Towards one-shot Learning via Attention

Andrej Lucny[1]

[1]Faculty of Mathematics, Physics and Informatics, Comenius University, Mlynska Dolina Bratislava 84248, Slovakia

Abstract

Though the deep neural networks enabled us to create systems that would be incredible ten years ago, still most of them learn gradually and offline. We introduce an approach to how to overcome this limitation. We have implemented it by the well-known attention mechanism that transforms one latent space into another using a list of key-value pairs defining the correspondence between points in the two spaces. We can express any point in the first space as a mixture of keys and map it to a point in the second space that is an analogical mixture of values. While encoders and decoders of these spaces we train only gradually, the keys and values of the transformation we can collect online so that we constantly improve the mapping quality, achieving the perfect mapping of the current situation immediately. We demonstrate our approach to one-shot learning on the simplified imitation game in the human-robot interaction, where we map the representations of the robot's body and the examinator's body seen by the robot.

Keywords

one-shot learning, attention, imitation game, deep learning models, self-supervision

1. Introduction

Artificial intelligence is a rapidly growing domain mainly due to deep learning technology. Nowadays, the ambition is to achieve general artificial intelligence, so we aim to develop a model that simultaneously processes image, text, and voice, incorporating multimodal knowledge. However, how we create the model is still close to developing any model tailored to a particular task; we need just much bigger datasets, data storage, and much more powerful hardware for training. Nevertheless, it is possible to train a model that can answer what the longest river in Africa is. But the model learns this fact gradually, processing it many times until it can answer correctly. On the other hand, we learn such facts in one shot. It is enough to tell us that the longest river in Africa is the Nile, and we can remember or forget it, but we avoid evolving answers from "blah blah blah" through "Egypt" to "the Nile." In the worst case, we produce errors like "the Mississippi."

Philosophers analyzing natural intelligence enlight that what today we call general intelligence is very far from the human one. For example, Daniel Dennet provided a famous analysis that recognized four kinds of minds [1]. The first kind (Darwinian) – although its behavior can be very effective – cannot adapt. Most of our artificial solutions correspond to this type that, in nature, we can observe mainly on insects. The second kind (Skinnerian) can adapt gradually, needing many shots to achieve a reasonable probability of intelligent behavior. The behavior of these creatures can change, e.g., by Pavlovian conditioning. The third kind (Popperian) can create mental models and, thanks to that, can suddenly adapt behavior, needing few or one shot. Finally, the fourth kind (Gregorian) manages language communication to transfer the content of the mental models from one individual to others. In this way, these creatures adapt even without a single shot.

The typical approach of deep learning is to train a Skinnerian system and interpret some observed behavior as Popperian or Gregorian capabilities. Such systems are Skinnerian at the structural level, but higher faculties emerge as side effects. A different approach could look for structural changes that correspond to, e.g., the acquisition of associations. Of course, some parts of an intelligent system can grow only gradually. But, after achieving a certain complexity level, one-shot learning should appear. We suppose that responsibility for this faculty is laying on processes different from those for gradual growth.

In this paper, we outline how it could work. At first, we need to develop structures that map some structured data (like the seen image or joint setup) into a (so-called latent) space in which any point codes a reasonable instance of the data. We can provide that by technologies such as autoencoders or contrastive learning that do not need annotations. We need a vast set of samples and a lot of time, but after some time, the mapping converges, and we can fix it. It is essential to mention that though the used examples correspond only to isolated points, any point in the latent space corresponds to an instance of the data. At that moment, we can start a different process that maps one latent space to another, e. g. perception to action. This process creates a list of associations between points in one latent space and the other. For the two mapped points, the new association works correctly and immediately. However, the more associations we collect,

ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia

lucny@fmph.uniba.sk (A. Lucny)

0000-0001-6042-7434 (A. Lucny)

the more accurate the mapping of the whole spaces we get. Anytime we can express any point as a mixture of the associated points from one space and map it to an analogical mix from their pairs in the other space. In the deep neural network, we can provide it by the Attention module. As a result, the system can learn the presented example in one shot and use the knowledge for approximation for situations never seen.

