An error-based addressing architecture for dynamic model learning

Nicolas Bach*, Andrew Melnik*, Federico Rosetto, and Helge Ritter

CITEC, Bielefeld University, Bielefeld, Germany
nbach@techfak.uni-bielefeld.de
andrew.melnik.papers@gmail.com

Abstract. We present a distributed supervised learning architecture, which can generate trajectory data conditioned by control commands and learned from demonstrations. The architecture consists of an ensemble of neural networks (NNs) which learns the dynamic model and a separate addressing NN that decides from which NN to draw a prediction. We introduce an error-based method for automatic assignment of data subsets to the ensemble NNs for training using the loss profile of the ensemble. Our code is publicly available[1].

1 Introduction

In this work, we present a data-driven learning architecture that can learn a dynamic model of an agent in an environment while staying responsive to control commands. The proposed model consists of two parts, an ensemble of neural networks and an addressing neural network. The ensemble consists of neural networks (NNs) that learn situational models [1], i.e. each such model is tuned for only a subset of the training data. This is akin to a divide-and-conquer-strategy and is especially useful for problems with repeating cycles of state trajectories, where the ensemble NNs can specialize on different phase ranges, for example, a cyclic motion. Our approach is motivated by the phase-functioned neural networks (PFNN) approach [2] for developing a real-time controller for a virtual agent. This approach can generate accurate kinematic trajectories for locomotion learned from real-world motion capture data. The PFNN architecture generates weights of a regression network using a numerical phase value as input. This phase value is a time-resolving variable which labels each time step and helps to handle ambiguous data in the locomotion cycle. The required phase labeling of the training data is done in a semi-automatic and therefore time-consuming preprocessing step [2]. In contrast to this, our method implements an automated, error-based clustering mechanism of training data points, using the loss profiles of the ensemble NN across the training data points and, therefore, does neither require a handcrafted labeling preprocessing step nor a phase value.

[1] Code: https://github.com/NicoBach/distributed-dynamics-model
* Equal contribution.

Deep ensembles have been empirically shown to be a promising approach for improving accuracy, and out-of-distribution robustness of deep learning models [3]. One possible explanation is that deep ensembles tend to explore diverse modes in function space.

The problem of deriving a controller from motion capture data, similarly to PFNN, is also addressed by other approaches. Quaternet [4] uses a quaternion-based recurrent model to generate sequences of human poses. Others use deep reinforced learning (DRL) methods to produce motion control over a 3D virtual skeleton in kinematic or physical environments, [5], [6]. [7] propose an architecture based on hierarchical reinforcement learning to develop a model capable of locomotion. Modularization of end-to-end learning is another opportunity for improvement [8]. The MOSAIC architecture [9] is a modular architecture for motor learning and control based on multiple pairs of forward (predictor) and inverse (controller) models. The architecture simultaneously learns the multiple inverse models necessary for control as well as how to select the set of inverse models appropriate for a given environment. Similarly, [10] implement a biologically inspired, modular architecture for controlling a six-legged robot.

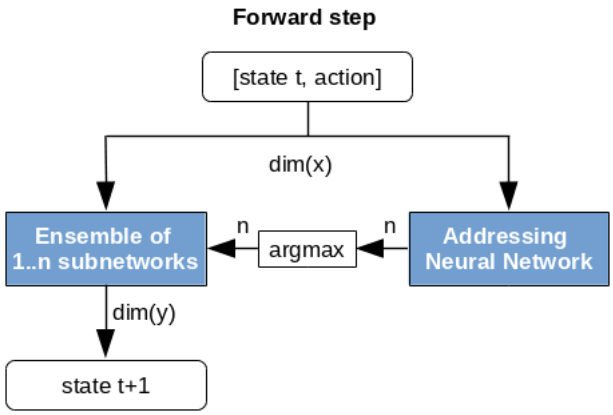


Fig. 1: The architecture consists of an ensemble of n fully connected networks and an addressing network, which chooses the network from the ensemble to predict the output y given the input x.

2 Methods

The structure of the architecture (Fig. 1) can be divided into two parts - the addressing NN and an ensemble of n networks. The task of the addressing NN is to choose a network from the ensemble. The addressing network is a fully connected network with the same input x as each subnetwork in the ensemble. The

output dimension of the addressing network is equal to the number of ensemble subnetworks n. The $argmax$ of the addressing NN output identifies the subnetwork from the ensemble to predict y from the *state* x. The input dimensions of all ensemble subnetworks are equal to $dim(x)$, and their task is to predict the next *state* y. With this input-dependent addressing, the architecture performs a regression step to predict the next state given the current state and action. The number of ensemble subnetworks is a hyperparameter and is chosen depending on the task. Thus, a forward step of the architecture consists of two steps, first, prediction by the addressing NN, second, prediction by the chosen subnetwork from the ensemble.

