Towards one-shot learning via Attention

Andrej Lúčny

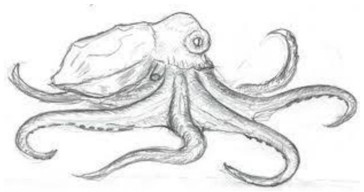
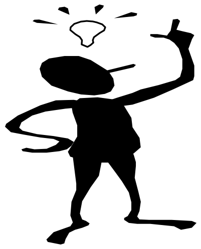
Department of Applied Informatics

Comenius University, Bratislava, Slovakia

lucny@fmph.uniba.sk

http://dai.fmph.uniba.sk/w/Andrej_Lucny

Kinds of Minds

			learning:
• Darwinian	∞		none
• Skinnerian	n		gradual
• Popperian	1		one-shot
• Gregorian	0		language-based

Towards one-shot learning

gradual

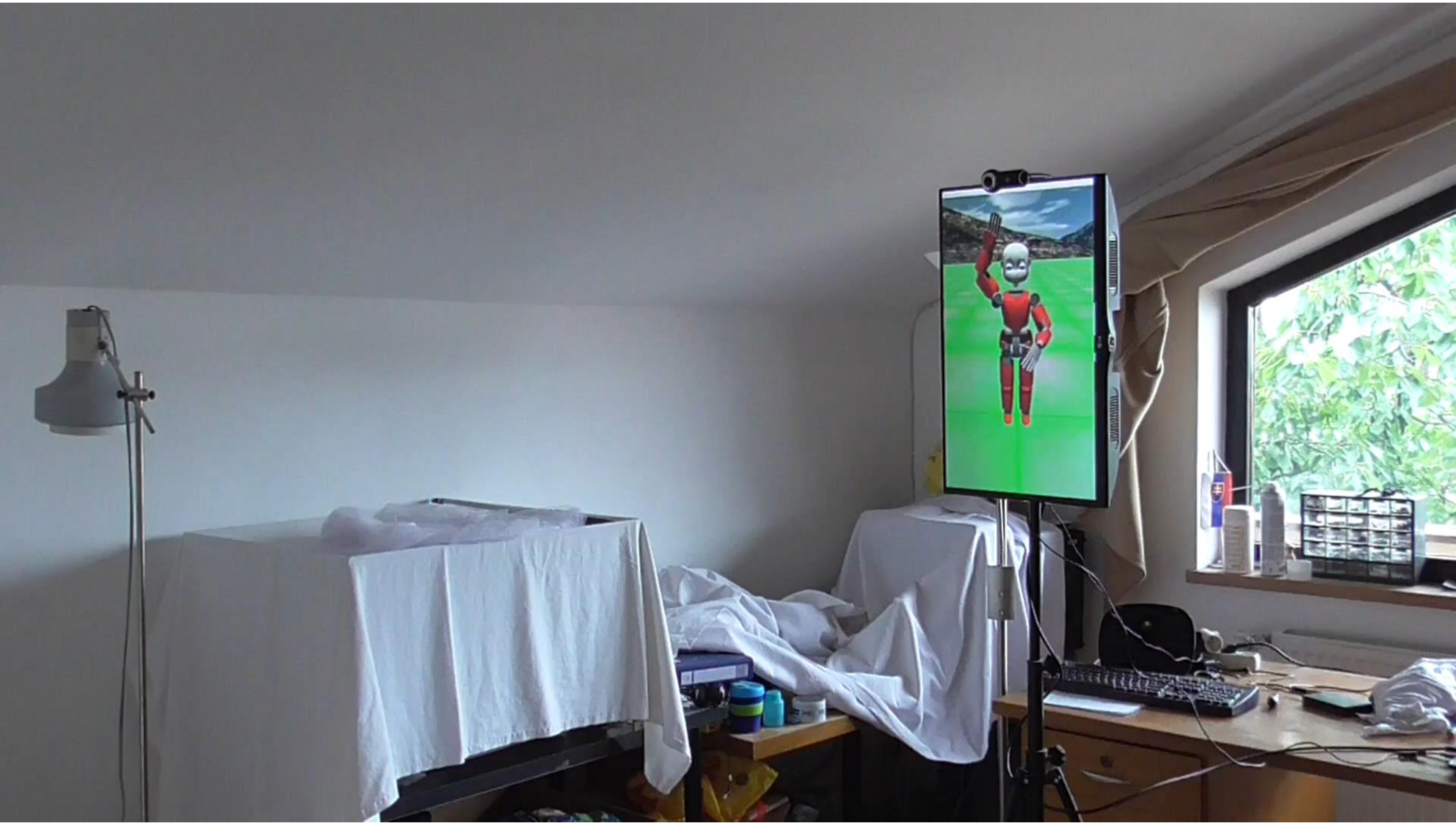
$\Downarrow$

one-shot

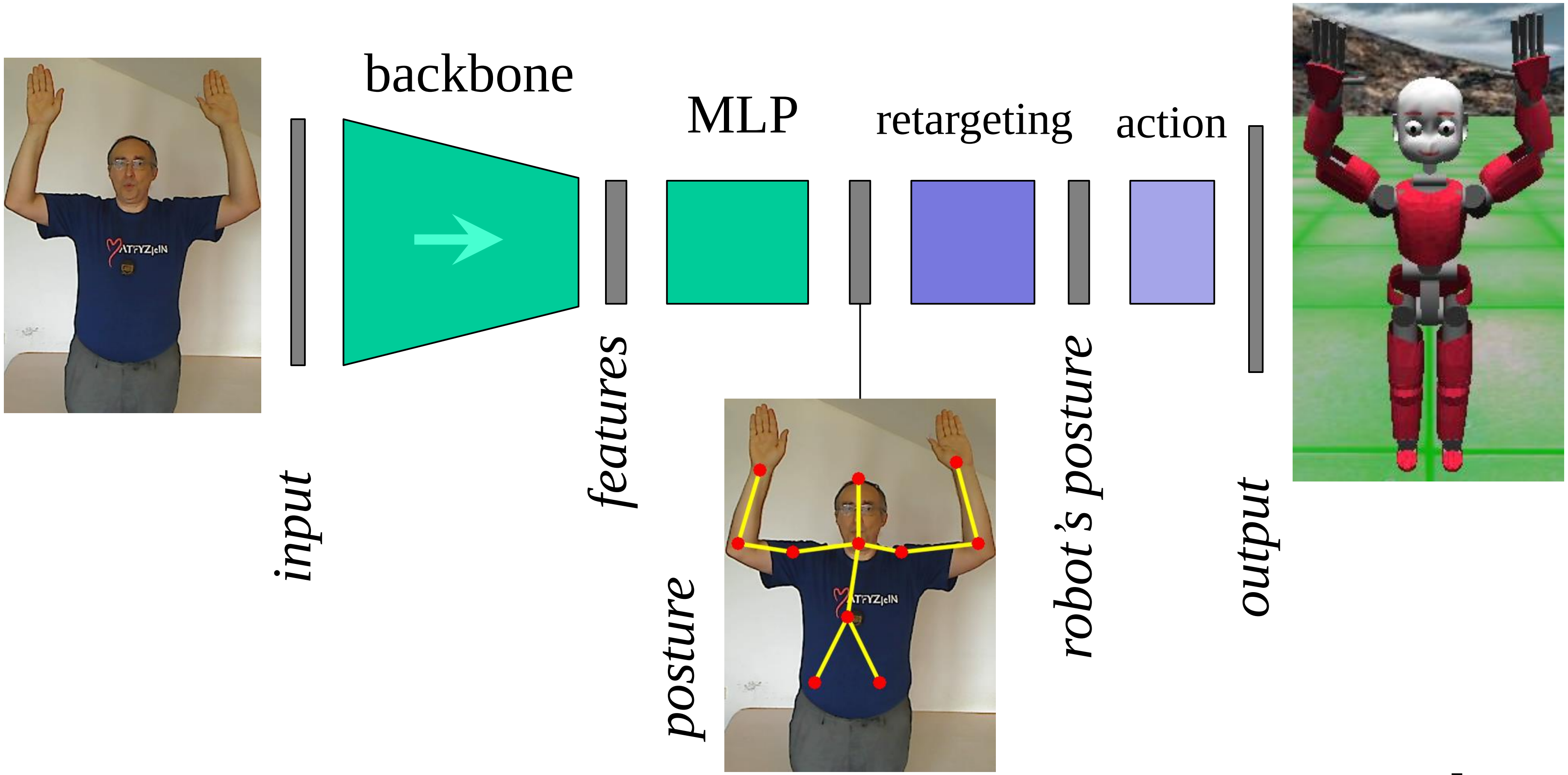
We investigate how to employ gradually learned components for starting one-shot learning.

We demonstrate that it is pretty easy with the Attention mechanism if we have the components (encoders, decoders) of good quality.