We test this one-shot learning process on a modern reimplementation of an imitation game with a robot, introduced in [2] [3]. We aim to learn the ability to imitate arm movements, for which we need associations between the robot's body and the body seen by the robot's camera. In the first phase, the robot invites us to imitate it. It generates various arm poses, and the human imitates them with its body in front of the robot's camera. In the second phase, the robot mimics the human by associations learned during the first phase.

We present details of our approach in the third chapter after discussing the related works in Chapter 2. Then we deal with its demonstration in Chapter 4. Finally, we discuss quality and the pros and cons.

2. Related Works

Looking at how to initiate one-shot learning, we prefer self-supervised gradual methods, i.e., gradient-based methods working with unlabelled data. Only these methods could correspond to how living creatures or robots in changing environments learn. We know several such ways and pay attention to autoencoders and metric learning. Their task is to provide us (by a gradual process) with extractors that map raw data into a latent space and generators that can turn any value from the latent space into raw data. Finally, the attention mechanism enables us to map one latent space to the other.

2.1. Autoencoders

Autoencoders [4] are predecessors of the early convolutional networks. They contain blocks of convolutional layers interleaved by dimension reduction in the first half and dimension expansion in the second half. Thus data like images with a typical dimension of hundreds of thousands are sequentially reduced to a feature vector with a size of hundreds or thousands and then expanded to the original extent. We train them from an unlabelled dataset to respond with an output equal to the input. If the training is successful, each part of the processing sequence contains the same information. As a result, the feature vectors have the same information as images from the dataset. Then we cut the part of the neural network after the feature vector and get the extractor (encoder). Or we remove the part before the feature vector and get the generator (decoder). Feature vectors of all instances of the raw data constitute a latent space with a dimension equal to the size of the feature vector.

The distribution of the feature vectors in the latent space is crucial [5]. We prefer to have similar data mapped to similar vectors. We can achieve a good performance mainly by the variational autoencoders [6]. They split the feature vector into two parts: one corresponds to average and the other to deviation. In this way, we push features to have the Gaussian distribution.

2.2. Metric learning

We can create feature extractors also without the necessity to develop both encoders and decoders. However, training only the encoder part, we lack the exact output we like to get for a given input. We do not know what feature vector we expect. We know only, e. g., the input category. Thus we cannot calculate the gradient from the difference between the actual and expected outputs. So, instead, we specify a metric that the mapping of all instances from our dataset should hold. Then we try our network with all inputs and identify the worst pairs of feature vectors that are close, have different categories, or are far, and have the same type. We want to move them in the latent space further or closer, and that direction provides us with the gradient for training the network. After many training cycles, the mapping holds the metric and becomes suitable [7].

Advanced methods, called contrastive learning, employ metrics based on the similarity or diversity of outputs of two copies of the same network. We feed them with two augmentations of the same instance from the dataset and require similar outcomes or with different samples expecting different results. We can train both networks [8] or only one of them [9]. In the second case, one copy is the teacher and the other student. During the training, we adjust the student network weights and occasionally copy them to the teacher.

In this way, we get feature extractors of better quality. Then we can train the corresponding decoders and use them as generators when we feed them with inputs different from the feature vectors of instances from the dataset.