2.1 Clustering of training data

In this section, we describe the automatic clustering algorithm (algorithm 1; Fig. 3) for the training data, i.e. how the training set can be partitioned into n approximately equally-sized (in terms of the number of data points) bins for the n ensemble networks. Each subnetwork will be trained only on data points from its bin. An equal number of samples in bins ensures good competitiveness of subnetwork specialization. Not providing equally sized bins for the ensemble networks will preclude specialization if at one point an ensemble network becomes strictly better than all the others and then wins all training data for itself.

Algorithm 1 Clustering of training data using the loss profile space of the ensemble

Input: Training data points $\mathbf{x} \in X$, target $\mathbf{y} \in Y$, ensemble of n models

for $\mathbf{x}$ in X **do**
  Perform forward-step with every ensemble model on $\mathbf{x}$
  Compute losses between outputs of $\mathbf{x}$ for every model and target $\mathbf{y}$
  Form loss profile L of the data point
end for
Compute norms $||\mathbf{l}||^2$ for $\mathbf{l} \in L$
Sort loss profiles in L regarding norms from highest to lowest ($Ls = \text{sort}(L)$).
while len(Ls) < 0 **do**
  $\mathbf{x} = Ls$.pop()
  $processing = True$
  while $processing$ **do**
    if len($bins[argmin(\mathbf{x})]$) $<$ len(X) / len($bins$) **then**
      $bins[argmin(\mathbf{x})]$.append(x)
      $processing = False$
    else
      $\mathbf{x}[argmin(\mathbf{x})] = inf$
    end if
  end while
end while

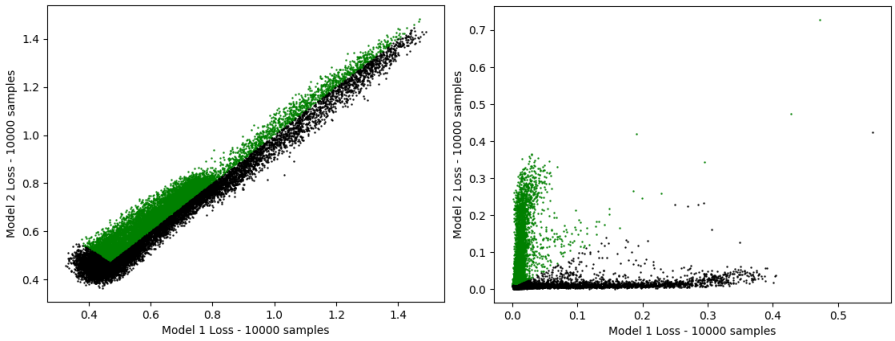


Fig. 2: Loss profile for 2-NNs ensemble in the Acrobot environment. On the left are loss profiles of data points of untrained subnetworks in the ensemble. We apply clustering of these data points into two bins (green and black). On the right, in the alignment of the data points along the axes (loss profile) is reflecting the successful specialization of the subnetworks after training of the ensemble of the two subnetworks (green and black).

In the first step of the clustering algorithm, each of n ensemble networks produces predictions for all data points in the training data. Next, for each data point, we collect the prediction losses for each subnetwork into a n-dimensional vector that we refer to as the *loss profile* of the data point. Third, we sort the data points in descending order of the euclidean norms of their loss profiles. Data points sorted towards the end of this ordering have small loss norms, indicating that they pose no particular challenge for any of the n subnetworks and, thus, have little information value for the specialization of the subnetworks. Conversely, data points near the beginning of the sequence are challenging and thus they should guide the optimization of clustering more strongly (Fig. 2). Therefore, we assign the data points into n bins by the following sequential process: we take a data point with the highest norm value from the sorted list of data points and delete this data point from the list. The *argmin* of the loss profile of this data point determines its destination bin if the bin still is filled below its maximal capacity (*bin size = number of all data points / n bins*). Otherwise, if the bin is already full, the destination bin is chosen by the index of the next lowest loss entry in the data point's loss profile.