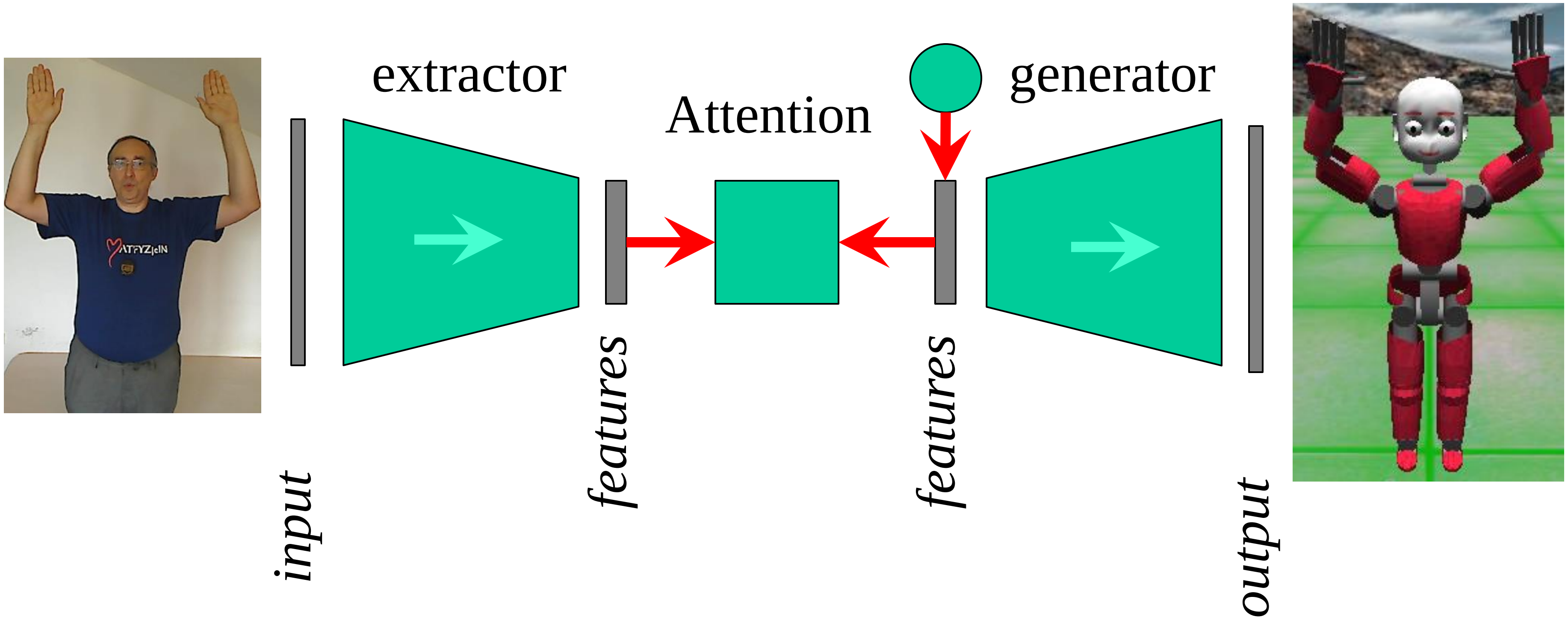
An example: The imitation game



Classic solution – the wired imitation



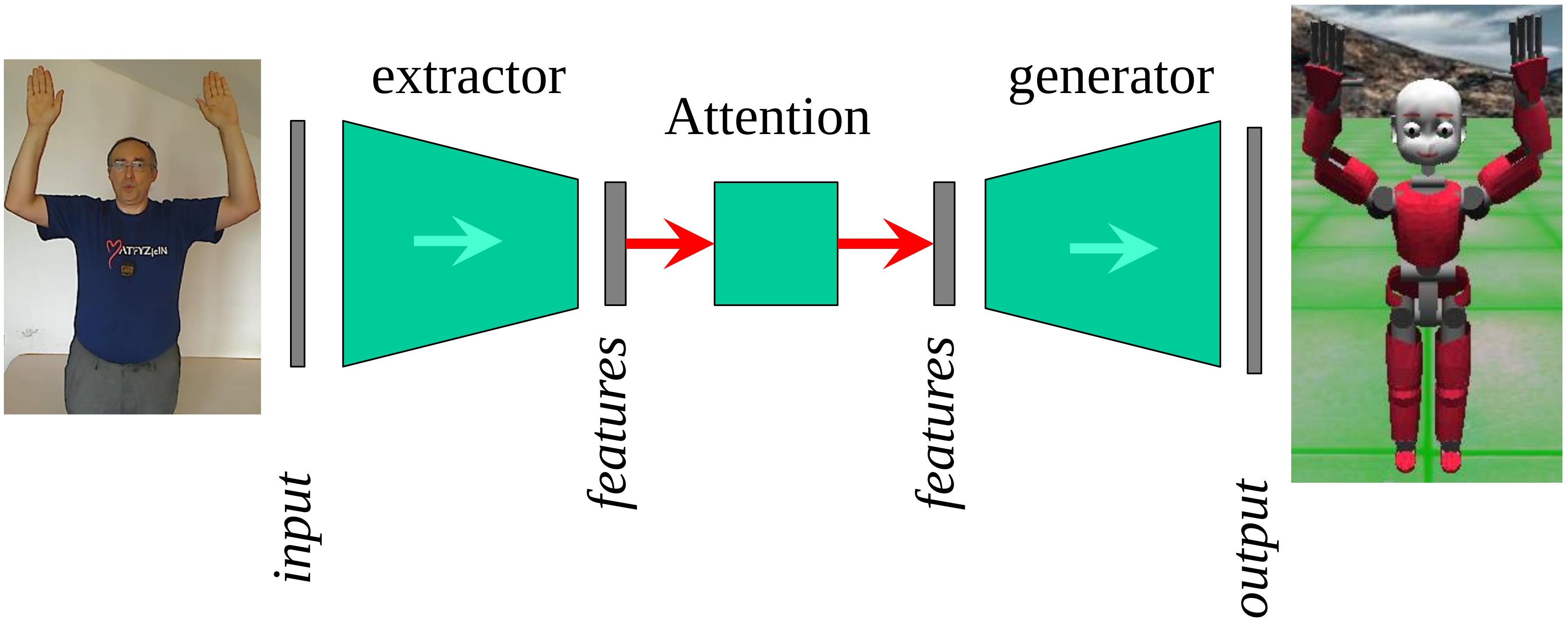
Our approach: One-shot learning



Phase 1: ACQUIRE

no postures

Our approach: One-shot learning

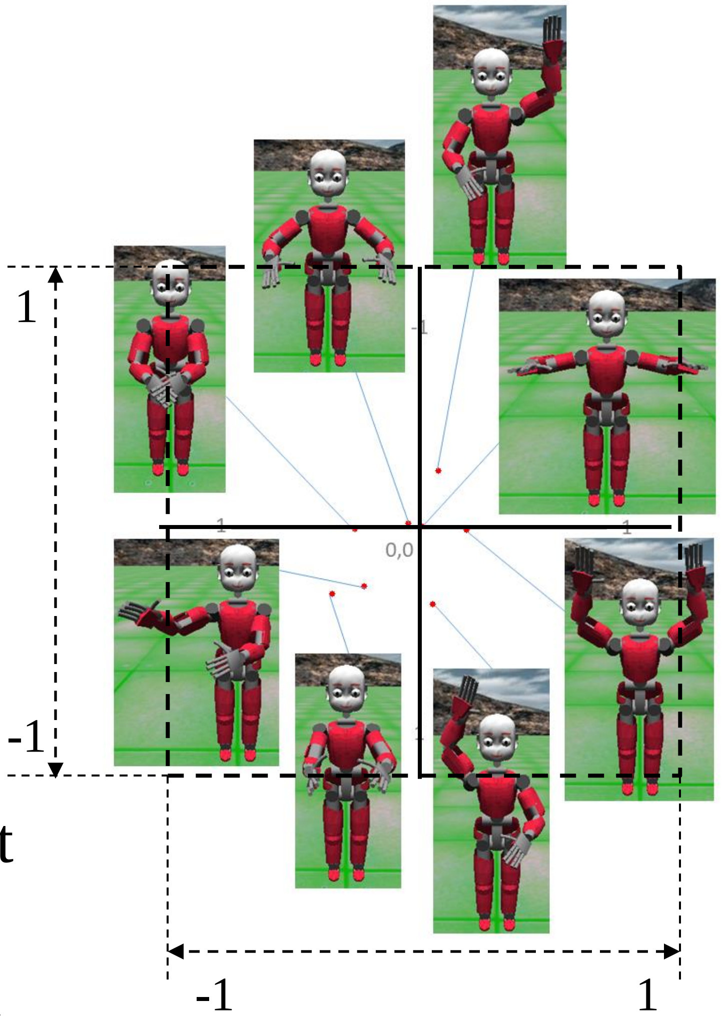


Phase 2: USE

no postures

Features

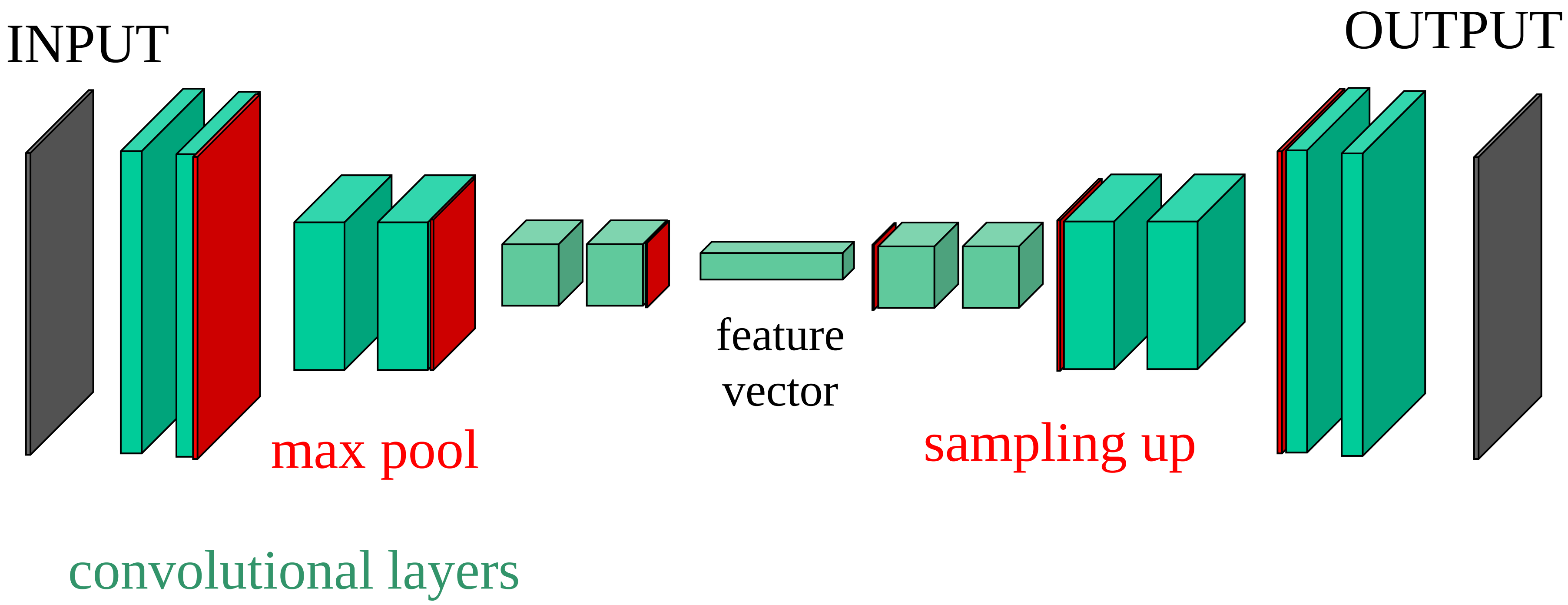
- We turn inputs to feature vectors (extraction) or feature vectors to outputs (generation)
- The features contain the compressed information
- The dimension of the feature vector is much lower than the dimension of the input or output
- All possible feature vectors form the latent space
- The essential faculty of features is that each possible value corresponds to a reasonable instance of the represented data