2.3. Attention

The invention of the attention mechanism in natural language processing enabled us to take to regard the global content during processing sequences of words, e. g., for distinguishing the meaning of synonyms upon their content. Then it helped to avoid sequential processing and start the transformer revolution in the whole domain of deep learning [10]. But, for our purposes, it is enough to concern it from a mathematical point of view. The

attention mechanism works with a set of key-value pairs. Having a query q on input, we mix the query from keys K and outputs an analogical mixture from the corresponding values V, where:

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

All queries and keys are vectors of dimension n, so K is a matrix $l \times n$. Values and outputs are vectors of dimension m, so V is a matrix $l \times m$. At first, we find such $c_i \in \langle 0, 1 \rangle$ that $\sum c_i k_i = q$, $\sum c_i = 1$, and $i = 1, 2, ...l$. Since we like to mix the query more from keys similar to it and less from different keys, we can define the mixture roughly as dot product $c_i^{(0)} = qk_i = \|q\|\|k_i\| \cos \phi_i$, where ϕ_i is the angle between q and k_i. Since $\cos \phi_i$ is 1 for same, 0 for different, and -1 for opposite vectors, we can turn the products to high, middle, and small values from $\langle 0, 1 \rangle$ by $softmax(x)_i = \frac{e^{x_i}}{\sum_k e^{x_k}}$. Thus $c = softmax(\frac{c^{(0)}}{d}) = softmax(\frac{qK^T}{d})$, where d is a constant that enables us to scale how much we mix from similar keys and how much from different ones. Since the length of vectors is growing with $\sqrt{n}$ and dot product with n, it is popular to define $d = \sqrt{n}$. It would mean we include different and opposite keys, even if one key equals the query. For our purposes, we use a much smaller scale factor $d = \sqrt[5]{n}$, since we prefer to mix almost from a single key if the key is equal to the query. Having coefficients of the mixture c, we can mix values V to output $o = cV$. So, the complete response of the attention module to a single query q is:

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

Yet we mention that the typical use of the attention mechanism is the so-called self-attention, for which the queries, keys, and values are coming from the same input, and we aim to get the query compatibility to each key. However, this is not the case. So instead, we will use the mechanism for mapping two latent spaces; queries and keys are from one space, and values and outputs are from another.

3. Method

Let us assume we have a system with one feature extractor F and one generator G. The extractor represents perception and generator action. The extractor implements function $F : I \to L_F$, where I is a set of system inputs and L_F, is their latent space, i.e., turns the input data into feature vectors. The generator implements function $G : L_G \to O$, where O is a set of system outputs and L_G is their latent space, i.e., turns feature vectors into the output data.

We map the spaces L_F and L_G by the attention module A. We are gradually building matrices K and V containing keys and values. Each pair key-value corresponds to an association between two feature vectors representing stimulus and response at the system level. The system employs the generator to act with $a = G(v)$ upon a random value v. As a result, it receives a response r and extracts key $k = F(r)$. Then, concerning their quality, the system can include k, v into K, V. Thus, the mapping L_F and L_G become richer.

On the other hand, when the system receives an input p independent of the system activities, it turns it into query $q = F(p)$. If the query is equal to one of the keys, i. e. $q = k_i$, we aim to act with $G(v_i)$. However, it is much more probable that we cannot translate the query so directly. Therefore we operate with $G(v)$ where $v = A(q, K, V)$.

So, we can summarize the system operation into two processes (procedures $ACQUIRE$ and USE):

Algorithm Method of attention-based one-shot learning

F is extractor, G generator
A is attention, K keys, V values

procedure ACQUIRE(F,G,K,L)
  loop
    $v \leftarrow random()$
    $o \leftarrow G(v)$
    $output(o)$
    $r \leftarrow input()$
    $k \leftarrow F(r)$
    $K \leftarrow K \cup \{k\}$
    $V \leftarrow V \cup \{v\}$

procedure USE(F,G,K,L)
  loop
    $p \leftarrow input()$
    $q \leftarrow F(p)$
    $v \leftarrow A(q, K, V)$
    $o \leftarrow G(v)$
    $output(o)$

Of course, this schema is not generally applicable. However, if we can apply it, it grants one-shot learning. The system capability is still growing gradually but in steps, without transient states. Each demonstration invokes immediate faculty to act accordingly under situations close to the seen example. The system operates somehow also upon unseen conditions, and the quality of these actions grows with the number of key-value pairs.