2.2 Training

We train the addressing NN and the ensemble separately in two sequential phases (Fig. 3). In the first phase, the ensemble of 1..n subnetworks is trained for a fixed number of epochs. At the beginning of each epoch, we first apply the clustering algorithm described in algorithm 1 to fill n data bins. Each data bin corresponds to one of the subnetworks and is used to train the corresponding subnetwork. Thus, for each subnetwork, we sample data points from its corresponding bin in

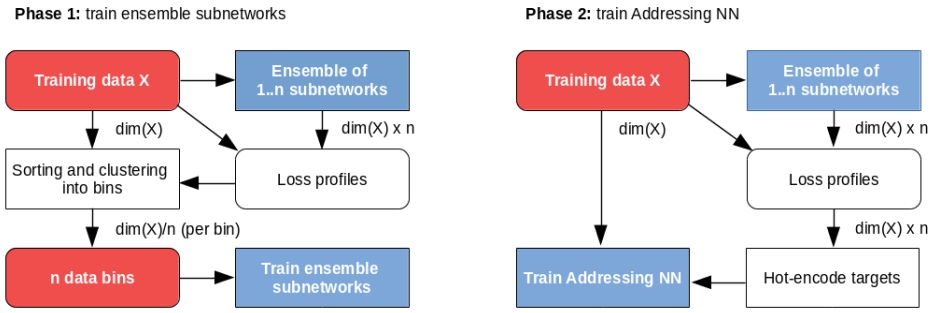


Fig. 3: Data flow graphs. Phase 1: Sorting and clustering of training data into equal-sized bins (one bin for each subnetwork in the ensemble) and ensemble training. Phase 2: Training of the Addressing NN using targets created by the loss profiles of the data points.

some random order, until all training samples of the bin have been used once. Then, we continue to the next subnetwork and its corresponding bin. When all subnetworks have been traversed, the current epoch is finished and the next epoch is started.

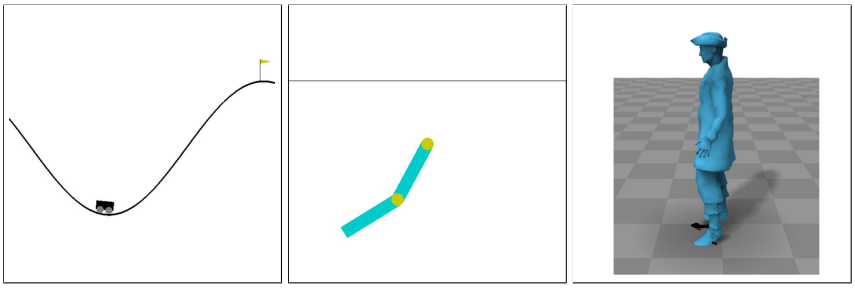
When a fixed number of epochs is completed, we start the second phase of training (Fig. 3), where we train the addressing NN for several epochs. We first compute an updated set of loss profiles that reflects the training progress of the ensemble networks during phase one. Then, for each data point (x, y) the *argmin* of its loss profile determines the target output of the addressing network for input x (i.e. the addressing network's target output is the k-unit vector, where k identifies the smallest component of the n-dimensional loss profile of (x, y)) With this choice of target output, the addressing network is adapted towards predicting the ensemble subnetwork with the lowest loss for an input $x \in X$. To this end, we train the addressing NN for several epochs, each consisting of sampling batches of a certain size from the training set and optimizing the network using these batches.

Finally, when both ensemble and addressing networks have been trained, the forward step consists of two simple parts: a prediction of the addressing NN which subnetwork to use given input x, and computing the prediction of this subnetwork for obtaining the output y.

3 Results

3.1 Mountaincar environment

For the first experiment, we train a model to simulate the dynamics of the mountaincar environment [11] shown in figure 4a. Its observation space and action space consists of two dimensions (position and velocity) and three dimensions (actions: push left, push right, no push), respectively. We encode the



(a) (b) (c)

Fig. 4: Environments of the experiments: Mountaincar (a), acrobot (b) and the virtual environment used in the phase-functioned neural network paper [2] (c).

Table 1: Addressing and losses in the mountaincar environment.

Mean loss of selected and not selected subnetworks	
mean subnetwork 1 loss selected	0.2 * 1e-6
mean subnetwork 1 loss not selected	262.6 * 1e-6
mean subnetwork 2 loss selected	1.2 * 1e-6
mean subnetwork 2 loss not selected	1861.3 * 1e-6
Subnetwork 1 missclassification per thousand data points	
number of missclassified datapoints	27
median loss of missclassified predictions	0.4 * 1e-6
mean loss of missclassified predictions	3.5 * 1e-6
max. loss of missclassified predictions	65.2 * 1e-6
Subnetwork 2 missclassification per thousand data points	
number of missclassified datapoints	1
median loss of missclassified predictions	7.8 * 1e-6
mean loss of missclassified predictions	7.8 * 1e-6
max. loss of missclassified predictions	7.8 * 1e-6

