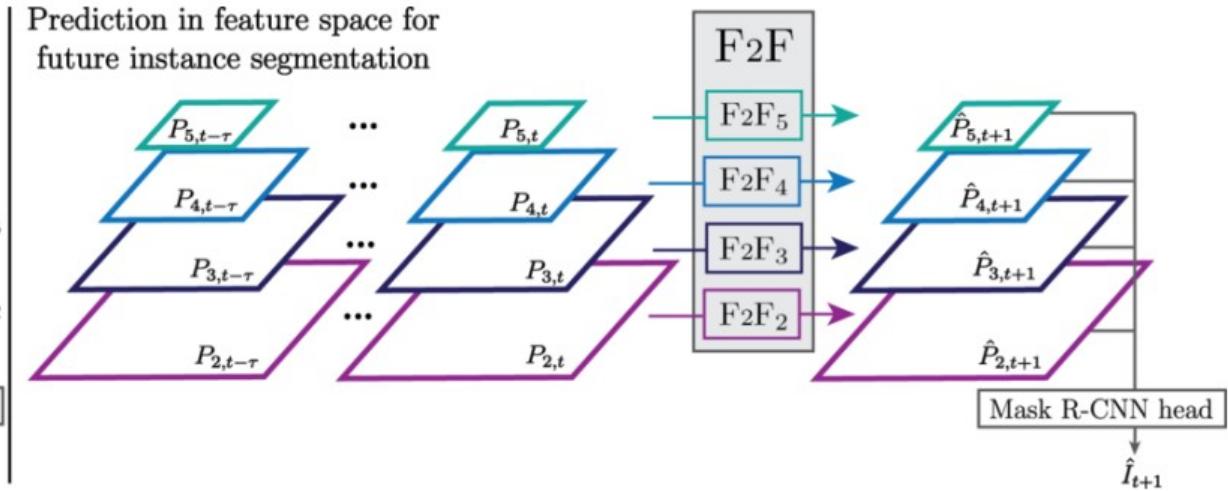
# Predicting Instance Segmentation Maps

- ▶ [Luc, Couprise, LeCun, Verbeek ECCV 2018]
- ▶ Mask R-CNN Feature Pyramid Network backbone
- ▶ Trained for instance segmentation on COCO
- ▶ Separate predictors for each feature level

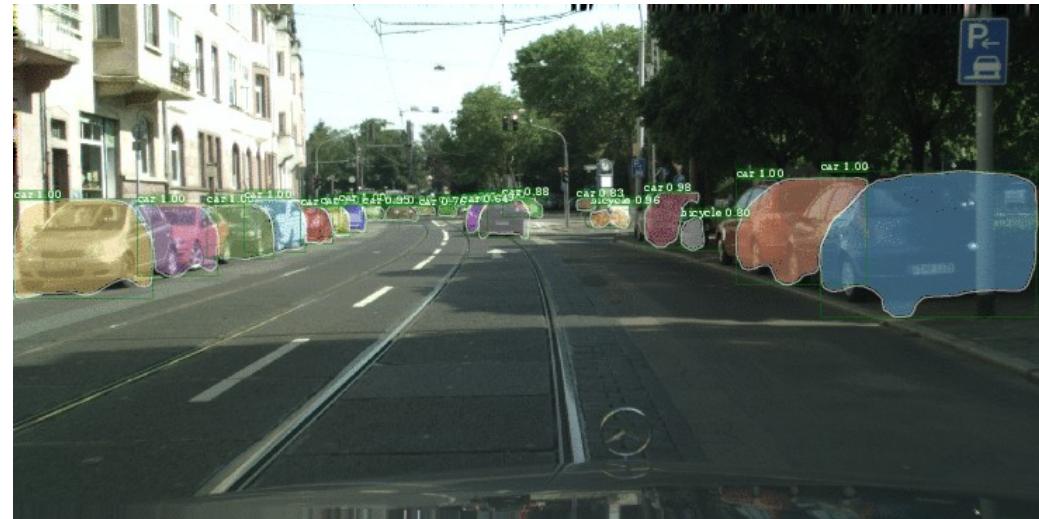
Instance segmentation



Prediction in feature space for future instance segmentation



# Long-term predictions (10 frames, 1.8 seconds)



[Luc, Couprie, LeCun, Verbeek ECCV 2018]

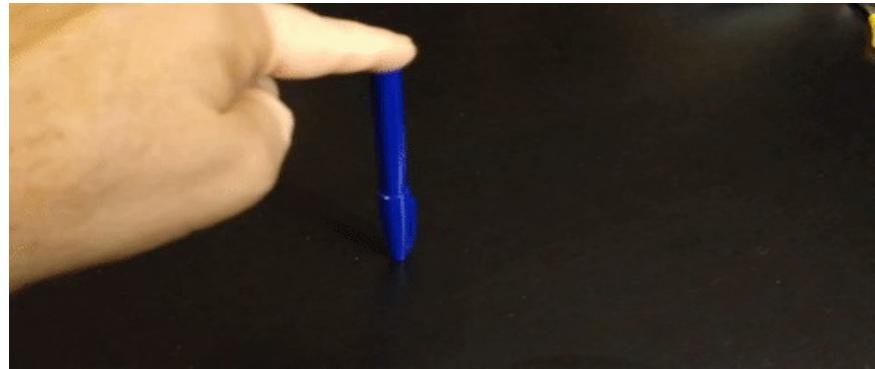


# Video Prediction With Uncertainty

Conditional Regularized Auto-Encoders

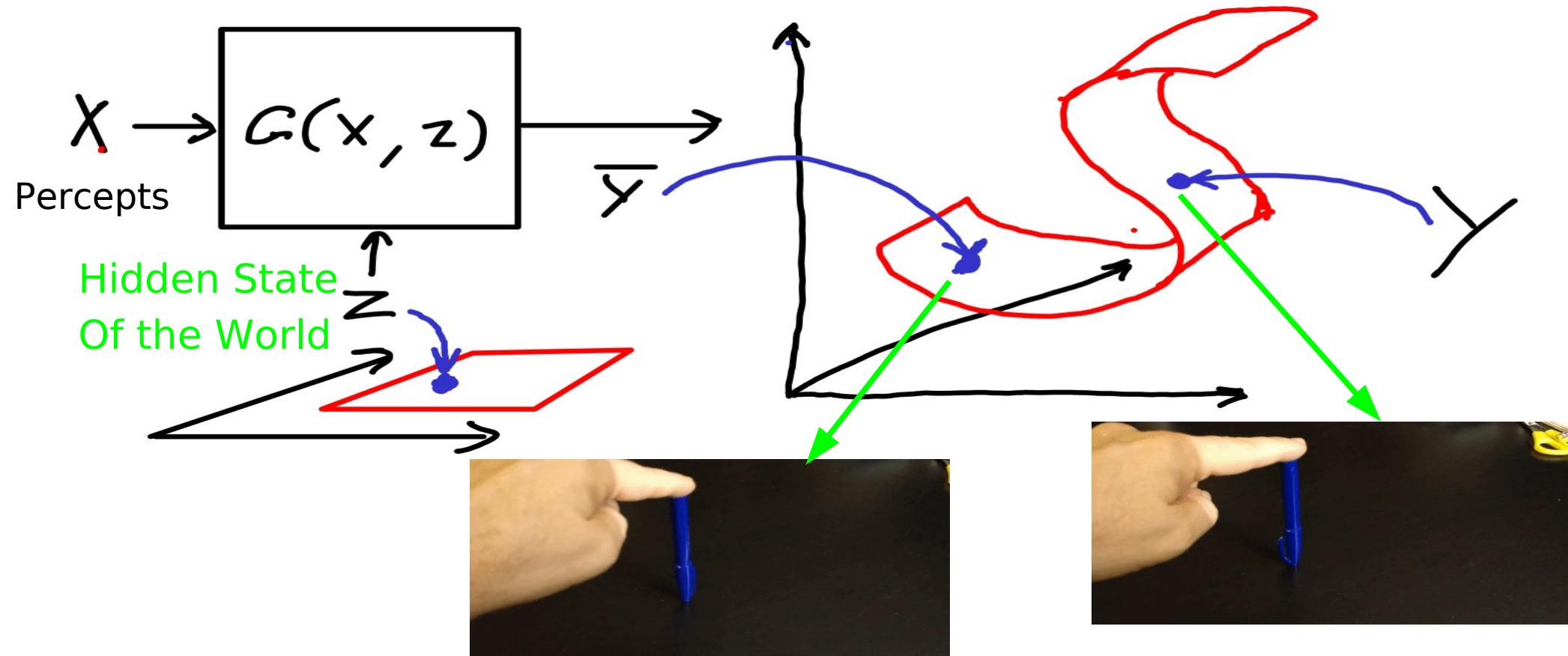
# The world is not entirely predictable / stochastic