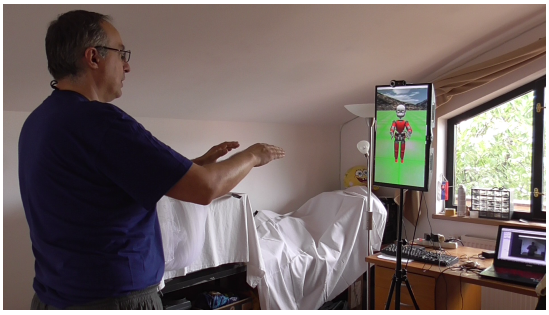

Figure 1: The imitation game.

4. Demonstration and Evaluation

To demonstrate the method, we deal with the imitation game between a human and a humanoid robot. Though it is possible to implement this task with classic computer vision [3] and deep learning [11], for our purposes, it has the meaning to re-implement it in a novel way. In this game, a humanoid robot with a body similar to humans invites a human to mimic his movements. If the human accepts the invitation and imitates the robot, the robot learns how to mimic the human. As a result, the robot can mimic the movements of humans (Figure 1).

For implementation, we need a humanoid robot; we employ iCubSim, the simulator of the iCub robot [12], equipped with an external camera. We control it from Python via pyicubsim, ONNX-runtime and OpenCV [13] libraries. Further, we need a feature extractor that turns images seen by the robot to feature vectors. For our purpose, it is not necessary to train it; we can get it from a pre-trained model for computer vision (obtained by a self-supervised method described in Chapter 2.2). However, we need to invest more effort in generating robot movements since neither pre-trained models nor datasets are available to us for the chosen robot. First, we create the dataset as a set of the robot joint positions while moving its arm to random points in the robot's vicinity. We avoid abnormal setups of robot joints by their calculation by inverse kinematics. Then we train (using Keras [14]) the variational autoencoder (see Chapter 2.1) and get the generator model as its part. Having the extractor and generator, we can define the overall model controlling the robot as their integration by the attention module (see Chapter 2.3). Since the system operates in real-time and calls models, the integration employs a blackboard architecture [15] that helps us to combine slower and faster processes. Finally, we test the system.

4.1. Extractor

We turn images into features by the pre-trained model DINO [9]. In detail, we use dino_deits8.onnx – the middle-sized version of the latter vision-transformer backbone, distributed in the ONNX format. Though it turns color images with resolution 224x224 into feature vectors of mere 384 numbers, its quality is incredible, demonstrated by several successful applications, including pose detection. Thus we are sure that the vector also contains information representing the person's pose on the image. But, of course, the pose is in a raw form: we use the backbone only, while the applications mentioned above add further processing layers. The model is relatively large, but its middle-sized version can fit into the 4GB GPU. Moreover, its inference takes 0.05s on an ordinary gaming notebook; thus, it is very suitable for building real-time applications.

4.2. Generator

The iCub robot arm contains five significant degrees of freedom, two in the shoulder and three in the elbow joints. All together pose of the left and right arms is coded by ten angles.

Aiming to create a decoder generating accurate poses, we first need to get their dataset. We cannot get it by random generation of joints because it would also contain unnatural poses. So we need to define what it means natural here. To do that, we have decided to concern natural poses generated by the inverse kinematics. We have asked the robot to move its arm to all possible coordinates in its vicinity, and if it succeeded in reaching the point, we have added the current joint setup into the dataset. It was not an easy job because inverse kinematics for iCub was not available to us. Therefore we have started from the known Denavit-Hartenberg parameters of the robot, and – using on-the-shelf direct kinematics [16] – we have implemented the FABRIK algorithm [17] adjusted for Denavit-Hartenberg notation and extended by constraints [18]. It is a slow but fully operational solution that not only defines the natural poses of the robot but speeds up the creation of the dataset. We speed up the process because we can reliably calculate all data on the kinematics model, and we do not need to try them on the robot. Also, as we see later, it is profitable that our dataset can contain the Euler coordinates corresponding to the recorded joint setups. We do not use them for model creation, but they are helpful for model visualization. In this way, we have collected all possible poses (Figure 2), 23470 for each arm. Then we randomly selected 60000 examples concerning the equal probability that the robot uses the left arm, the right arm, both arms symmetrically, and both arms in different poses.