actions via one-out-of-k-hot-encoding for feeding it into the network, which extends the input dimension of our networks to five dimensions (state plus action). Output dimensionality amounts to two for each of the ensemble models as we predict the next state. Training data is derived from Deep Q-network training, where we recorded the data points, which are generated by the policy stepping in the environment during training and amounts to T=1,000,000 data points. Generating the next state is realized by one of the n subnetworks, mapping $f(s_t, a) = s_{t+1}$, while the addressing NN $g(s_t, a) = argmax(\{f_1, ..., f_n, \})$ predicts which model to use. We used an ensemble with of two subnetworks. Each

subnetwork has two hidden layers with 128 neurons. The addressing NN is configured with three layers (128, 128 and 64 neurons). We use the mean-squared-error as loss-function in this experiment. We used ADAM [12] optimizer in this experiment with standard-setting of learning rate $\mu = 1e^{-4}$ and $\beta = [0.9, 0.99]$.

The output of the environment and the states produced by the architecture are nearly identical. The mean squared error for an episode of 1000 steps between environment steps and predicted states amounts to $2.225e^{-5}$ for position prediction and $4.859e^{-8}$ for velocity prediction. The performance in terms of loss of both subnetworks can be seen in table 1. The addressing network is able to correctly distribute the samples the subnetworks are specialised on. In autoregressive runtime, the architecture could generate the same dynamics as the original environment. In contrast, a model with only one fully connected NN with 3 hidden layers with 512 neurons each was unable to learn the dynamics of the agent.

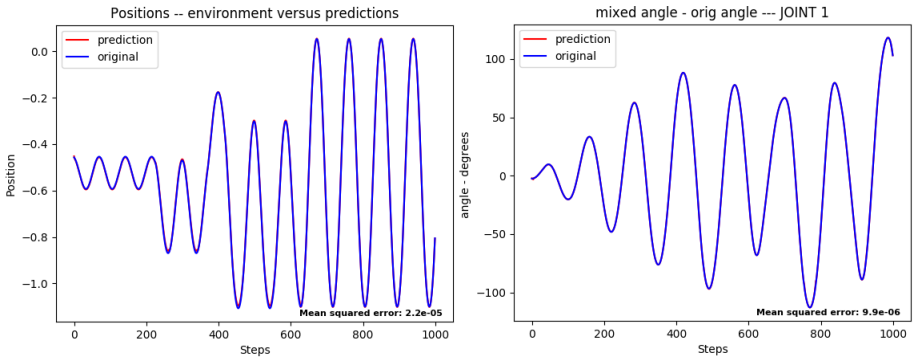


Fig. 5: Left: Comparison between ground truth and one-step-predictions in terms of position in the mountaincar environment. Right: Comparison between ground truth and one-step-predictions in terms of angle for the first joint in the acrobot environment.

3.2 Acrobot environment

The acrobot environment [13], shown in 4b, consists of two rotational joints with angles α_1, α_2 linked in a row. The six dimensional observation space consists of the tuple $\{\cos\alpha_1, \sin\alpha_1, \cos\alpha_2, \sin\alpha_2, \omega_1, \omega_2\}$, where ω_1, ω_2 are angle velocities for joint one and two, respectively. Its action space consists again of three actions. These three actions correspond to fixed torque values $(-1, 0, 1)$. We used one-out-of-k-hot-encoding of the actions to feed it to the network. We again use the learning process of a Deep-Q network to record 150,000 data points during training. In addition to that, the training data consists of 50,000 data points, in which the Acrobot was recorded while applying zero torque to highlight behavior in

the low-velocity range. We use an ensemble of eight NNs. We organize the input into 6 sequential observations, thus requiring 54 input neurons, while the output dimension for each subnetwork amounts to four $(\cos\alpha_1, \sin\alpha_1, \cos\alpha_2, \sin\alpha_2)$. The addressing NN has the same input as the ensemble models, while its output dimension is equal to the number of ensemble networks ($n = 8$). We build subnetworks with three hidden layers consisting of 256 neurons each. Two layers with 512 neurons are provided for the addressing NN. Furthermore, we equip the addressing NN with 8 heads of two layers with 256 neurons each. The output dimension of each of the eight heads is one and a softmax function is applied on the outputs of all eight heads. For optimization we chose to use batches of 32 data points to perform updates in both ensemble and addressing NN training. As activation functions for all neurons in all networks we use hyperbolic tangent. The loss is the mean absolute error. The optimzer is ADAM with standard setting of learning rate $\mu = 1e^{-4}$ and $\beta = [0.9, 0.99]$.