- ▶ **Video prediction:**
- ▶ Multiple futures are possible.
- ▶ Training a system to make a single prediction results in “blurry” results
- ▶ the average of all the possible futures



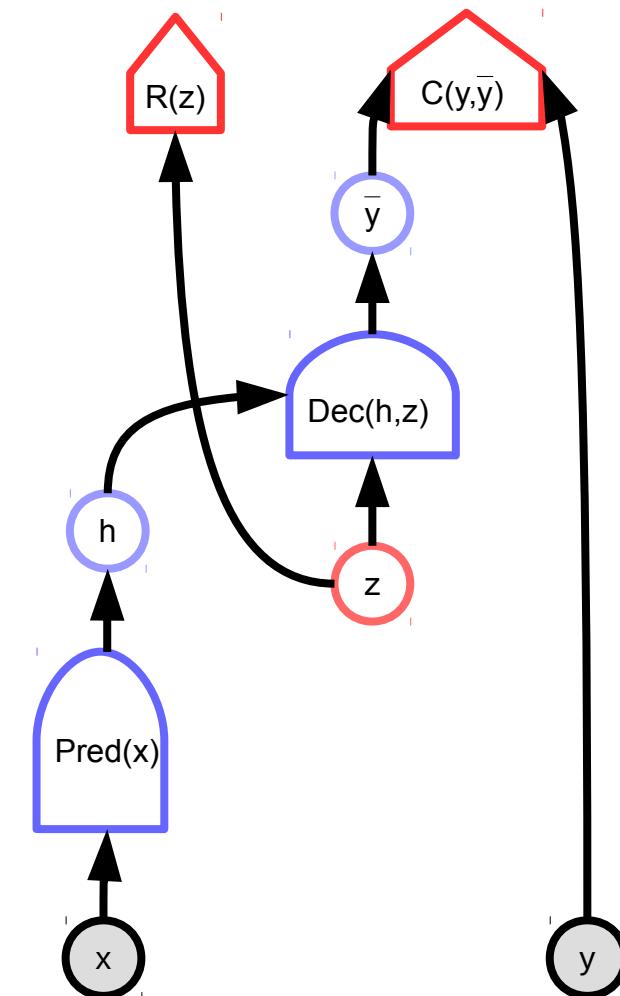
# We need to use latent-variable to pick up the uncertainty

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



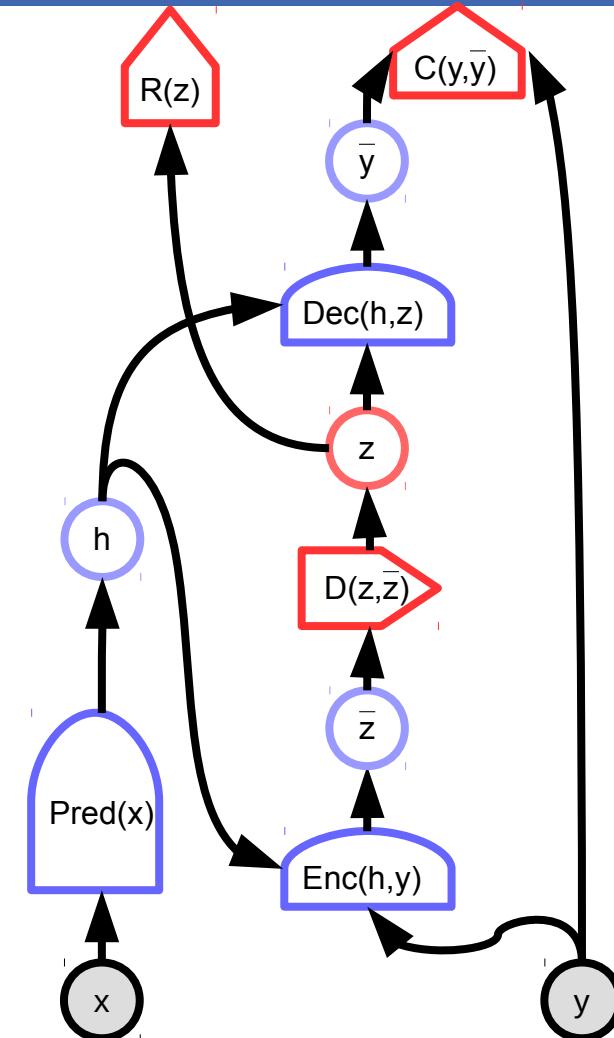
# Regularized Latent-Variable Predictive Models

- ▶ **x: observed variables**
- ▶ **y: variable to be predicted**
- ▶ **h: encoding of x**
- ▶ **z: latent variable**
  - ▶ Captures what is unpredictable
- ▶  $E(x,y,z) = C[y, \text{Dec}(\text{Pred}(x), z)] + R(z)$



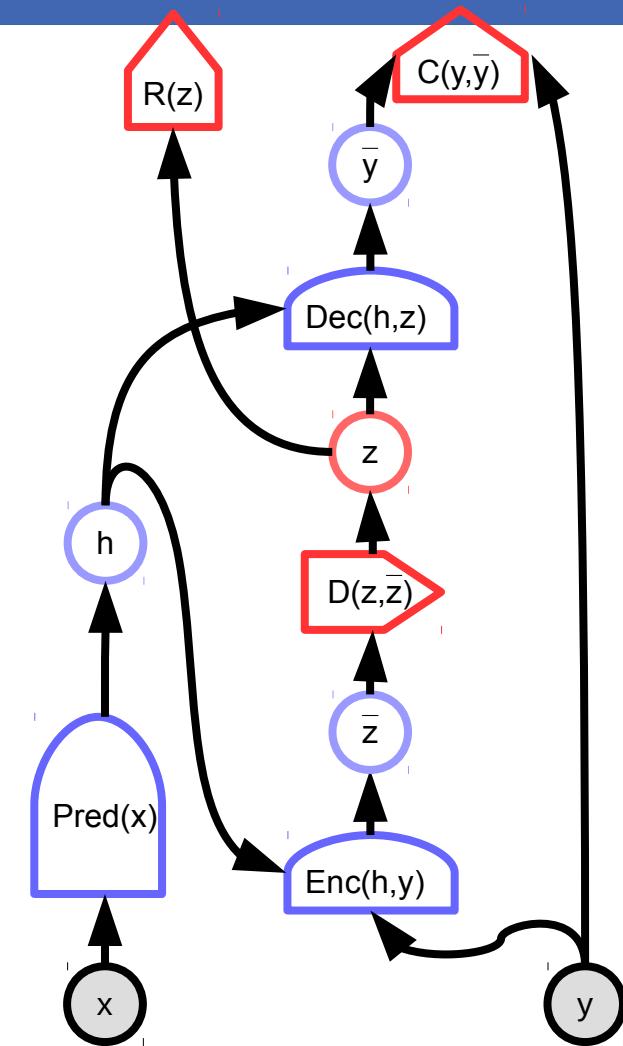
# Regularized Conditional Auto-Encoder

- ▶ **x: observed variables**
- ▶ **y: variable to be predicted**
- ▶ **h: encoding of x**
- ▶ **z: latent variable**
- ▶ Captures what is unpredictable