In the second phase, we used Keras to train the varia-

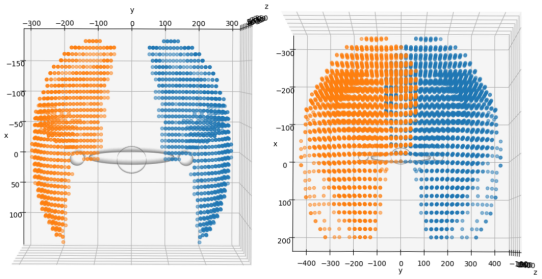

Figure 2: The iCub's kinematics (on the left: coordinates reachable by the elbow, on the right: coordinates reachable by the wrist).

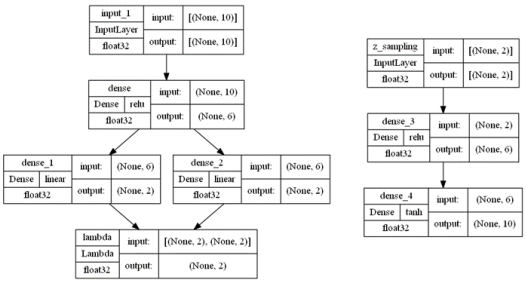

Figure 3: The architecture of the iCub's actions autoencoder (on the left: encoder, on the right: decoder).

tional autoencoder of the selected joint setups. Since the space of iCub's arm action is not ample, we have used just ten input, six intermediate, two feature, six intermediate, and ten output neurons. Of course, we double the internal structures because the features are the sum of the average and random multiple of standard deviation (Figure 3). We have used ReLU and tanh activations since we turned joint angles from $-180°$ to $180°$ into code from -1 to 1. Before training, we shuffled the dataset and split it into 50000 training and 10000 testing examples. The training required ten epochs with batch size 32 and took mere 92s (Figure 4). Finally, we have distilled the decoder part of the trained architecture and saved it as our generator. Yet we have converted the generator model from the .h5 format to the .pb format that we can open in the OpenCV library.

This model can turn any pair of numbers from -1 to 1 into a proper joint setup on the robot (Figure 5). But, further, we need to check that the space of generated actions is well-organized. In other words, we need to check that a fluent change of the feature vector causes only a fluent shift in the joint setup. Since the feature vector has only two numbers, we can easily visualize its quality in six pictures depicting the x,y, and z coordinates

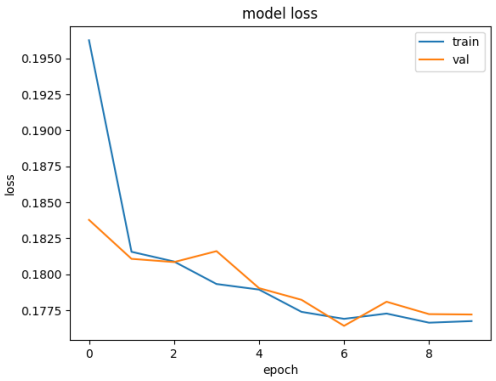

Figure 4: Training the variational autoencoder of iCub's arms movement.

of the right and left arms. We put a point to the picture for each example from the testing set. Its color corresponds to the value of the coordinate. Then fluent color gradient means that the space is well-organized (Figure 6).

4.3. Integration

Now we can integrate the extractor and generator models into one system. Since such a system needs to combine fast data sources like a camera with slower models and languid robot movement, the integration employs a blackboard architecture. Concretely, we use our solution named Agent-Space architecture [15] to split the system into a set of agents communicating via blackboard and let the overall control emerge from the individual behaviors of the agents.