The architecture was able to predict the next state in terms of cosine and sine with high precision. The prediction of sine and cosine values plus the computation of the weighted angle using those exhibits an error of $1e^{-6}$. The velocities which were also fed to the network were computed using the current and the previous state and exhibit a higher error in the range of $1e^{-2}$. The errors were computed by comparing the output of the environment to the predicted output of the state. At autoregressive runtime, the architecture with an ensemble of eight networks was able to produce similar behavior as the original environment exhibits.

3.3 Prediction of cosine and sine values - Weighted angle

Predicting cosine and sine values is necessary for modeling the acrobot environment as described in 3.2. Using a function approximator for predicting cosine and sine values and constructing an angle from these predictions isn't straightforward due to imprecisions. Especially the edges of the intervals, that the inverse trigonometric functions arccosine and arcsine map cosine and sine values to, can cause problems. Further, both intervals for cosine translated to radians $[0, .., \pi]$ and for sine translated to radians $[-\frac{\pi}{2}, ..., \frac{\pi}{2}]$ are displaced by $\frac{\pi}{2}$ and together contain the information, in which quadrant of the unit circle the angle is located. Therefore, we use a weighted sum of both angles, which complementary uses the predictions to cover the edges of each other's intervals:

$$w = \frac{c}{c+s} \tag{1}$$

$$\alpha = (1-w) * \arccos(c) + w * \arcsin(s) \tag{2}$$

c represents the network's prediction of the cosine of an angle of a joint, while s is the network's prediction of the sine of the angle of the same joint. Variable w represents the weight, which is calculated by formula 1. When cosine angle is around 0 or π (refers to cosine of 1 and -1 respectively), then sine prediction is used and vice versa. In figure 6, the reconstruction of the angle using sine

and cosine values can be observed. We use this weighted angle at runtime in the forward step after training is done.

3.4 Phase-functioned neural network database

We trained our architecture on the training set provided in the phase-functioned neural network implementation [2]. For a detailed description of the training set, refer to [2]. The database consists of recorded motion capture data and provides about four million data samples. The input dimension of a sample is 342 dimensions and consists largely of positional information, as well as velocities in three different directions (x, y, z coordinates) for 31 joints which make up the agent. The output dimension per sample has 311 dimensions and also predicts position and velocities, plus rotational values of each joint.

The phase function of PFNN generates weights of a regression network from a number of trained NNs, which then predicts the next state. In the code implementation, the PFNN consists of four pre-trained networks and uses a cubic catmull-rom spline as the phase function. For the comparison between our architecture and PFNN, we choose the same number of parameters in the network ensemble as the implementation of PFNN provides (four ensemble networks,

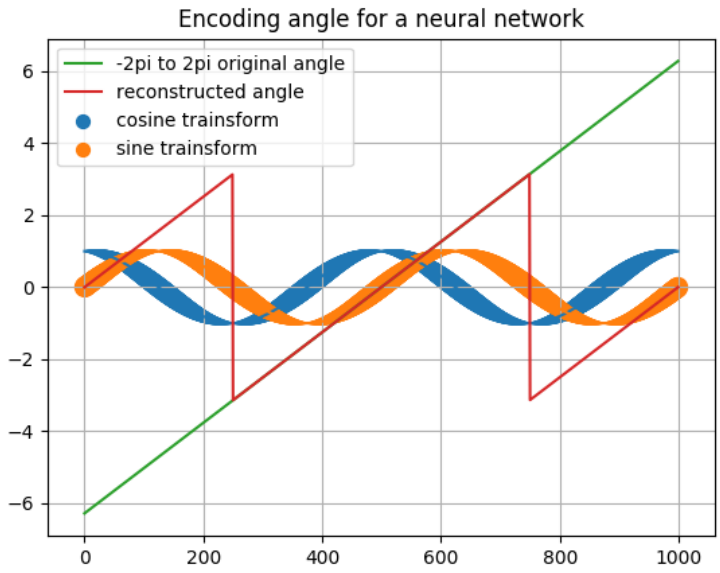


Fig. 6: Construction of angle by mixture of cosine and sine prediction. For illustrating, which portion of the angle relies on cosine prediction and which on the sine prediction, we highlighted the corresponding areas of sine and cosine by the curve width.