# Regularized Auto-Encoders

- ▶  $y$ : variable to be predicted
- ▶  $h$ : encoding of  $x$
- ▶  $z$ : latent variable
  
- ▶ **Sparse Auto-Encoder**
  - ▶  $E(y,z) = C[y, \text{Dec}(\text{Enc}(y))] + D[z, \text{Enc}(y)] + R(z)$
  - ▶ [Kavukcuoglu CVPR 2009] [Gregor ICML 2010]





# Learning Models of the World

Learning motor skills with no interaction with the real world

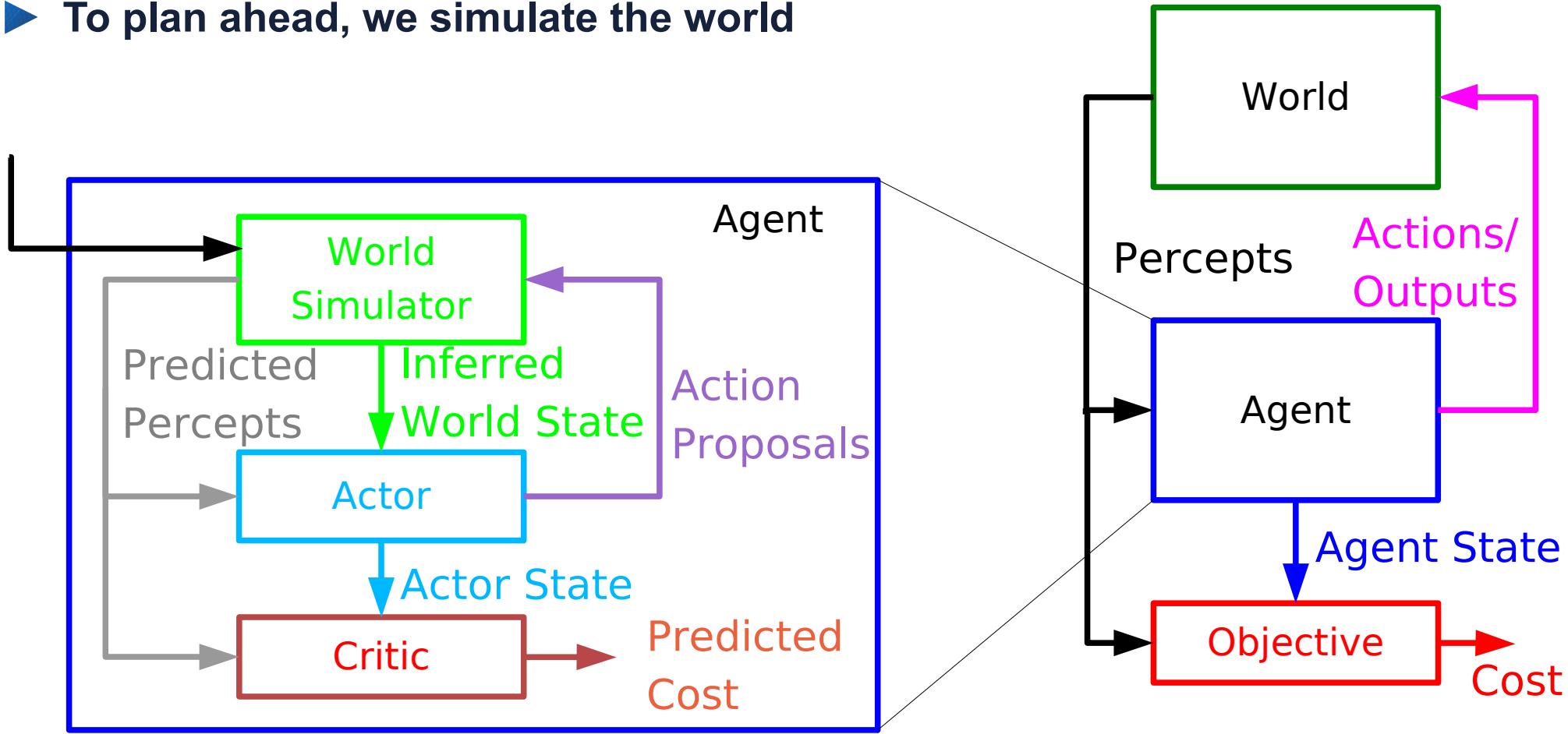
[Henaff, Canziani, LeCun ICLR 2019]

[Henaff, Zhao, LeCun ArXiv:1711.04994]

[Henaff, Whitney, LeCun Arxiv:1705.07177]

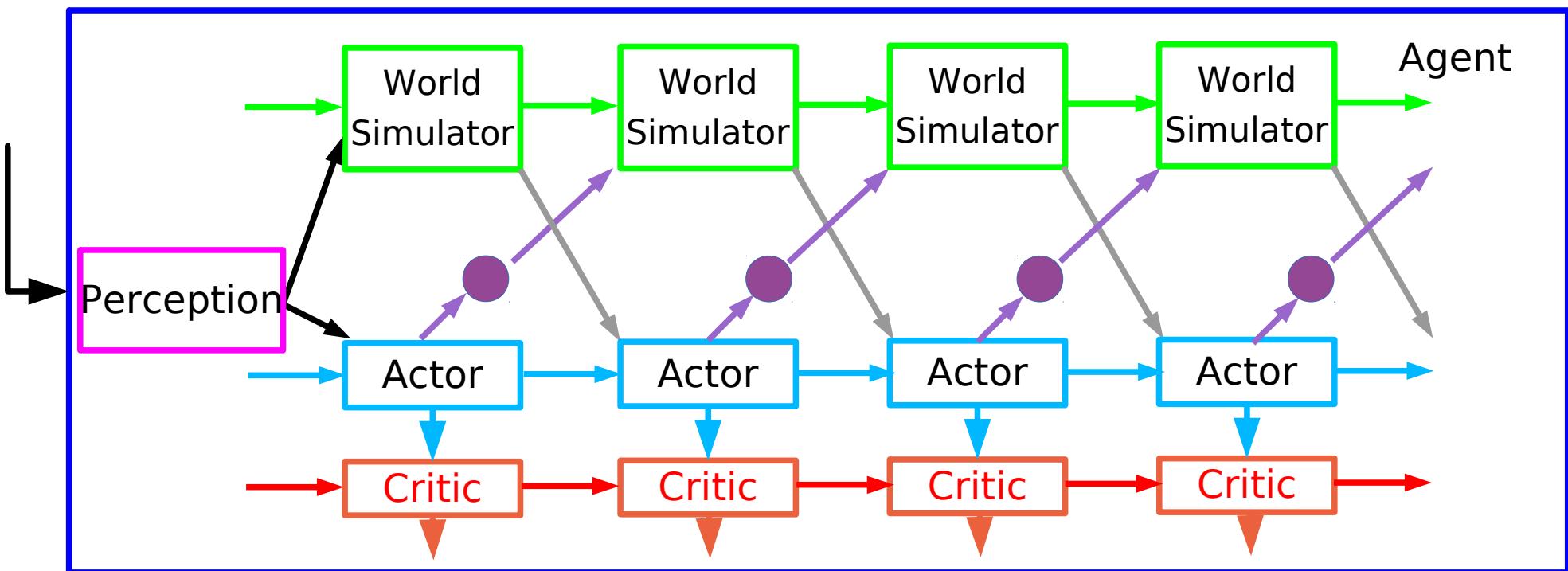
# Planning Requires Prediction

- ▶ To plan ahead, we simulate the world



# Training the Actor with Optimized Action Sequences

- ▶ 1. Find action sequence through optimization
- ▶ 2. Use sequence as target to train the actor
- ▶ Over time we get a compact policy that requires no run-time optimization