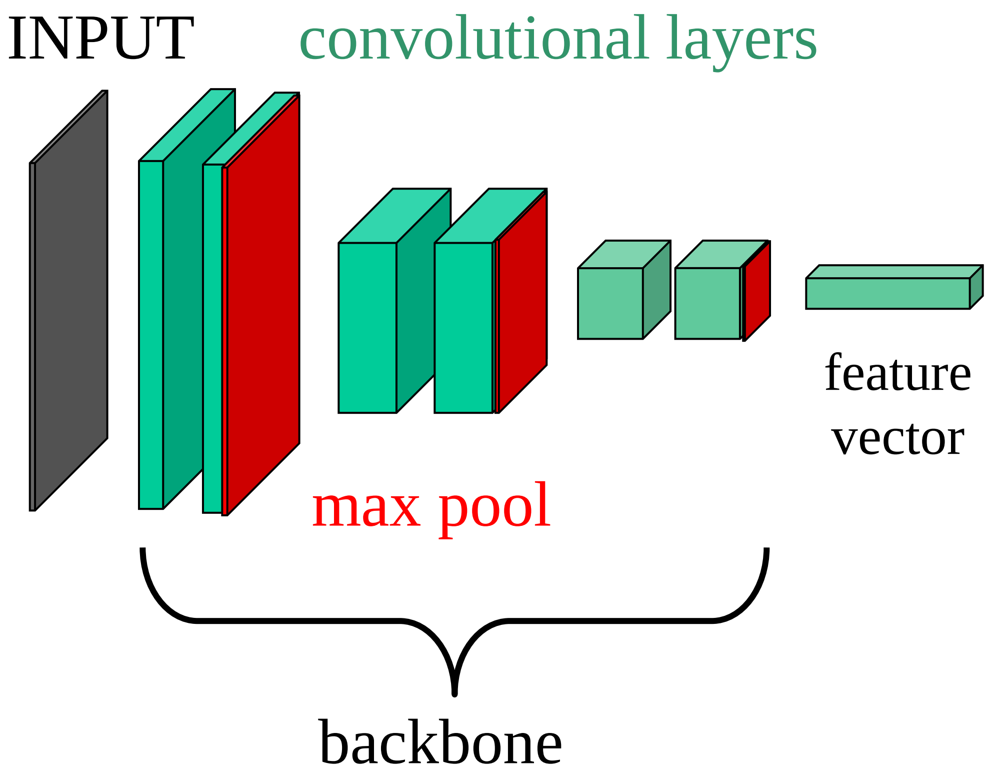
Extractors and Generators

- How can a system develop proper extractors and generators?
- We can rely on several methods of self-supervised learning. These methods work with a dataset but do not solicit any annotation.
- Namely, we can employ:
  - autoencoders
    - convolutional autoencoders
    - variational autoencoders
  - metric learning:
    - self-supervised siamese networks
    - self-supervised teacher-student networks

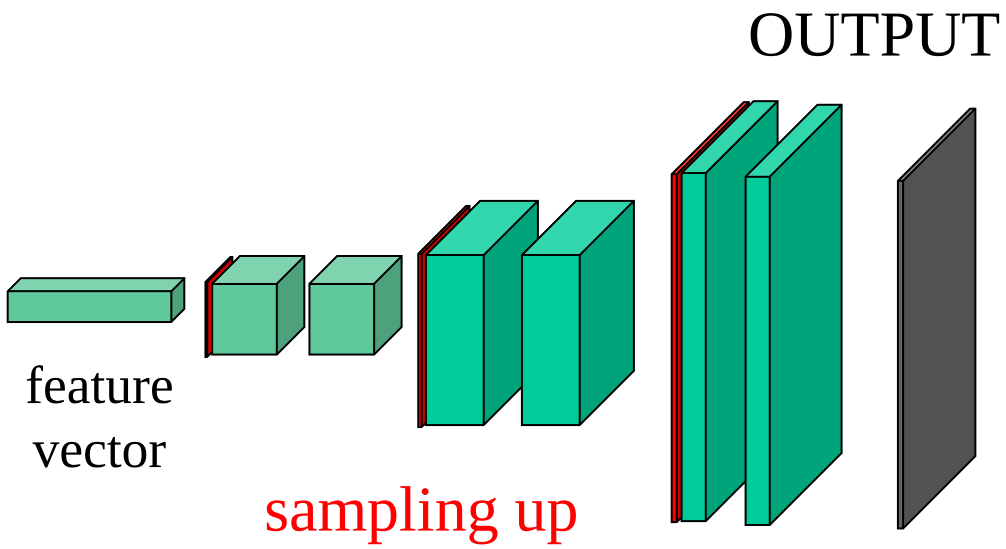
The Convolutional Autoencoder



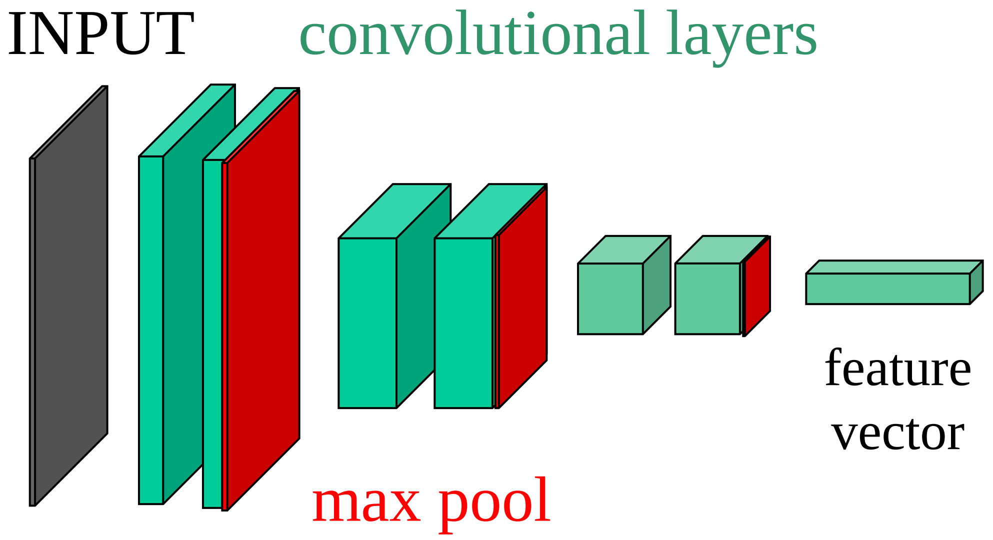
Encoder (Extractor)



Decoder (Generator)