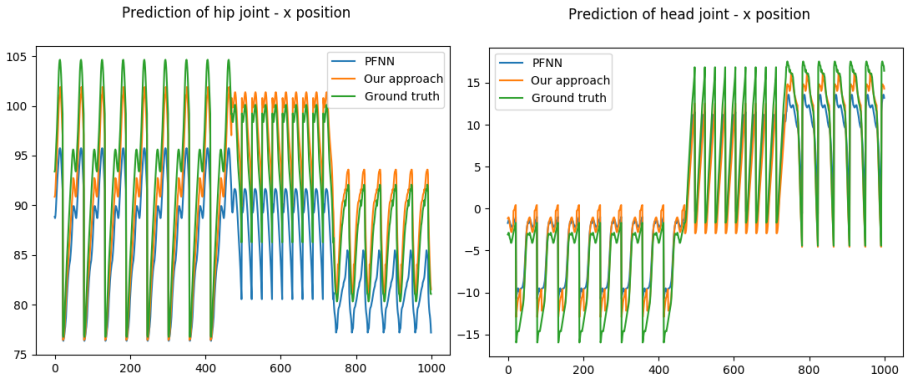


Fig. 7: Predictions of hip (left) and head joint (right) of trained model on a test set of the PFNN database.

with two hidden layers and 512 neurons per layer). In addition to that, the addressing network consists of two hidden layers with 512 neurons each. Our approach trained for forty epochs, each with about 80,000 training steps. The batch size was chosen to be 32.

The comparison was performed on a test set of the PFNN database. The size of the test set amounts to 40,000 data samples. The mean absolute error over the training set for PFNN amounts to 2.21. Our architecture is able to fit the same data, producing a much lower mean absolute error of 0.69. A series of one-step-predictions over 1000 steps of the hip joint in terms of forward-direction and for prediction of the head joint can be seen in figure 7.

4 Discussion

We presented an error-based addressing architecture for dynamic model learning, which is able to generate trajectory data conditioned by control commands. The key part is an addressing network that automates the selection of specialized models implemented by an ensemble of subnetworks. Specialization of the subnetworks during training is achieved through a self-organized partitioning of the training data set that is based on the loss profiles of the data points with the subnetworks as predictors.

We successfully simulated the dynamics of agents conditioned by control commands in three simulated environments (Fig. 4). The architecture successfully generated a well-working simulation of environmental dynamics. In direct comparison to the environment, it can approximate these dynamics with an error of about $1e^{-6}$. In autoregressive runtime, the proposed architecture generates agent's behavior in a similar fashion, when comparing it to the original environments. In the latter we could determine, that the addressing NN's choice which model to use is in line with the lowest loss for 77.5% of all training data points at the end of a training process. The addressing NN predicted the best subnetwork

from the ensemble with a high accuracy. The loss values for the data points, which were chosen for a specific model, were low compared to those not chosen (see Appendix A).

Furthermore, we showed that the proposed architecture performs well on the training set of the PFNN database [2]. In comparison to the PFNN model provided in the original implementation, the mean absolute error of the present model was lower than for the PFNN model. However, the PFNN in the code implementation uses regularization (dropout layers after each hidden layer with 0.7 dropout rate) and we still need to measure the performance of the proposed architecture in the autoregressive mode in the virtual environment provided by the PFNN implementation.

As a future work we can investigate several methods to perform a forward-step prediction. In the default case, we compute the output $g(X) = (b_1, ..., b_n)$, where b_k is an estimation value representing model $k \in f_1, ..., f_n$ and apply $argmax(b_1, ..., b_n)$ to determine which model to choose. We can also use $(b_1, ..., b_n)$ as a probability of which model to choose to further the fluctuation between subnetworks. Another variant is using output $(b_1, ..., b_n)$ as the weight for outputs computed by the subnetworks to create a mixture of states. At last, to create a pseudo-model by creating a weighted mixture of parameters of each subnetwork $f_1, ..., f_k$.

References

1. Teun Adrianus Van Dijk, Walter Kintsch, et al. Strategies of discourse comprehension. 1983.
2. Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 36(4):42, 2017.
3. Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.
4. Dario Pavllo, David Grangier, and Michael Auli. Quaternet: A quaternion-based recurrent model for human motion. *arXiv preprint arXiv:1805.06485*, 2018.
5. Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.
6. Łukasz Kidziński, Sharada Prasanna Mohanty, Carmichael F Ong, Zhewei Huang, Shuchang Zhou, Anton Pechenko, Adam Stelmaszczyk, Piotr Jarosik, Mikhail Pavlov, Sergey Kolesnikov, et al. Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. In *The NIPS'17 Competition: Building Intelligent Systems*, pages 121–153. Springer, 2018.
7. Malte Schilling and Andrew Melnik. An approach to hierarchical deep reinforcement learning for a decentralized walking control architecture. In *Biologically Inspired Cognitive Architectures Meeting*, pages 272–282. Springer, 2018.
8. Andrew Melnik, Sascha Fleer, Malte Schilling, and Helge Ritter. Modularization of end-to-end learning: Case study in arcade games. *arXiv preprint arXiv:1901.09895*, 2019.
9. Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. Mosaic model for sensorimotor learning and control. *Neural computation*, 13(10):2201–2220, 2001.