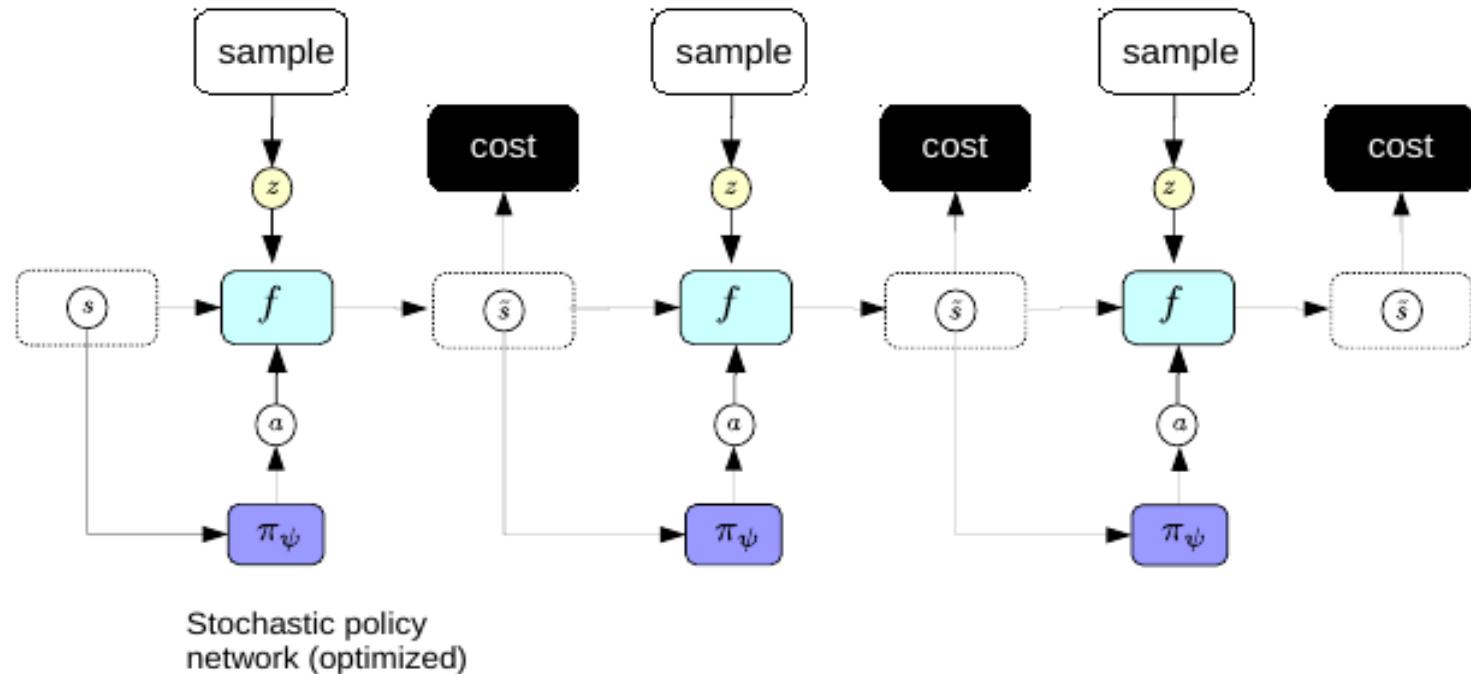


# Planning/learning using a self-supervised predictive world model

- ▶ Feed initial state
- ▶ Run the forward model
- ▶ Backpropagate gradient of cost
- ▶ Act
  - ▶ (model-predictive control)

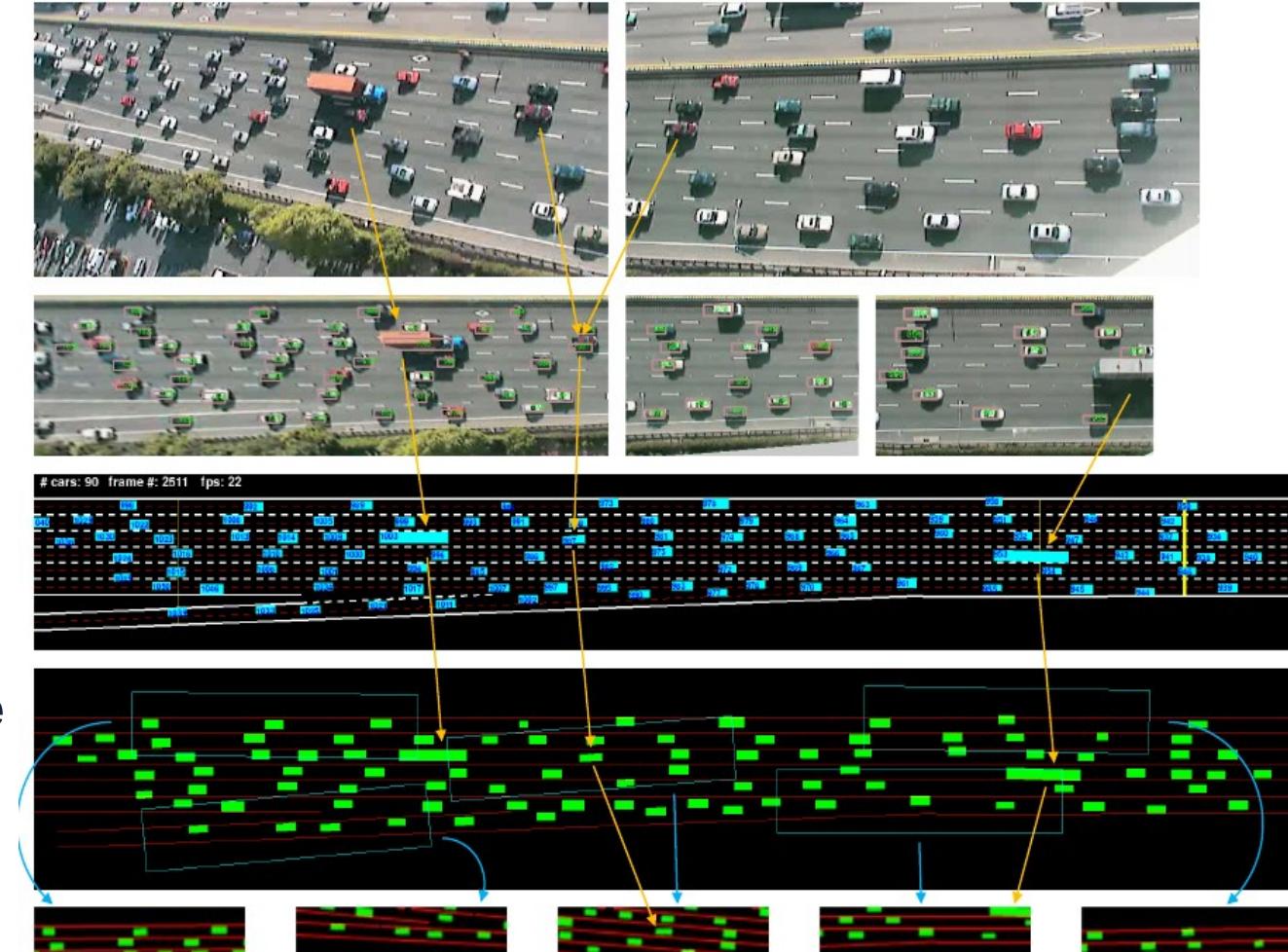
or

- ▶ Use the gradient to train a policy network.
- ▶ Iterate



# Using Forward Models to Plan (and to learn to drive)

- ▶ Overhead camera on highway.
- ▶ Vehicles are tracked
- ▶ A “state” is a pixel representation of a rectangular window centered around each car.
- ▶ Forward model is trained to predict how every car moves relative to the central car.
- ▶ steering and acceleration are computed