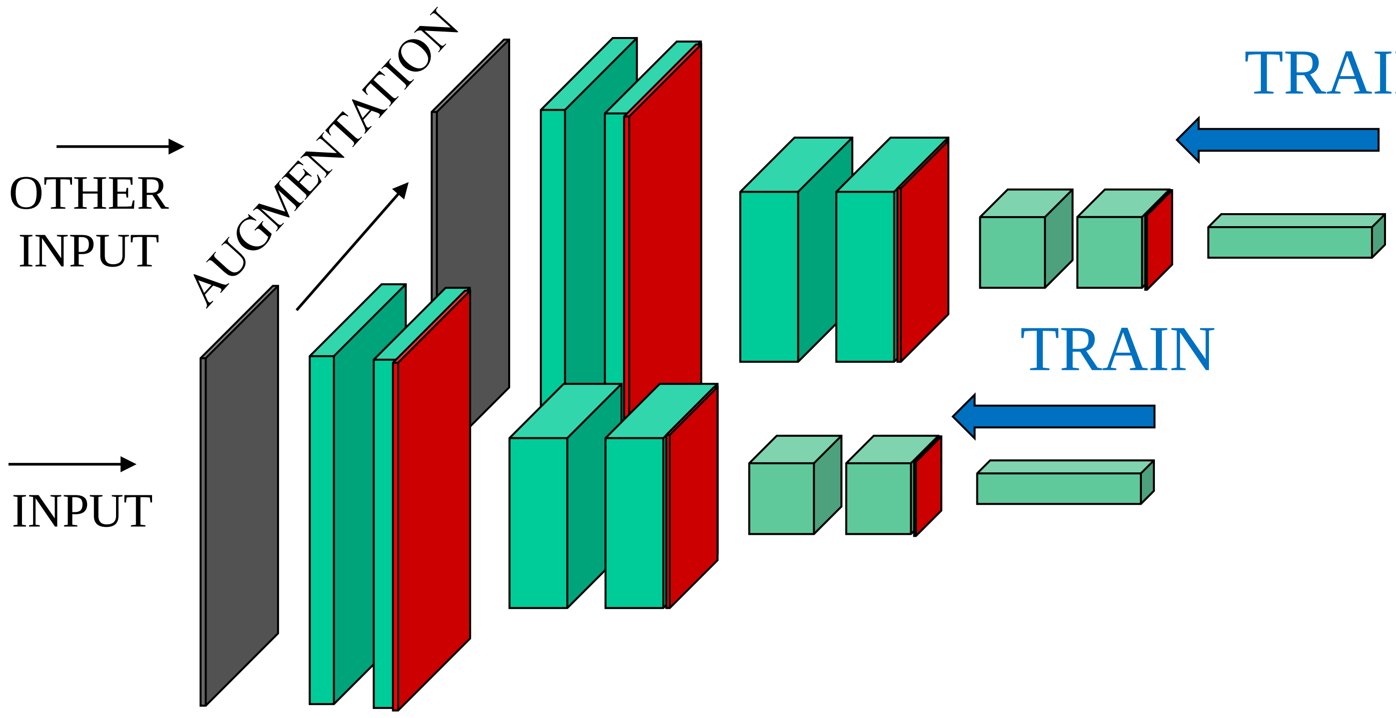


Metric learning



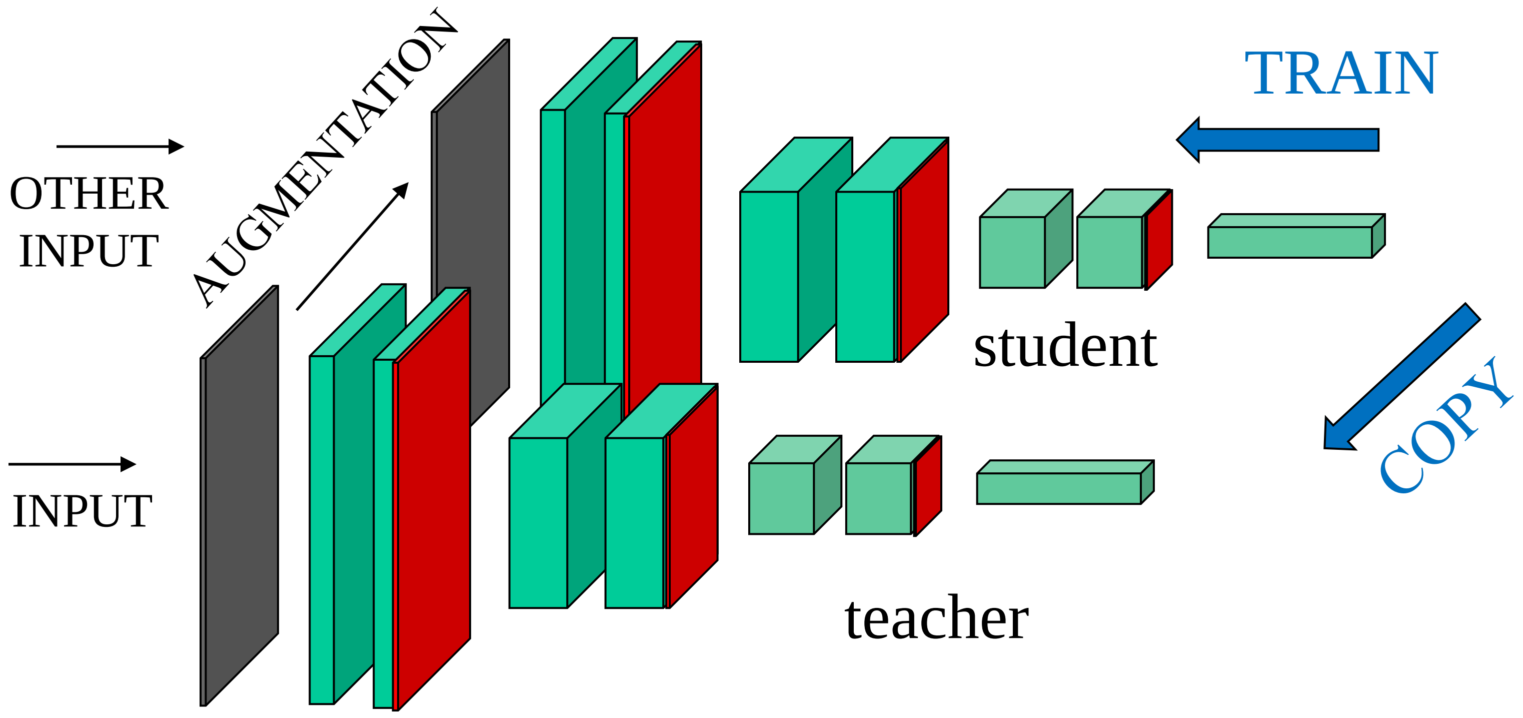
We train only backbone. We do not know the exact feature vector we aim to get. But we can define a metric that the feature vectors corresponding to the instances from the dataset should hold.

Self-supervised Siamese network



- When we feed input and its augmentation, we solicit to get similar feature vectors.
- When we provide two different inputs, we solicit to get other feature vectors.

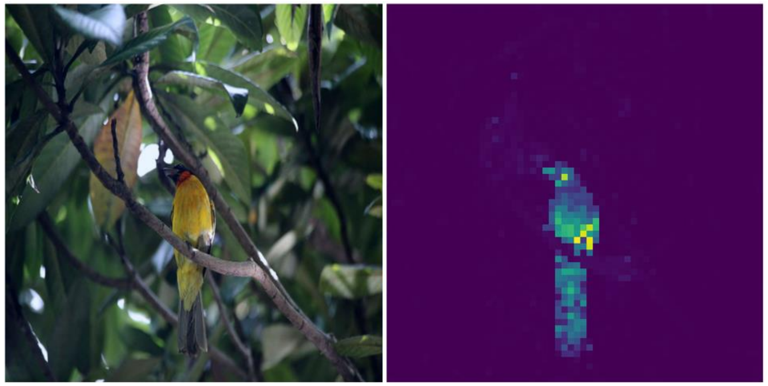
Self-supervised Teacher-Student network



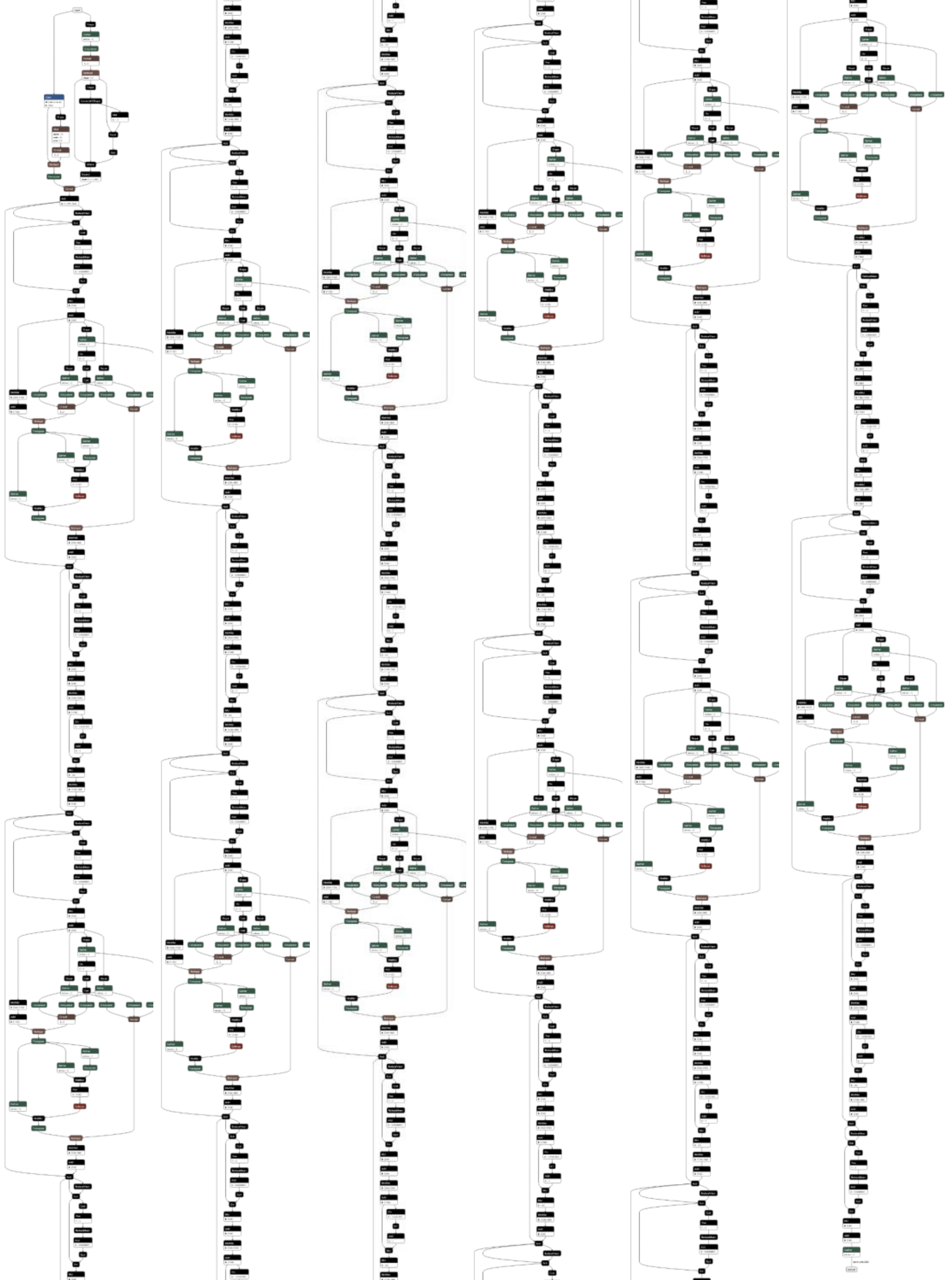
- We train only the student network
- From time to time, we copy the student to the teacher.

DINO

- Self-supervised Teacher-Student network pretrained to provide attention maps



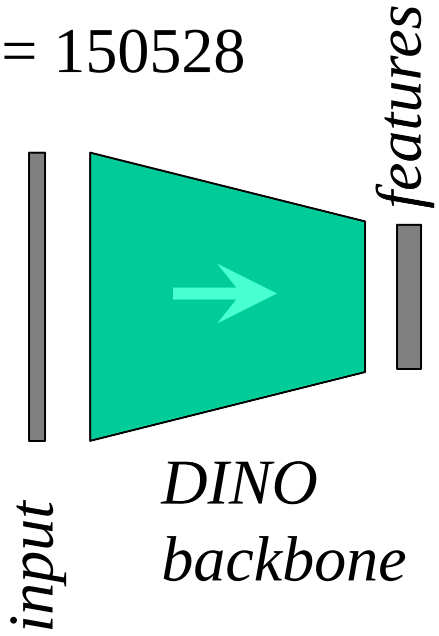
- Backbone: ViT turns images into mere 384 features