10. Kai Konen, Timo Korthals, Andrew Melnik, and Malte Schilling. Biologically-inspired deep reinforcement learning of modular control for a six-legged robot. In *2019 IEEE International Conference on Robotics and Automation Workshop on Learning Legged Locomotion Workshop,(ICRA) 2019, Montreal, CA, May 20-25, 2019*, 2019.
11. Andrew William Moore. Efficient memory-based learning for robot control. 1990.
12. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
13. Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

A Appendix - Performance of the architecture in the acrobot environment

The prediction generated by the addressing NN and the losses computed between the output of the ensemble models and the target outputs correlated as designed, resulting in a 77.5% overlap between addressing NN prediction and minimal loss. Figure 8 shows this correlation. The distribution of about 200,000 data points is approximately uniform, excluding *model* 3 and *model* 7. In theory, each model should perform on 25,000 data points, which is mostly the case. The addressing network performs slightly worse in terms of uniformity, having also *model* 1 as an additional outlier. The reasons for that are hypothesized in the discussion.

In figure 11, we can observe the prediction of the addressing NN when distributing data points over models. On the x-axis the predicted values are divided into intervals of $[0, ..., 0.1]$, $[0.1, ..., 0.2]$ and so forth. The shown x-value represents the upper bound of the interval. The accumulated number of points in the corresponding interval is shown on the y-axis. Most of the data points get distributed between a value of 0.9 to 1.0 in all models.

Additionally, figure 10 shows the losses of the distributed data points. After the data points get distributed in clusters by the addressing NN, the chosen models perform a forward step on these data points and the L1-loss is computed. They again get subdivided into ten equally sized intervals, ranging from lowest to highest loss per model. As we can see, most data points fall into the first interval with the lowest upper bound.

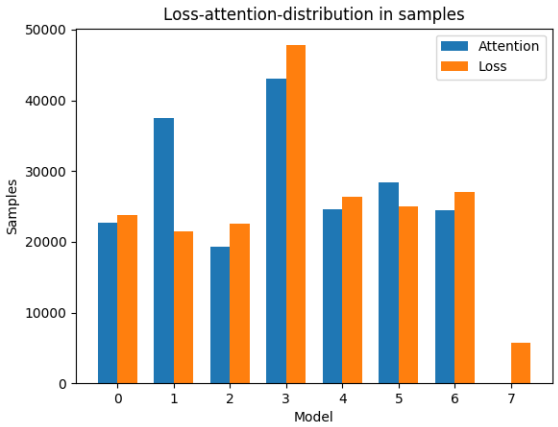


Fig. 8: data points per model in terms of distribution by minimal loss and by prediction.

Next, we calculate the mean loss over the training data, as well as the mean loss over the clusters of training data provided by the sorting process for each

ensemble network. For comparing the performance of the architecture to traditionally used feed-forward NNs, we also trained three fully connected models with three hidden layers of different sizes per model on the same training data. The training duration and optimizer settings were the same as for the architecture, but the activation function was changed to ReLu. Then the mean loss was computed for each of the three models over the whole training data. In figure 9, this can be observed. The performance of each ensemble model over the whole training data is worse than the fully connected models, although they exhibit a lower mean loss on each of the sorted training clusters. Furthermore, a low loss on the specific training cluster corresponds to a high loss on all training data.

In figure 9 we show, that the architecture is able to outperform basic NNs, which are trained on all training data points. Specialized models had a bigger mean loss over the complete training set. However, the mean loss over the clusters of training data provided by the sorting process for each subnetwork could be shown to be lower (see 9). The training process of the NN architecture could, therefore, be interpreted as more sample efficient as the commonly used training procedure because the subnetworks were trained on a lower number of data points due to the sorting mechanism. The here developed technique could also dynamically subdivide training data into coherent clusters, such that the subnetworks learned specific parts of the dynamics, which an environment can provide.