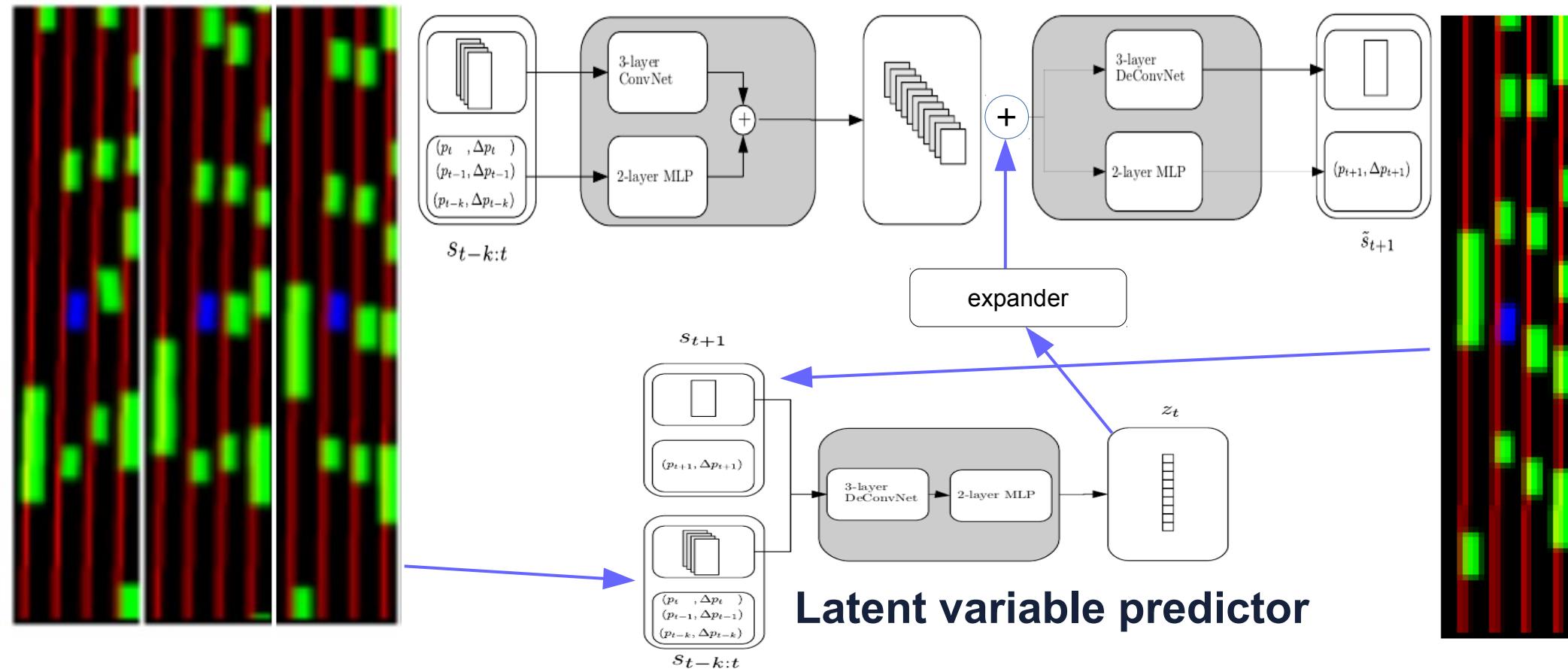


# Forward Model Architecture

## ► Architecture:

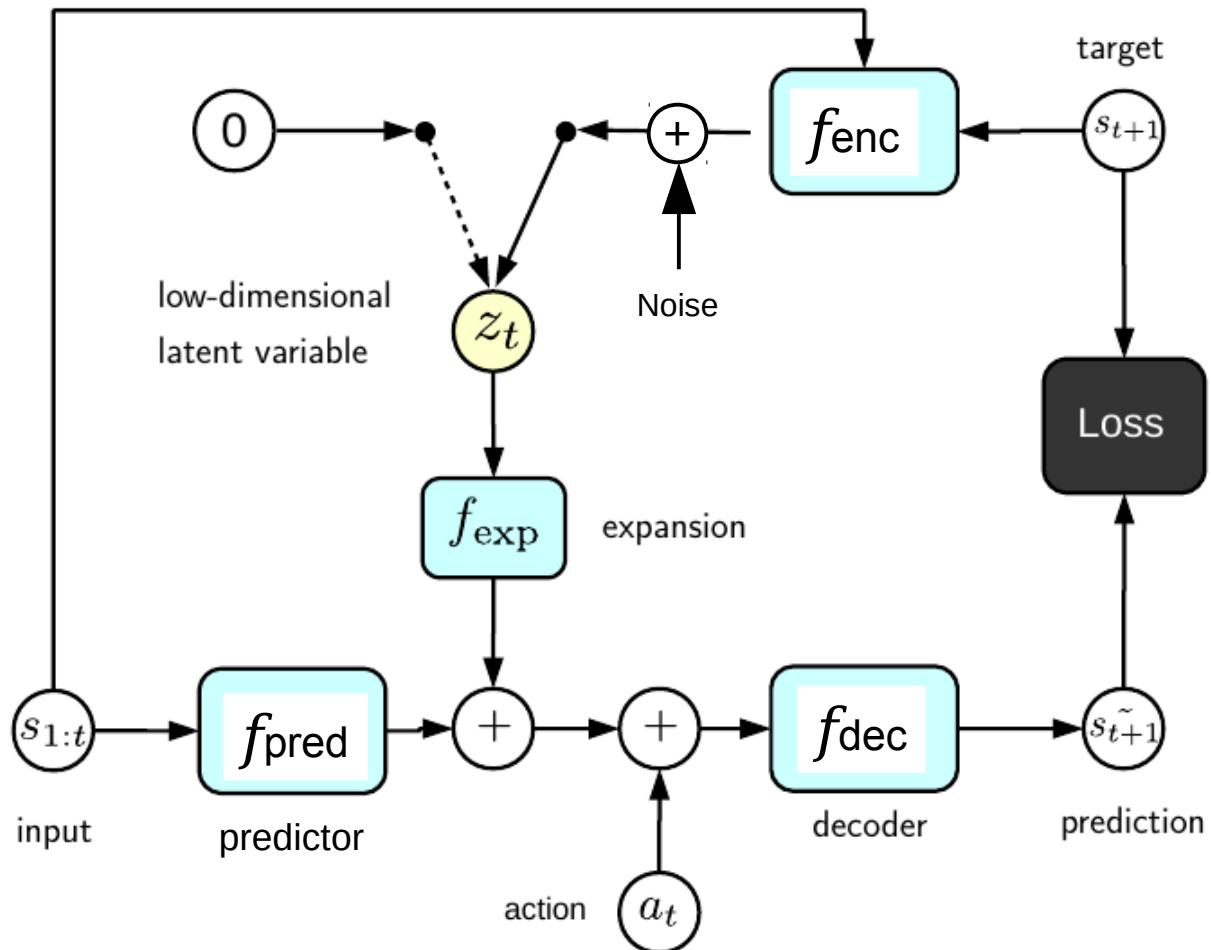
### Encoder

### Decoder

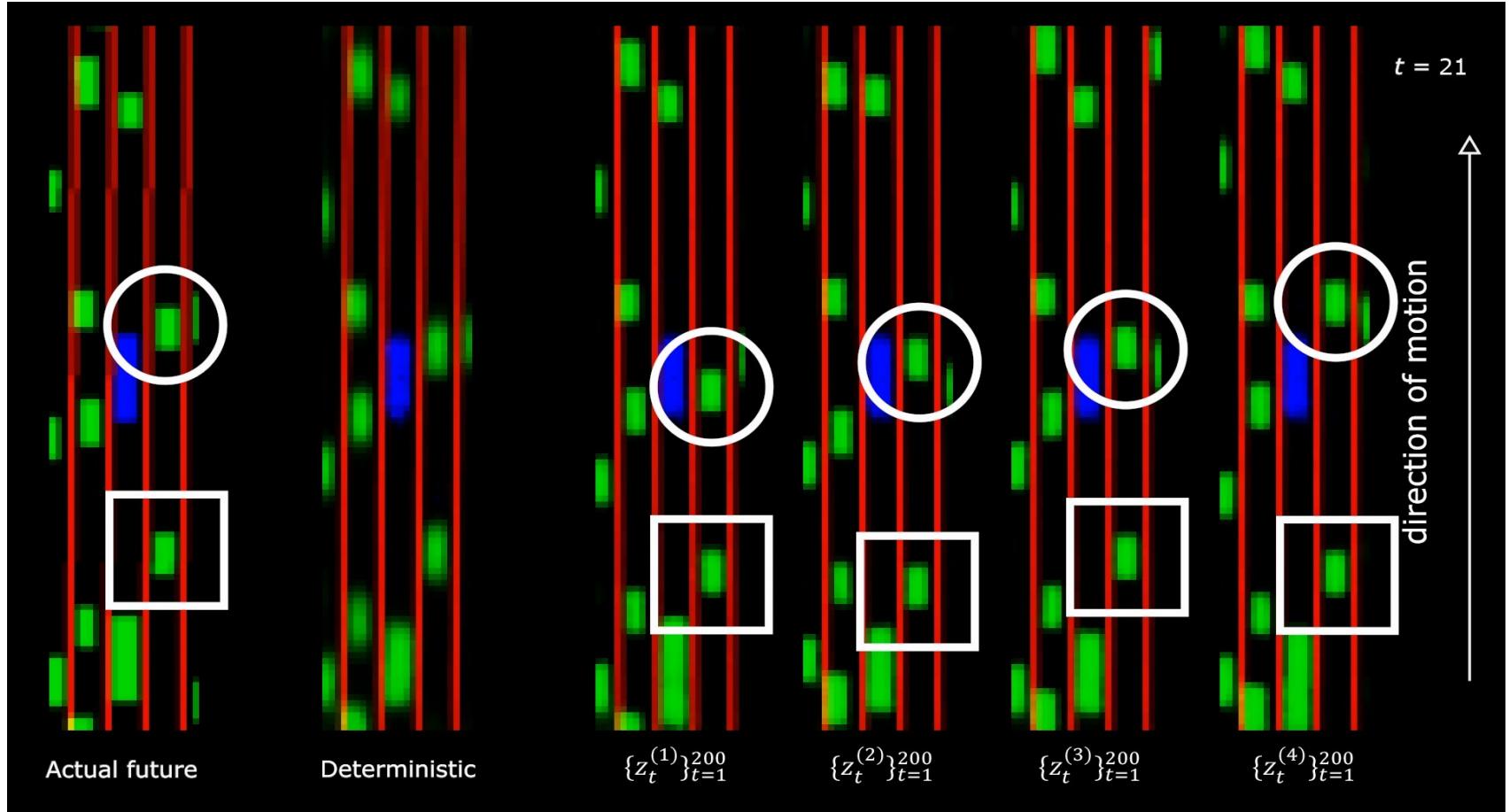


# Stochastic Forward Modeling: regularized latent variable model

- ▶ Latent variable is predicted from the target.
- ▶ The latent variable is set to zero half the time during training (drop out) and corrupted with noise
- ▶ The model predicts as much as it can without the latent var.
- ▶ The latent var corrects the residual error.