Our system contains the following agents (Figure 7):

- The camera agent grabs images from the camera and writes them onto the backboard, where other agents can read image samples according to their processing capacity. (This way, we avoid delays and overloadings appearing if we put grabbing and processing images into the same loop.)
- The perception agent reads the grabbed image from the blackboard, turns it into a blob, feeds the extractor model, and writes the provided feature vector to the blackboard.
- The control agent operates in two modes: ACQUIRE and USE. In the first mode, it collects lists of keys and values corresponding to the feature vectors of the extractor and generator in the following way. First, it randomly generates a feature vector for the generator, writes it to the blackboard, waits, and reads the feature vector provided by the extractor. Then it adds them to the

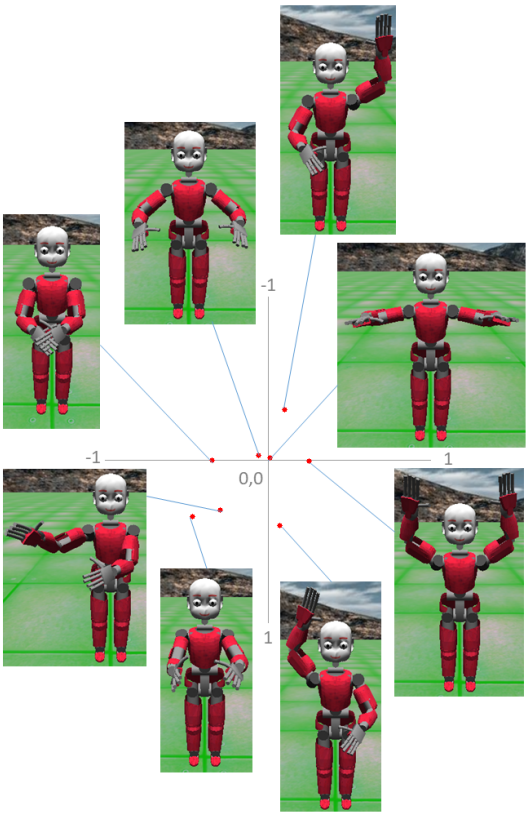

Figure 5: Examples of iCub's actions generated from the feature vectors.

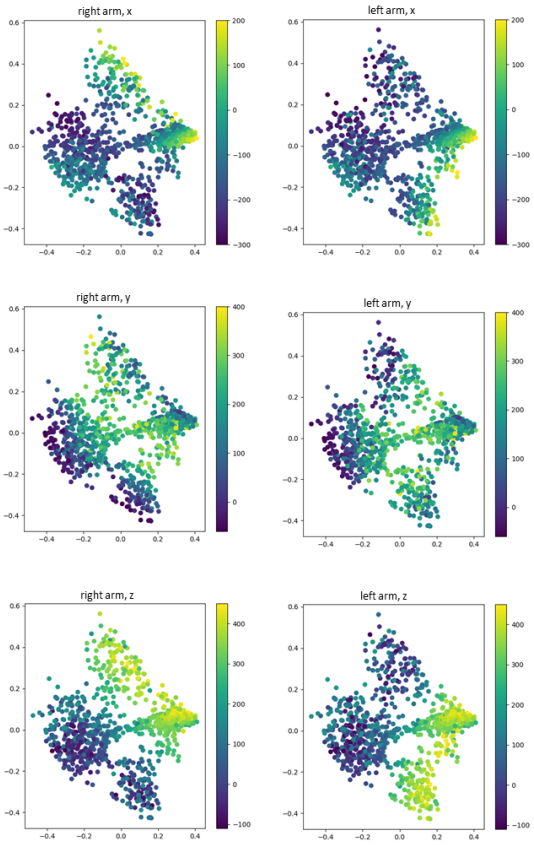

Figure 6: The x, y, and z coordinates of the right and the left iCub's arm for the testing set. Each point represents one sample, and its color is the value of the coordinate.

lists. In the second mode, the agent reads the feature vector provided by the extractor and writes the feature vector for the generator calculated by the attention module from the lists of keys and values.