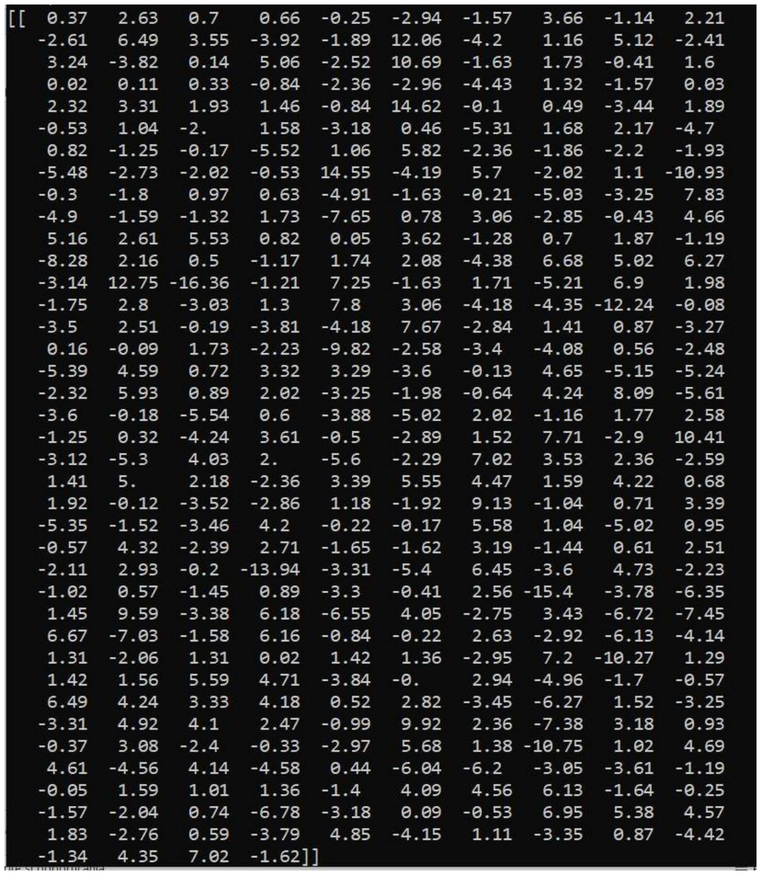
Our extractor

- It employs DINO’s medium size pre-trained ViT backbone

224x224x3 = 150528



1x384



```
[[  0.37   2.63   0.7    0.66  -0.25  -2.94  -1.57   3.66  -1.14   2.21
   -2.61   6.49   3.55  -3.92  -1.89  12.06  -4.2    1.16   5.12  -2.41
    3.24  -3.82   0.14   5.06  -2.52  10.69  -1.63   1.73  -0.41   1.6
    0.02   0.11   0.33  -0.84  -2.36  -2.96  -4.43   1.32  -1.57   0.03
    2.32   3.31   1.93   1.46  -0.84  14.62  -0.1    0.49  -3.44   1.89
   -0.53   1.04  -2.     1.58  -3.18   0.46  -5.31   1.68   2.17  -4.7
    0.82  -1.25  -0.17  -5.52   1.06   5.82  -2.36  -1.86  -2.2   -1.93
   -5.48  -2.73  -2.02  -0.53  14.55  -4.19   5.7   -2.02   1.1  -10.93
   -0.3   -1.8    0.97   0.63  -4.91  -1.63  -0.21  -5.03  -3.25   7.83
   -4.9   -1.59  -1.32   1.73  -7.65   0.78   3.06  -2.85  -0.43   4.66
    5.16   2.61   5.53   0.82   0.05   3.62  -1.28   0.7    1.87  -1.19
   -8.28   2.16   0.5   -1.17   1.74   2.08  -4.38   6.68   5.02   6.27
   -3.14  12.75 -16.36  -1.21   7.25  -1.63   1.71  -5.21   6.9    1.98
   -1.75   2.8   -3.03   1.3    7.8    3.06  -4.18  -4.35 -12.24  -0.08
   -3.5    2.51  -0.19  -3.81  -4.18   7.67  -2.84   1.41   0.87  -3.27
    0.16  -0.09   1.73  -2.23  -9.82  -2.58  -3.4   -4.08   0.56  -2.48
   -5.39   4.59   0.72   3.32   3.29  -3.6   -0.13   4.65  -5.15  -5.24
   -2.32   5.93   0.89   2.02  -3.25  -1.98  -0.64   4.24   8.09  -5.61
   -3.6   -0.18  -5.54   0.6   -3.88  -5.02   2.02  -1.16   1.77   2.58
   -1.25   0.32  -4.24   3.61  -0.5   -2.89   1.52   7.71  -2.9   10.41
   -3.12  -5.3    4.03   2.    -5.6   -2.29   7.02   3.53   2.36  -2.59
    1.41   5.     2.18  -2.36   3.39   5.55   4.47   1.59   4.22   0.68
    1.92  -0.12  -3.52  -2.86   1.18  -1.92   9.13  -1.04   0.71   3.39
   -5.35  -1.52  -3.46   4.2   -0.22  -0.17   5.58   1.04  -5.02   0.95
   -0.57   4.32  -2.39   2.71  -1.65  -1.62   3.19  -1.44   0.61   2.51
   -2.11   2.93  -0.2  -13.94  -3.31  -5.4    6.45  -3.6    4.73  -2.23
   -1.02   0.57  -1.45   0.89  -3.3   -0.41   2.56 -15.4   -3.78  -6.35
    1.45   9.59  -3.38   6.18  -6.55   4.05  -2.75   3.43  -6.72  -7.45
    6.67  -7.03  -1.58   6.16  -0.84  -0.22   2.63  -2.92  -6.13  -4.14
    1.31  -2.06   1.31   0.02   1.42   1.36  -2.95   7.2  -10.27   1.29
    1.42   1.56   5.59   4.71  -3.84  -0.     2.94  -4.96  -1.7   -0.57
    6.49   4.24   3.33   4.18   0.52   2.82  -3.45  -6.27   1.52  -3.25
   -3.31   4.92   4.1    2.47  -0.99   9.92   2.36  -7.38   3.18   0.93
   -0.37   3.08  -2.4   -0.33  -2.97   5.68   1.38 -10.75   1.02   4.69
    4.61  -4.56   4.14  -4.58   0.44  -6.04  -6.2   -3.05  -3.61  -1.19
   -0.05   1.59   1.01   1.36  -1.4    4.09   4.56   6.13  -1.64  -0.25
   -1.57  -2.04   0.74  -6.78  -3.18   0.09  -0.53   6.95   5.38   4.57
    1.83  -2.76   0.59  -3.79   4.85  -4.15   1.11  -3.35   0.87  -4.42
   -1.34   4.35   7.02  -1.62]]
```

Inference time on 4GB GPU: 0.05s

Our Generator

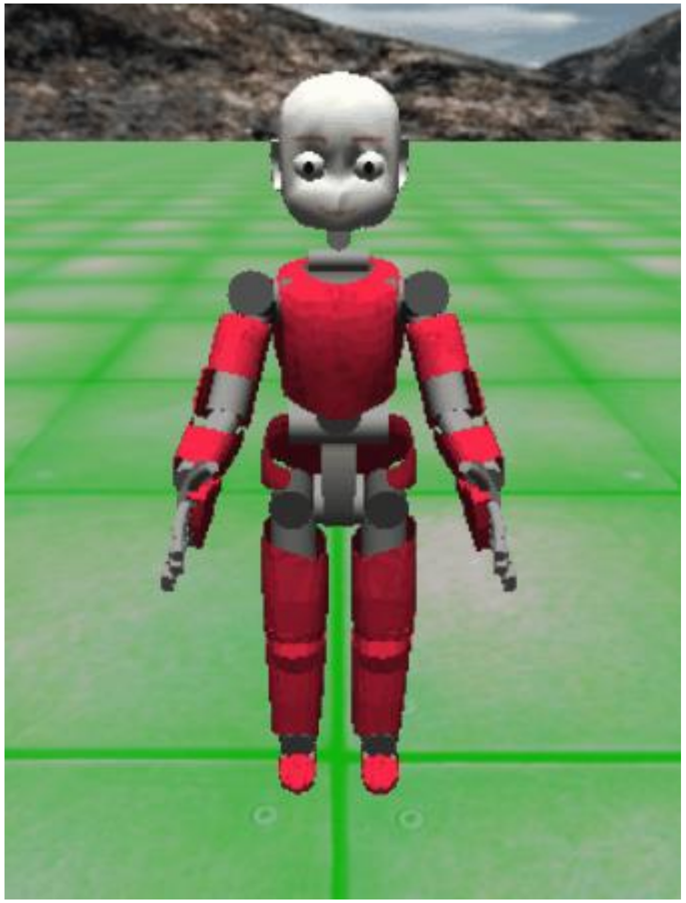
- It turns a feature vector (e.g., with dimension 2) into ten angles of iCub’s arm (5 for the right arm and 5 for the left one)
- Such a model is not available, so we need to train it from scratch
- We will prepare the dataset with reasonable iCub movements, train the autoencoder, and distillate its decoder part

Dataset

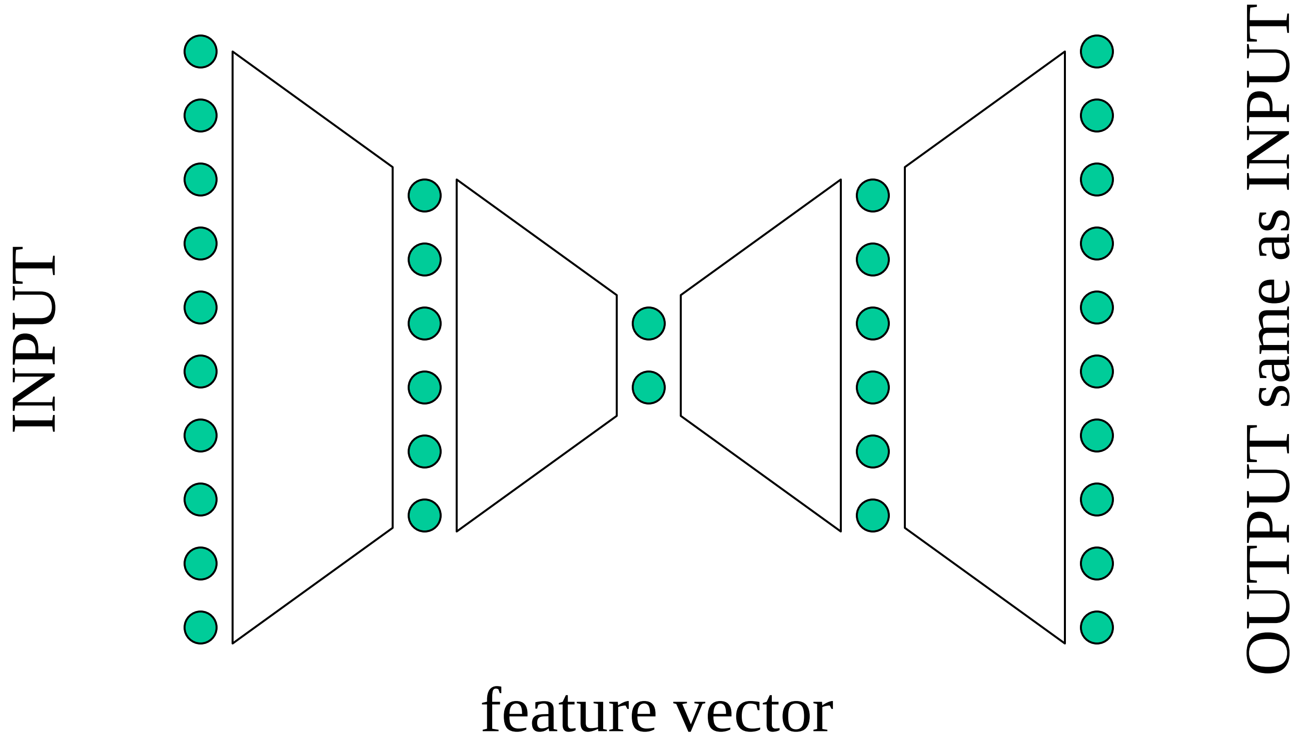
- Which joint angular positions are reasonable?
- Being a robot, to reveal them, we could move our arms randomly but in a comfortable way. However, since we are not, we rely on inverse kinematics.
- We employ known iCub’s Denavit-Hartenberg parameters to implement direct kinematics and extended FABRIK algorithm for implementation of the inverse one.