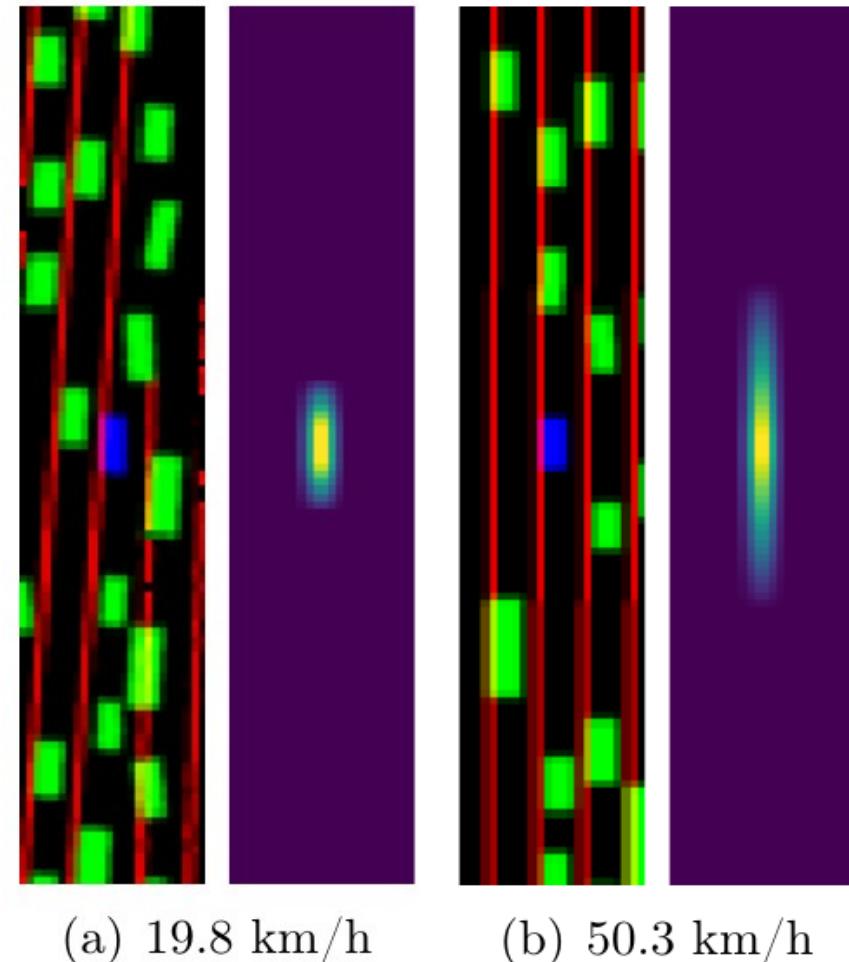


# Actual, Deterministic, VAE+Dropout Predictor/encoder



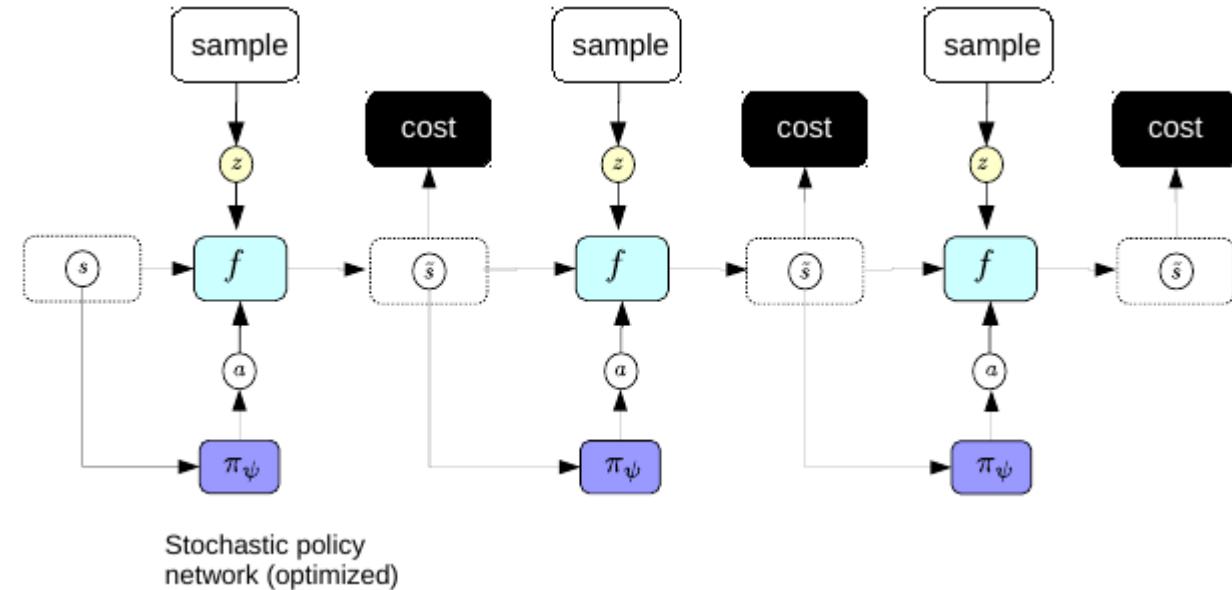
# Cost optimized for Planning & Policy Learning

- ▶ **Differentiable cost function**
  - ▶ Increases as car deviates from lane
  - ▶ Increases as car gets too close to other cars nearby in a speed-dependent way
- ▶ **Uncertainty cost:**
  - ▶ Increases when the costs from multiple predictions (obtained through sampling of drop-out) have high variance.
  - ▶ Prevents the system from exploring unknown/unpredictable configurations that may have low cost.



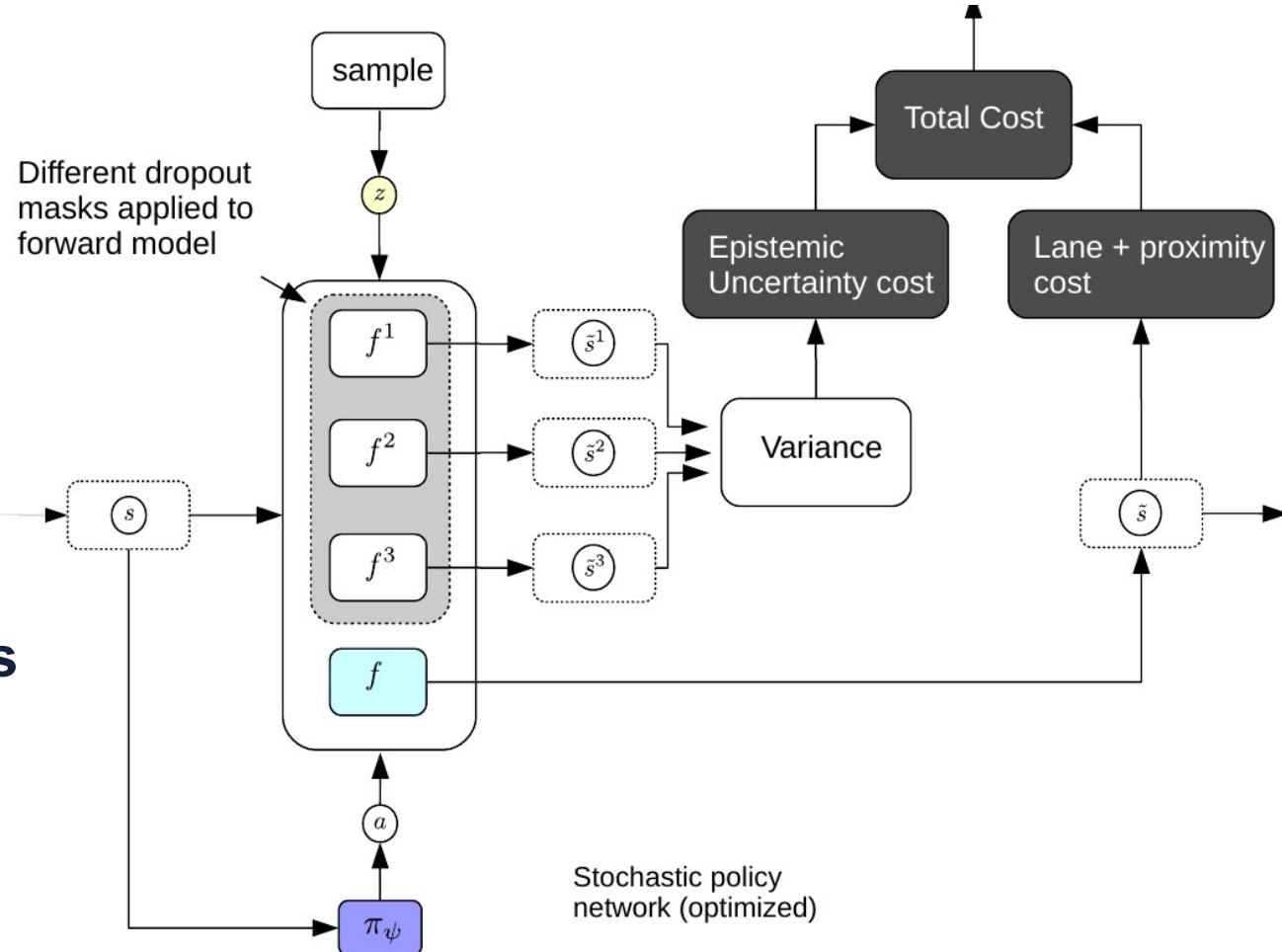
# Learning to Drive by Simulating it in your Head

- ▶ Feed initial state
- ▶ Sample latent variable sequences of length 20
- ▶ Run the forward model with these sequences
- ▶ Backpropagate gradient of cost to train a policy network.
- ▶ Iterate
  
- ▶ No need for planning at run time.