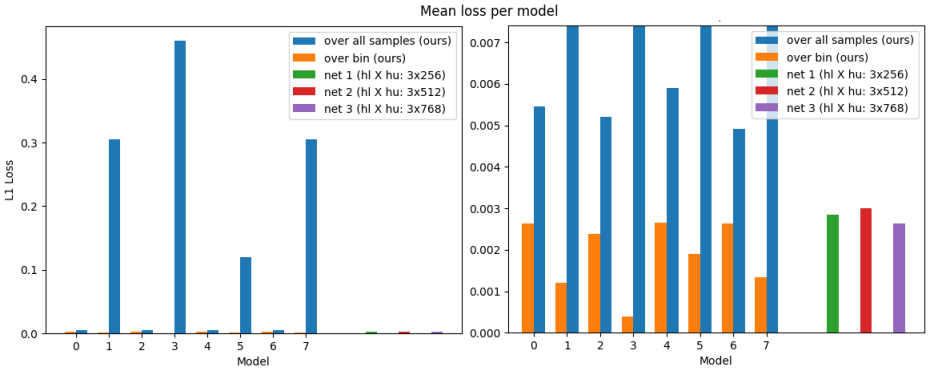


Fig. 9: Mean losses of all data points per ensemble model (right column) and of the distributed data points per ensemble model (left column). Additionally, mean losses for three fully connected neural networks. (right: focus on the distributed losses)

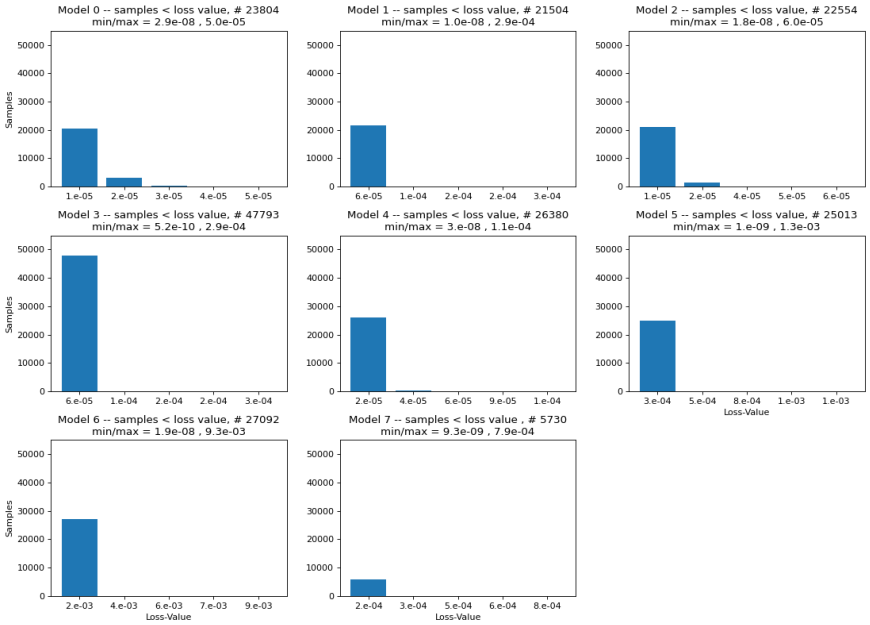


Fig. 10: Losses of the distributed data points.

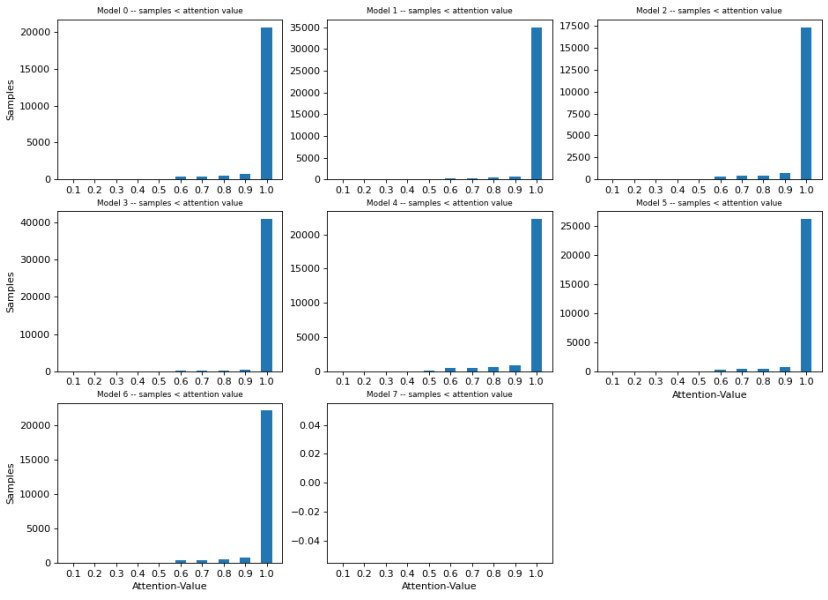


Fig. 11: Prediction values of the addressing neural network for data points for distributing them over the 8 models.