- The action agent reads the feature vector for the generator and controls the iCub robot by the commands of the YARP protocol encapsulated by the pyicubsim library.
- For simplification, at the current stage of development, we specify the accurate time for waiting in the ACQUIRE mode by the examinator. Since his hands are busy imitating the robot's pose, we manage this signaling by whistling. We implemented this input by the pitch agent. It processes sound by the Fourier transform and looks for the high frequencies.

4.4. Testing

We have developed the real-time system incrementally, working with the off-line version, which quality we can investigate more easily. In this phase, we have selected a bunch of ten examinator's poses and created a few images – under varying conditions – for each pose. Then we taught the system, and after each sample, we tested the system's capability to imitate all poses. The number of operational poses indicated whether the system could forget a learned pose. We found that it did forget neither one. Even the system learned one pose implicitly, compounding the correct response from two other already presented poses (Figure 8).

So far, we have not evaluated the real-time version in another way than by the examinator's opinion. In the future, we plan to employ pose detectors for this purpose.

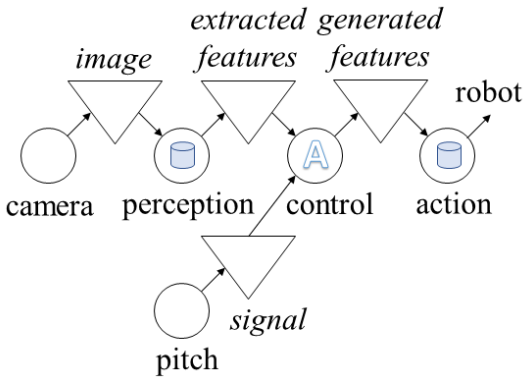

Figure 7: Schema of the integrated system for the imitation game. Circles represent agents, triangles the blackboard, cylinders models, and the letter A the attention module.

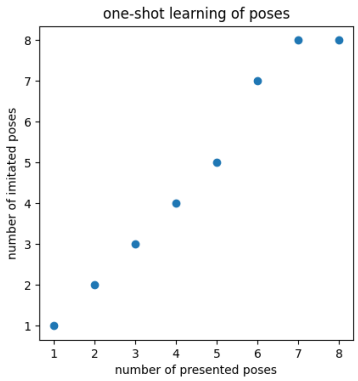

Figure 8: One-shot learning of selected arms poses.

5. Conclusion

In this paper, we introduced a kind of one-shot learning. Its key component is the attention module. We have used this existing component of deep neural networks for a new task: mapping two latent spaces. First, however, we had to adjust one of its parameters: the scale factor.

We demonstrated our approach to one-shot learning on imitation between human and humanoid robots. We built our demo from modules developed in a self-supervision way. Thus we avoided using datasets containing particular poses of the person on images and the robot's body. Instead, our robot has learned them by interacting with the examinator in a one-shot learning way.

For imitation, the robot needs to get the model of the body seen as analogical to the model of its own body. In humans, it is not clear where this ability originates. But our approach enlights that the imitation game not only solicits this ability but can also help it to emerge. Here imitation is an ability of a society [19], and one of its members learns it from other (a child from its parent or a robot from its user). Remarkably, this transfer could rely on the attention module, an essential building block for natural language processing. We could look at language as a kind of imitation related to the movement of vocal cords. Its nature is similar to hand movement. On the other hand, the presented one-shot learning mechanism could play a role in the early evolution of signal-based language.

Our approach also has weaknesses. The major one is that if we use an encoder that stems from general data, the mapping could be relevant only for specific conditions. For example, associations learned in the presented imitation game could be fooled by more persons in front of the camera. On the other hand, the quality of today's self-supervised models does not allow us to cheat the system by e. g. the different colors of the wall or the different dress of the seen person. We could decrease this problem by training the encoder from more specific data under specific conditions. However, it isn't easy to imagine that we could manage it in a self-supervision way.