Dataset

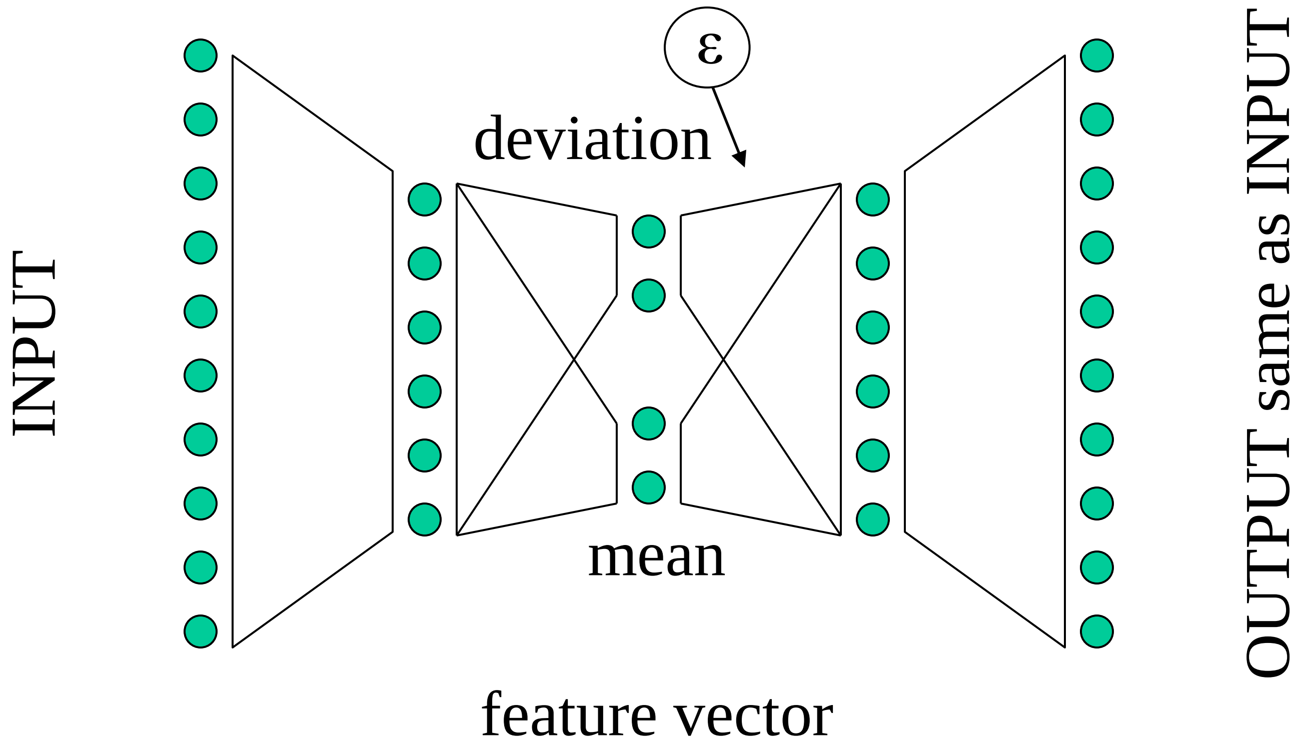
- Then, we explore all points in the robot’s vicinity and try to reach them by inverse kinematics. The found joint angles are concerned to be reasonable.
- Then, we combine kinematics data for one arm to build a dataset containing symmetric and asymmetric arm movements.



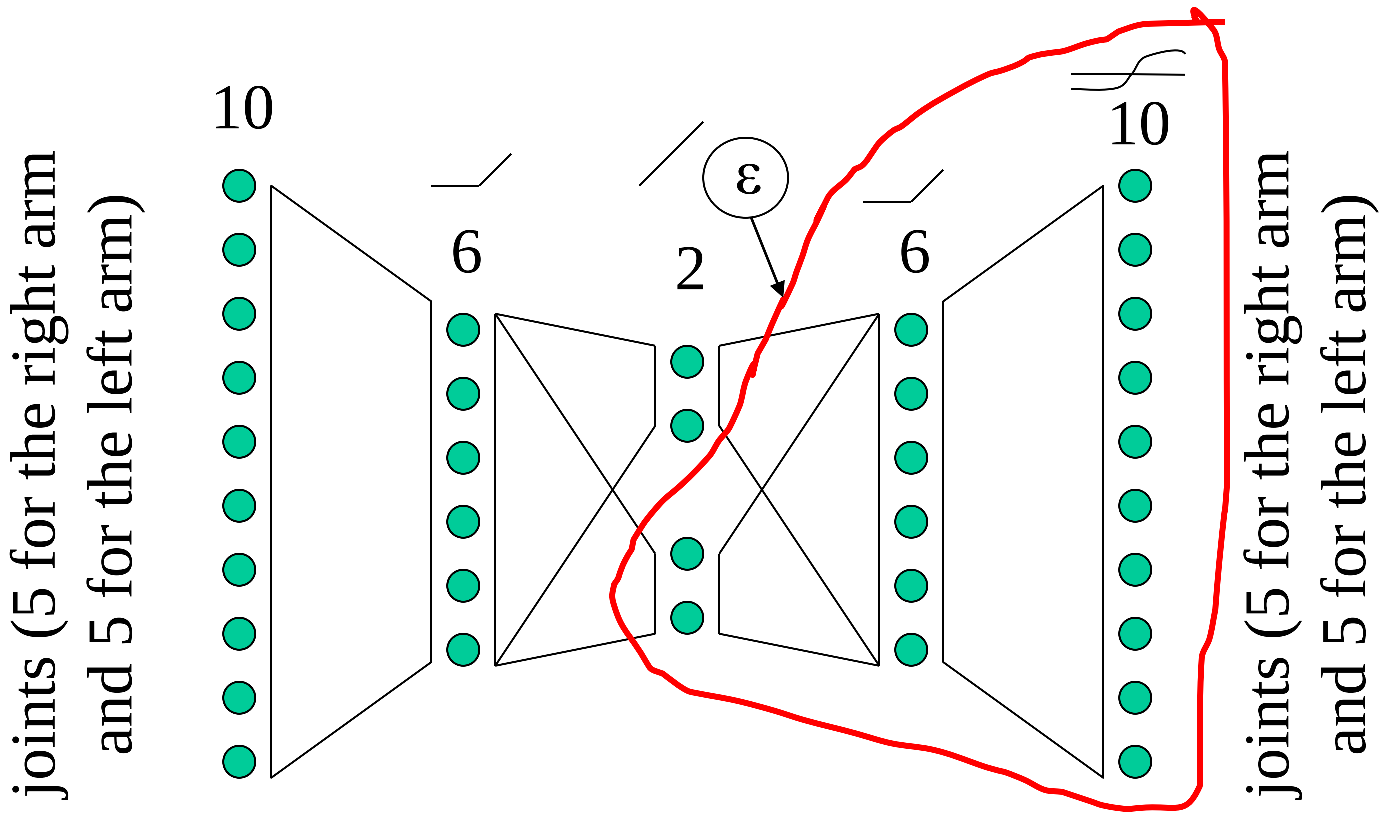
Deep Autoencoder



Variational Autoencoder

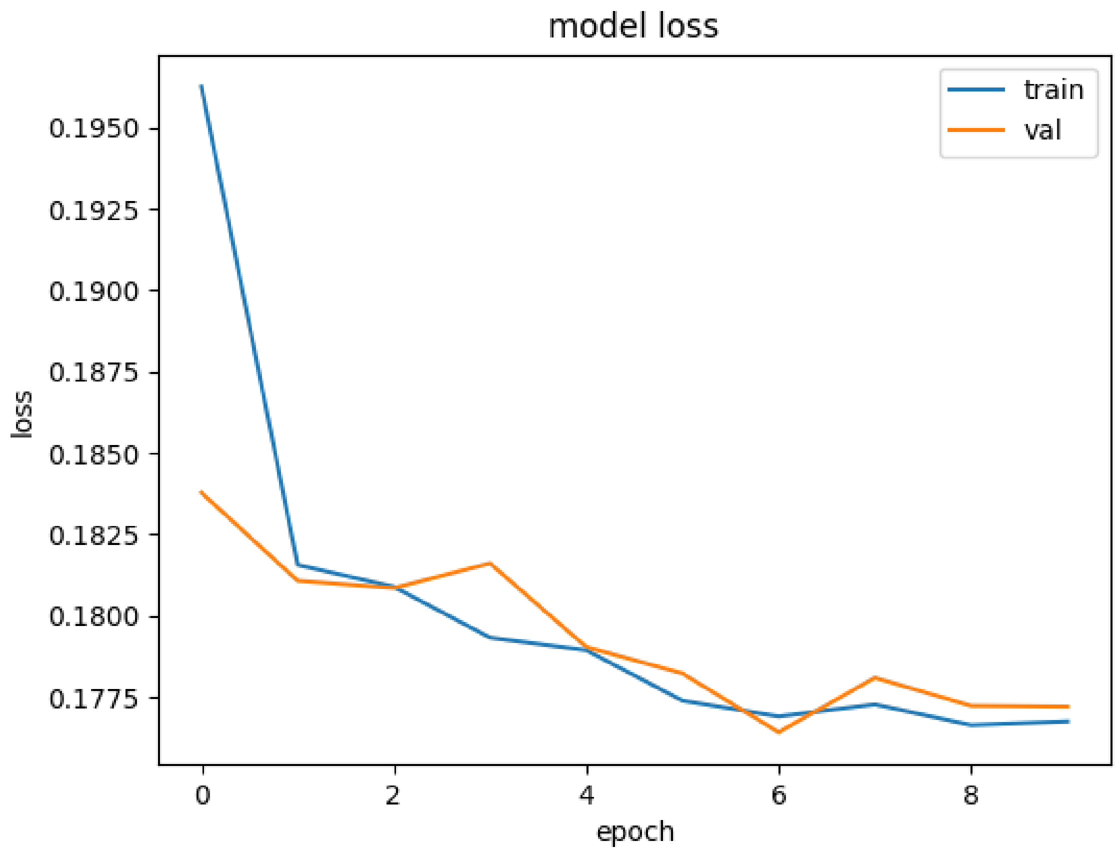


Our Generator = Decoder distillated from the Variational Autoencoder

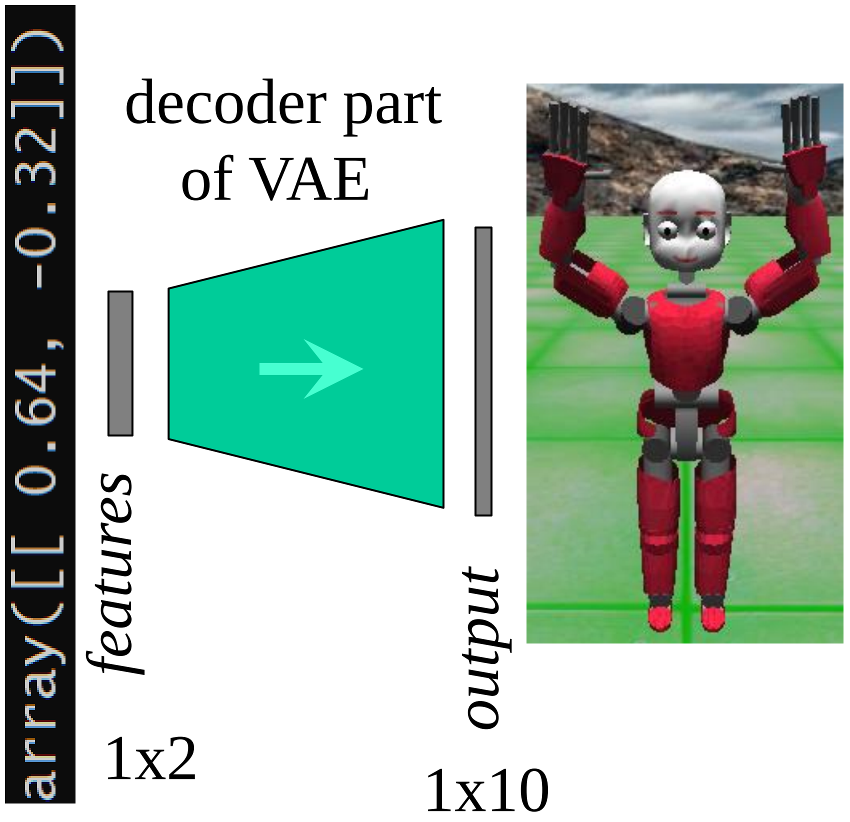

10
6
2
ε
6
10
joints (5 for the right arm and 5 for the left arm)
joints (5 for the right arm and 5 for the left arm)