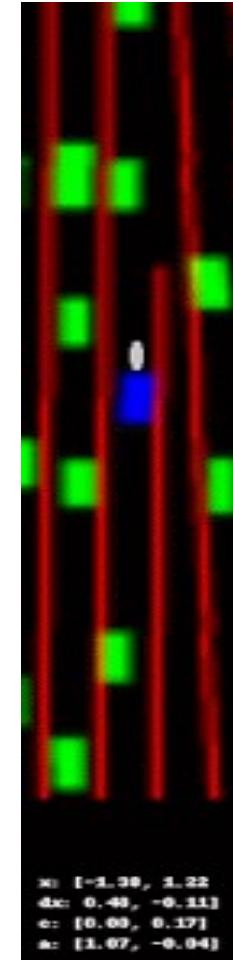
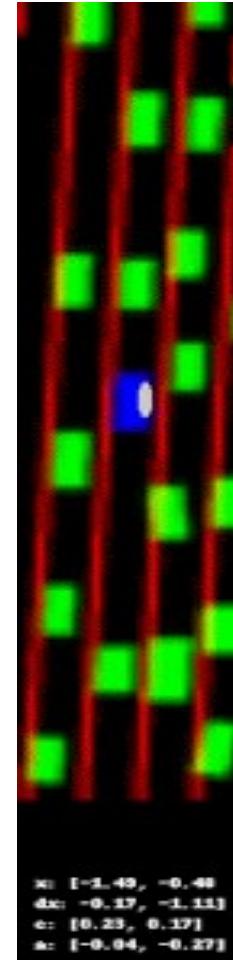
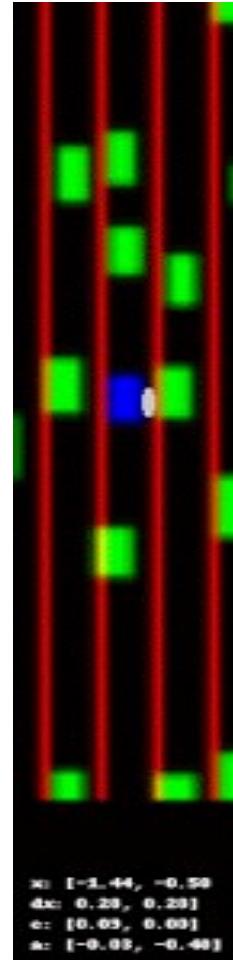
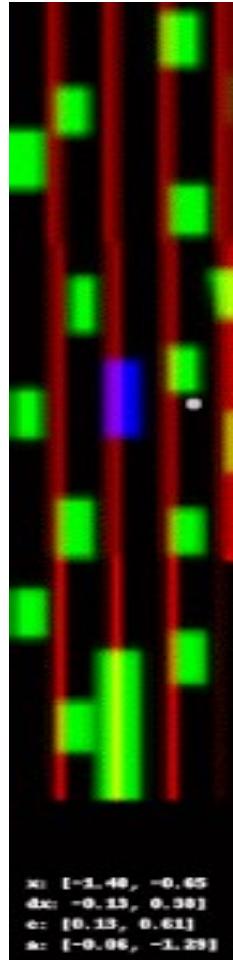
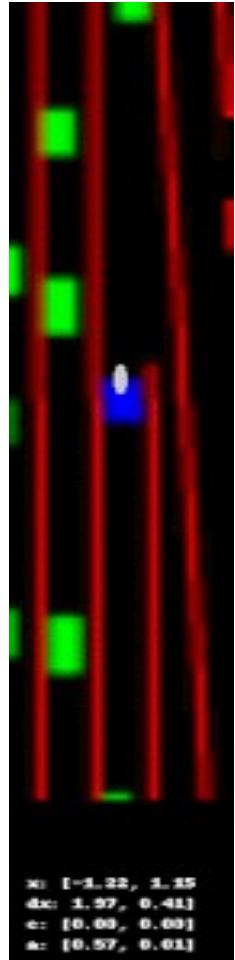


# Adding an Uncertainty Cost (doesn't work without it)

- ▶ Estimates epistemic uncertainty
- ▶ Samples multiple dropouts in forward model
- ▶ Computes variance of predictions (differentiably)
- ▶ Train the policy network to minimize the lane&proximity cost plus the uncertainty cost.
- ▶ Avoids unpredictable outcomes

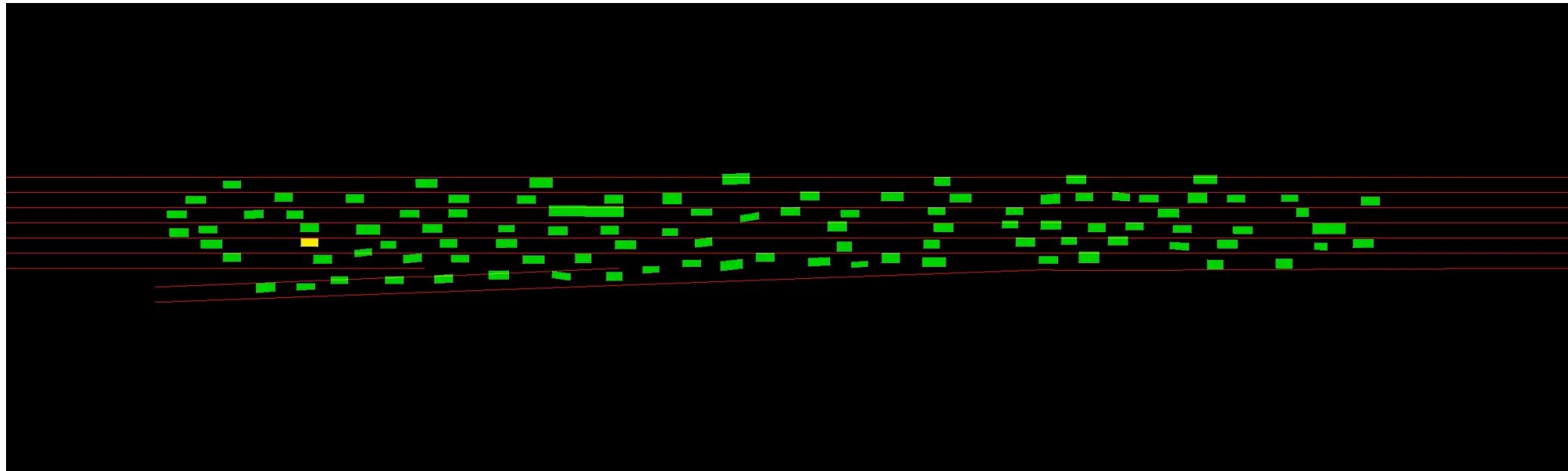


# Driving an Invisible Car in “Real” Traffic

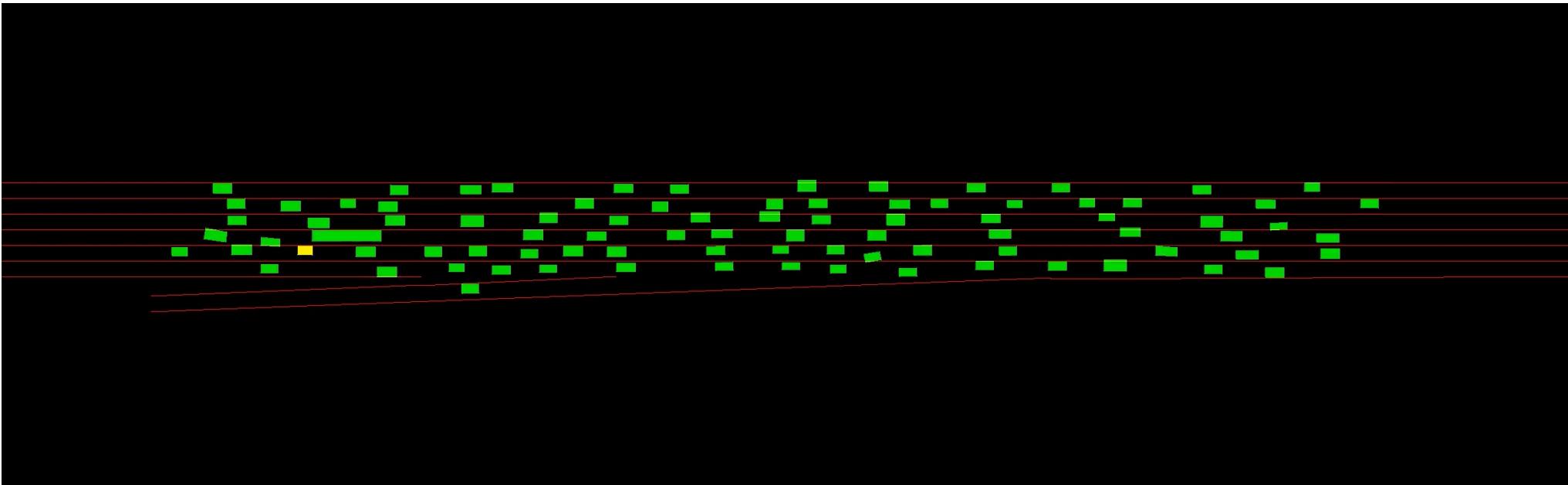


# Driving!

- ▶ Yellow: real car
- ▶ Blue: bot-driven car



- ▶ Yellow: real car
- ▶ Blue: bot-driven car



• \ |

# Thank you