Finally, our method is more general than an imitation game. For example, processing vision, the robot cannot see itself; therefore, it needs the help of an examinator. However, it could apply the same method for seeing itself in the mirror. Or it could similarly process voice. Having a speech generator corresponding to the physical capabilities of vocal cords, lips, and tongue and a voice listener analogical to the ear, it could start to produce random voices and learn the mapping between the listener perception and the generator action. As a result, it can reproduce the listened speech when another source makes it.

References

[1] D. C. Dennett, Kinds of minds: towards an understanding of consciousness, Weidenfeld & Nicolson, London, 1996.

[2] J. P. Bandera, J. A. Rodriguez, L. Molina-Tanco, A. Bandera, A survey of vision-based architectures for robot learning by imitation, International Journal of Humanoid Robotics 9 (2012). doi:10.1142/S0219843612500065, world Scientific Publishing Company.

[3] S. Boucenna, S. Anzalone, E. Tilmont, D. Cohen, M. Chetouani, Learning of social signatures through imitation game between a robot and a human partner, IEEE Transactions on Au-

tonomous Mental Development 6 (2014) 213–225. doi:10.1109/TAMD.2014.2319861.
[4] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (2006). doi:10.1126/science.1127647.
[5] Brownlee, Deep Learning for Computer Vision, 1.4 ed., machinelearningmastery.com, 2019.
[6] D. P. Kingma, M. Welling, An introduction to variational autoencoders, Foundations and Trends in Machine Learning 12 (2019) 307–392.
[7] D. King, High quality face recognition with deep metric learning, 2017. URL: http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html.
[8] T. Chen, S. Kornblith, M. Norouzi, G. Hinton, A simple framework for contrastive learning of visual representations, in: Proceedings of the 37th International Conference on Machine Learning, number 149 in ICML, 2020, pp. 1597–1607.
[9] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, A. Joulin, Emerging properties in self-supervised vision transformers, in: Proceedings of the International Conference on Computer Vision, ICCV, 2021.
[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, I. Polosukhin, Attention is all you need, in: 31st International Conference on Neural Information Processing Systems, ACM, Long Beach, 2017.
[11] M. Petrovich, M. J. Black, G. Varol, Action-conditioned 3D human motion synthesis with transformer VAE, in: International Conference on Computer Vision, ICCV, 2021.
[12] D. Vernon, G. Metta, G. Sandini, The icub cognitive architecture: Interactive development in a humanoid robot, in: 2007 IEEE 6th International Conference on Development and Learning, 2007, pp. 122–127. doi:10.1109/DEVLRN.2007.4354038.
[13] G. Bradski, The opencv library, Dr. Dobb's Journal of Software Tools (2000).
[14] F. Chollet, Deep Learning with Python, Manning Publications Co., Greenwich, CT, USA, 2017.
[15] A. Lucny, Building complex systems with agent-space architecture, Computers and Informatics 23 (2004) 1–36.
[16] L. Natale, C. Bartolozzi, F. Nori, G. Sandini, G. Metta, Humanoid Robotics, Springer, Dordrecht, 2017. doi:10.1007/978-94-007-6046-2.
[17] A. Aristidou, J. Lasenby, Fabrik: A fast, iterative solver for the inverse kinematics problem, Graphical Models 73 (2011) 243–260.
[18] R. A. Tenneti, A. Sarkar, Implementation of modified fabrik for robot manipulators, in: Proceedings of the Advances in Robotics 2019, 2019, pp. 1–6. doi:10.1145/3352593.3352605.
[19] A. Aristidou, J. Lasenby, Embodied gesture processing: Motor-based integration of perception and action in social artificial agents, Cognitive Computing 3 (2011) 419–435. doi:10.1007/s12559-010-9082-z.

6. Online Resources

We share codes of this project at GitHub