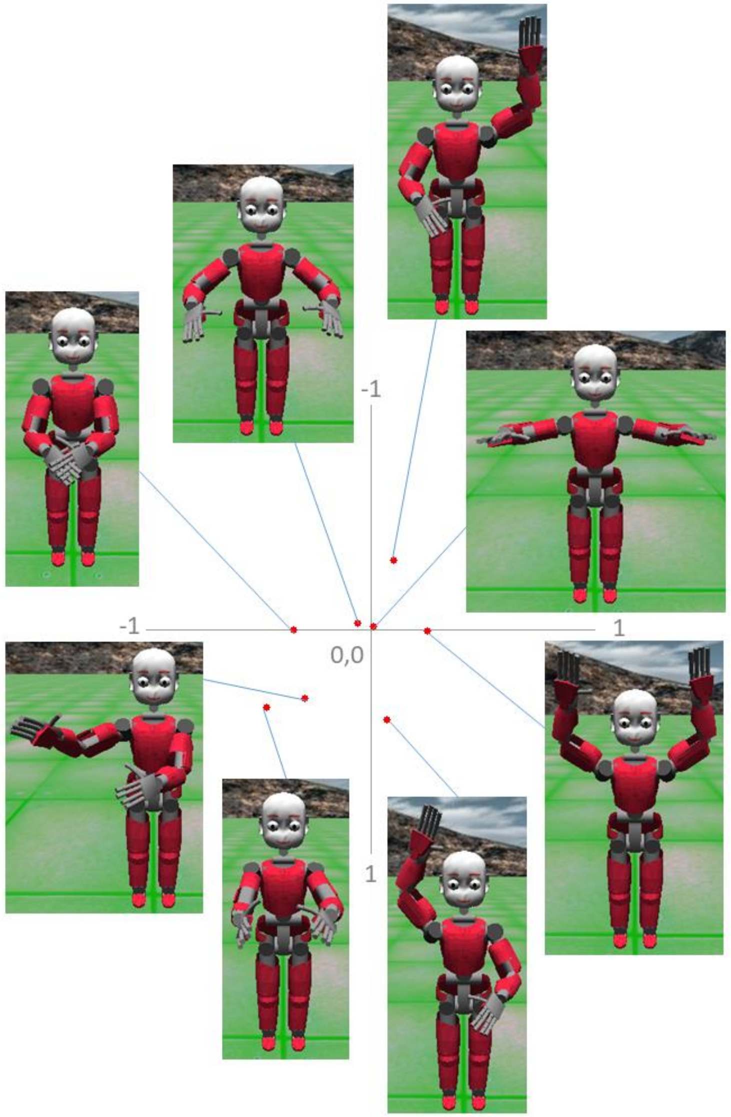
feature vector (x,y)
its dimension is 2

Training


model loss
train
val
loss
0.1950
0.1925
0.1900
0.1875
0.1850
0.1825
0.1800
0.1775
0
2
4
6
8
epoch

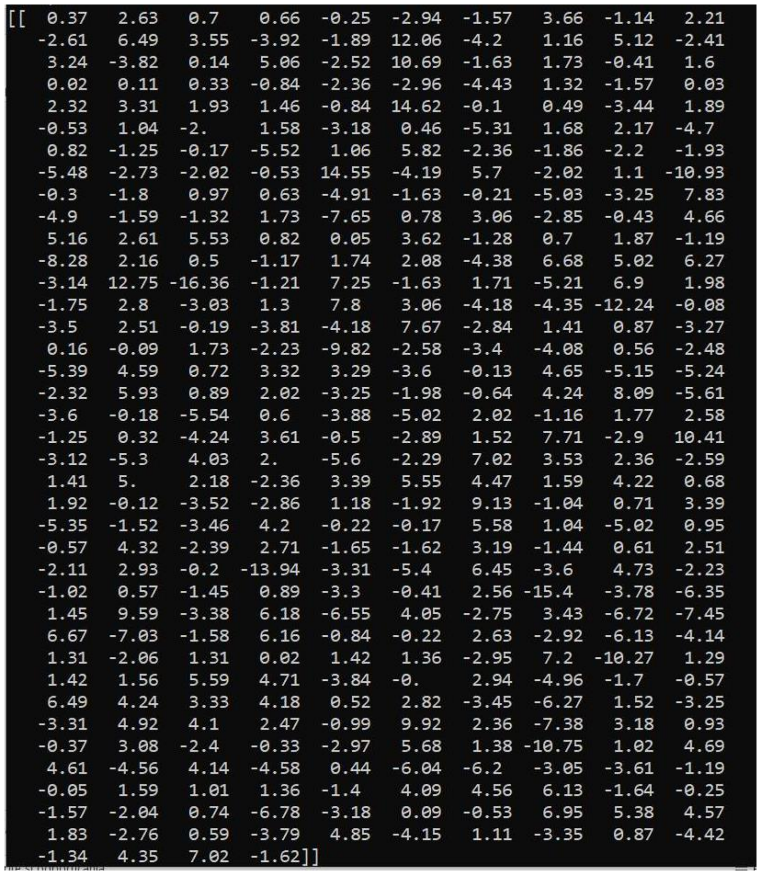
Our Generator





Associations via Attention

1x384



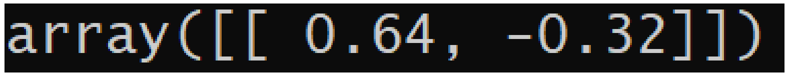
```
[[  0.37   2.63   0.7    0.66  -0.25  -2.94  -1.57   3.66  -1.14   2.21
   -2.61   6.49   3.55  -3.92  -1.89  12.06  -4.2    1.16   5.12  -2.41
    3.24  -3.82   0.14   5.06  -2.52  10.69  -1.63   1.73  -0.41   1.6
    0.02   0.11   0.33  -0.84  -2.36  -2.96  -4.43   1.32  -1.57   0.03
    2.32   3.31   1.93   1.46  -0.84  14.62  -0.1    0.49  -3.44   1.89
   -0.53   1.04  -2.     1.58  -3.18   0.46  -5.31   1.68   2.17  -4.7
    0.82  -1.25  -0.17  -5.52   1.06   5.82  -2.36  -1.86  -2.2   -1.93
   -5.48  -2.73  -2.02  -0.53  14.55  -4.19   5.7   -2.02   1.1  -10.93
   -0.3   -1.8    0.97   0.63  -4.91  -1.63  -0.21  -5.03  -3.25   7.83
   -4.9   -1.59  -1.32   1.73  -7.65   0.78   3.06  -2.85  -0.43   4.66
    5.16   2.61   5.53   0.82   0.05   3.62  -1.28   0.7    1.87  -1.19
   -8.28   2.16   0.5   -1.17   1.74   2.08  -4.38   6.68   5.02   6.27
   -3.14  12.75 -16.36  -1.21   7.25  -1.63   1.71  -5.21   6.9    1.98
   -1.75   2.8   -3.03   1.3    7.8    3.06  -4.18  -4.35 -12.24  -0.08
   -3.5    2.51  -0.19  -3.81  -4.18   7.67  -2.84   1.41   0.87  -3.27
    0.16  -0.09   1.73  -2.23  -9.82  -2.58  -3.4   -4.08   0.56  -2.48
   -5.39   4.59   0.72   3.32   3.29  -3.6   -0.13   4.65  -5.15  -5.24
   -2.32   5.93   0.89   2.02  -3.25  -1.98  -0.64   4.24   8.09  -5.61
   -3.6   -0.18  -5.54   0.6   -3.88  -5.02   2.02  -1.16   1.77   2.58
   -1.25   0.32  -4.24   3.61  -0.5   -2.89   1.52   7.71  -2.9   10.41
   -3.12  -5.3    4.03   2.    -5.6   -2.29   7.02   3.53   2.36  -2.59
    1.41   5.     2.18  -2.36   3.39   5.55   4.47   1.59   4.22   0.68
    1.92  -0.12  -3.52  -2.86   1.18  -1.92   9.13  -1.04   0.71   3.39
   -5.35  -1.52  -3.46   4.2   -0.22  -0.17   5.58   1.04  -5.02   0.95
   -0.57   4.32  -2.39   2.71  -1.65  -1.62   3.19  -1.44   0.61   2.51
   -2.11   2.93  -0.2  -13.94  -3.31  -5.4    6.45  -3.6    4.73  -2.23
   -1.02   0.57  -1.45   0.89  -3.3   -0.41   2.56 -15.4   -3.78  -6.35
    1.45   9.59  -3.38   6.18  -6.55   4.05  -2.75   3.43  -6.72  -7.45
    6.67  -7.03  -1.58   6.16  -0.84  -0.22   2.63  -2.92  -6.13  -4.14
    1.31  -2.06   1.31   0.02   1.42   1.36  -2.95   7.2  -10.27   1.29
    1.42   1.56   5.59   4.71  -3.84  -0.     2.94  -4.96  -1.7   -0.57
    6.49   4.24   3.33   4.18   0.52   2.82  -3.45  -6.27   1.52  -3.25
   -3.31   4.92   4.1    2.47  -0.99   9.92   2.36  -7.38   3.18   0.93
   -0.37   3.08  -2.4   -0.33  -2.97   5.68   1.38 -10.75   1.02   4.69
    4.61  -4.56   4.14  -4.58   0.44  -6.04  -6.2   -3.05  -3.61  -1.19
   -0.05   1.59   1.01   1.36  -1.4    4.09   4.56   6.13  -1.64  -0.25
   -1.57  -2.04   0.74  -6.78  -3.18   0.09  -0.53   6.95   5.38   4.57
    1.83  -2.76   0.59  -3.79   4.85  -4.15   1.11  -3.35   0.87  -4.42
   -1.34   4.35   7.02  -1.62]]
```

key

$\longleftrightarrow$

Somehow, we need to associate the image features with the action features.

1x2



```
array([[ 0.64, -0.32]])
```

value

We will build a list of the key-value pairs. Each pair represents one association. We respond with the corresponding value if an input equals one of the keys. Yet we must solve what to do if the input differs from all available keys.

Attention

We have *l* keys of dimension *n* and *l* values of dimension *m*

$$K = \begin{pmatrix} k_1 \\ k_2 \\ \dots \\ k_l \end{pmatrix} \qquad V = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_l \end{pmatrix}$$

Having a query *q* on input, we mix the query from keys *K:*

$$q = cK \qquad \text{where} \qquad c = softmax\left(\frac{qK^T}{d}\right)$$

and outputs an analogical mixture from the corresponding values *V*

$$A(q, K, V) = softmax\left(\frac{qK^T}{d}\right)V$$

for a suitable scale factor *d*

Softmax turns any vector *a* to a vector of probabilities *c*, which is reasonable mixture coefficients $c = softmax(a)$

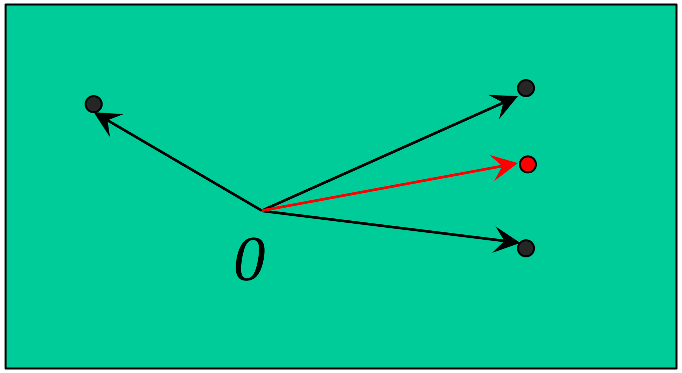
$$c_i \in \langle 0, 1 \rangle \; \sum c_i k_i = q, \; \sum c_i = 1, \text{ and } i = 1, 2, \ldots l$$

An excellent rough estimation of *a* is related to angle α between query and key: lower angle, higher coefficient.

$$cos\alpha_i = \frac{q \cdot k_i}{||q||\,||k_i||}$$

Yet the size of the key is relevant: a minor key, a smaller coefficient.

Thus $a_i \approx q.k_i$

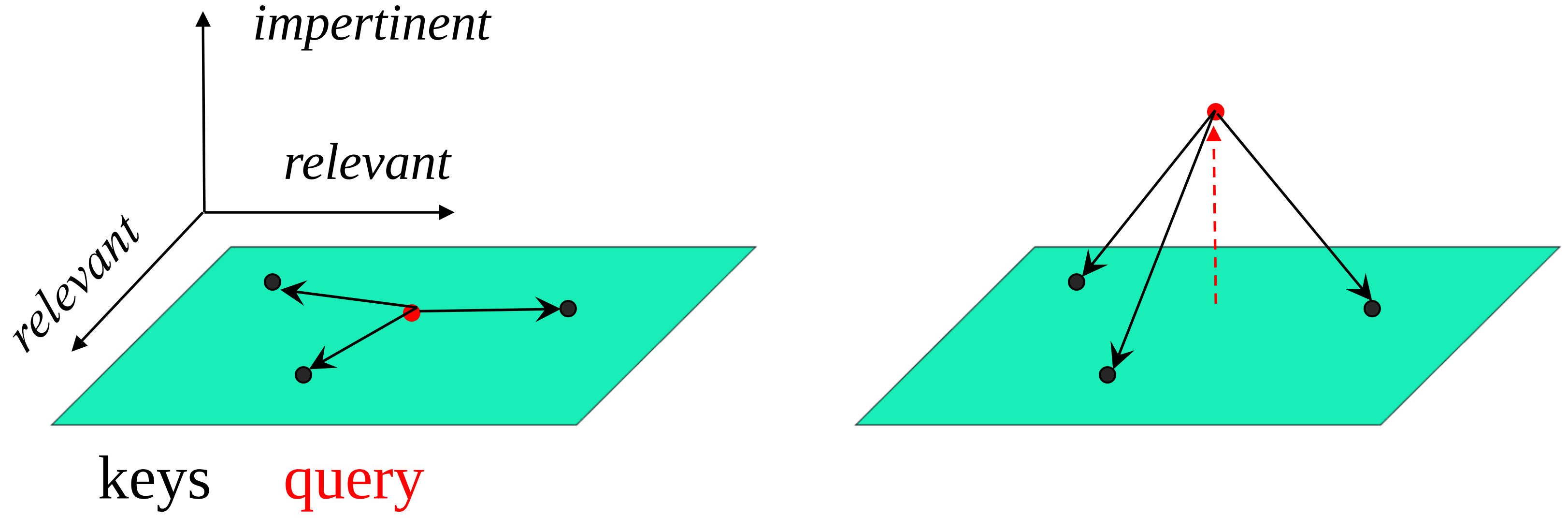


keys query

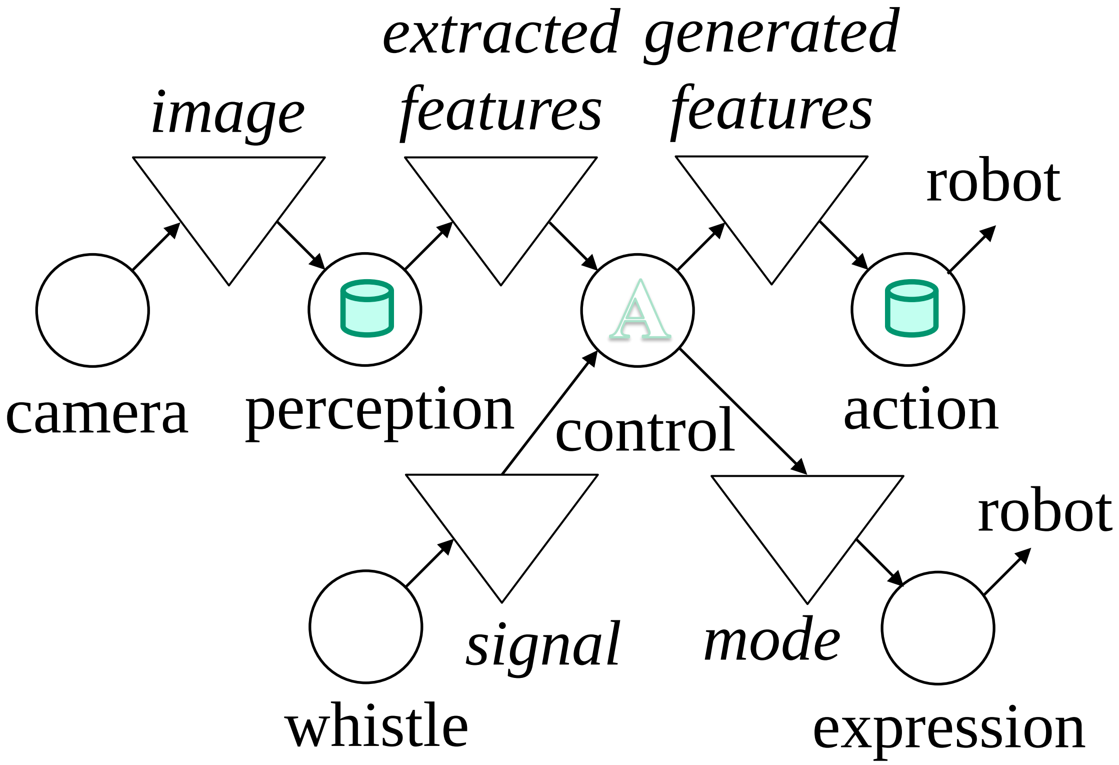
The scale factor *d* is a constant that enables us to scale how much we mix from similar keys and how much from different ones.

Why does Attention work?

Some features are relevant to the task, while others are not. If just the other changes, the mixture coefficients remain almost the same.

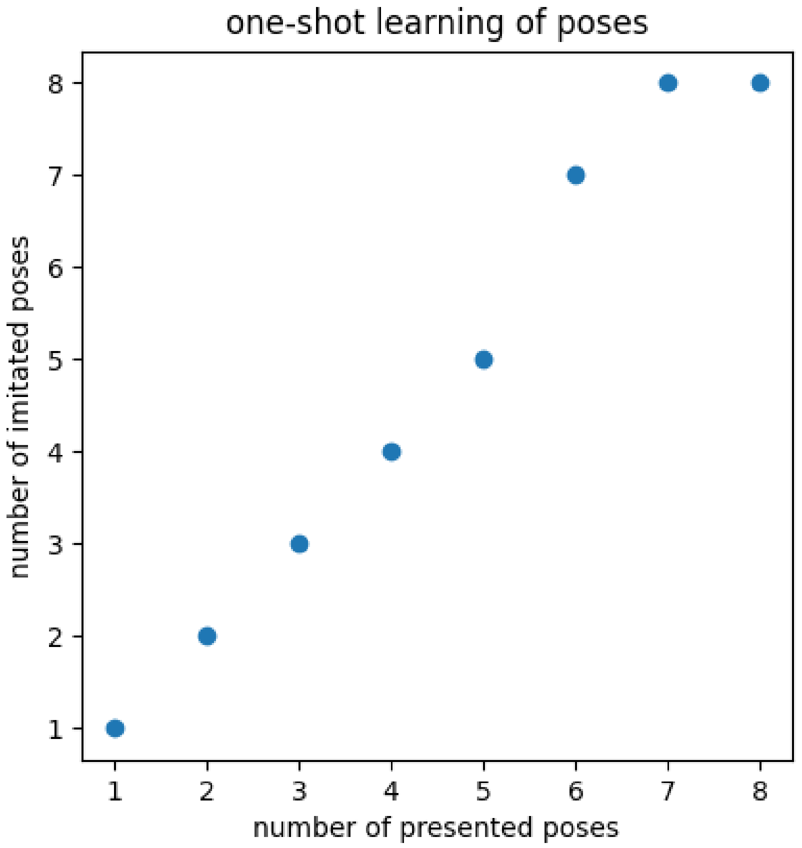


Real-time system



integration: blackboard architecture Agent-Space

Results


one-shot learning of poses
number of imitated poses
number of presented poses
1
2
3
4
5
6
7
8

Implementation details

github.com/andylucny/learningImitation

Generalized approach

The ACQUIRE-USE schema is not tied to the example with the imitation game. Further possible applications are:

- iCub at the mirror
- speech reproduction
- language acquisition

Conclusion

- We found that good-quality encoders and decoders (learned gradually) enable us to quickly start a qualitatively different learning process with a one-shot nature.
- We have designed a schema for the one-shot learning process originated in the attention mechanism.
- We have tested it on the imitation game.
- We have outlined further possible applications.

Thank you!

Towards one-shot learning via Attention

Andrej Lúčny

Department of Applied Informatics

Comenius University, Bratislava, Slovakia

lucny@fmph.uniba.sk

http://dai.fmph.uniba.sk/w/Andrej_Lucny

github.com/andylucny/learningImitation