

# Learning real manipulation tasks from virtual demonstrations using LSTM

Rouhollah Rahmatizadeh<sup>1</sup>, Pooya Abolghasemi<sup>1</sup>, Aman Behal<sup>2</sup> and Ladislau Bölöni<sup>1</sup>

**Abstract**—Robots assisting disabled or elderly people in activities of daily living must perform complex manipulation tasks. These tasks are dependent on the user’s environment and preferences. Thus, learning from demonstration (LfD) is a promising choice that would allow the non-expert user to teach the robot different tasks. Unfortunately, learning general solutions from raw demonstrations requires a significant amount of data. Performing this number of physical demonstrations is unfeasible for a disabled user.

In this paper we propose an approach where the user demonstrates the manipulation task in a virtual environment. The collected demonstrations are used to train an LSTM recurrent neural network that can act as the controller for the robot. We show that the controller learned from virtual demonstrations can be used to successfully perform the manipulation tasks on a physical robot.

## I. INTRODUCTION

Assistive robotics, whether in the form of wheelchair mounted robotic arms or mobile robots with manipulators, promises to improve the independence and quality of life of the disabled and the elderly. Such robots can help users to perform Activities of Daily Living (ADLs) such as self-feeding, dressing, grooming, personal hygiene and leisure activities. Almost all ADLs involve some type of object manipulation. While most current systems rely on remote control, there are ongoing research efforts to make assistive robots more autonomous. This includes the identification of the objects, grasping and manipulation executed on behalf and in collaboration with the disabled user [1], [2], [3]. One of the challenges of autonomous manipulation for ADLs is that the robot needs to learn the specific home environment, physical objects as well as the needs and preferences of the user. As the user is a non-programmer, a convenient approach is learning from demonstration (LfD) where the user demonstrates the task by guiding the robotic arm or through teleoperation. However, due to the specific circumstances of ADLs, it is unfeasible to collect large numbers of physical demonstrations. For instance, the manipulated objects and the environment can be fragile and the wheelchair-bound users might have difficulty recovering from a failed manipulation task, such as a dropped cup. For similar reasons, reinforcement learning done in the physical environment is also not a feasible choice.

In this paper we propose an approach where the users demonstrate the tasks to be performed in a virtual environment that is similar to their living environment and objects.

<sup>1</sup> Department of Computer Science, University of Central Florida, Orlando FL 32816

<sup>2</sup> Department of Electrical and Computer Engineering, University of Central Florida, Orlando FL 32816

This allows the collection of a larger number of demonstrations under various scenarios. For each demonstration we collect the pose of the objects involved as well as the gripper. The collected trajectories are used to train a neural network, which will serve as the robot controller: at each timestep it receives as input the current state and predicts the next waypoint in the trajectory and open/closed status of the gripper. The general flow is illustrated in Figure 1.

## II. RELATED WORK

**Recurrent Neural Networks.** Recurrent Neural Networks (RNNs) are an effective way to model and reproduce patterns in sequential data. Some of successful applications include handwriting generation [4], language modeling [5], machine translation [6], speech recognition [7], visual recognition [8], and image captioning [9].

Although recurrent neural networks had been proposed as early as the 1980s, early versions suffered from the difficulty of training over sequential data, due to the difficulty to account for events occurring at different times in the training data sequences (the “vanishing error gradient” problem). Newer RNN models such as LSTMs [10] feature explicit gating mechanism that helped in storing and retrieving information over long time periods. In recent years, similar mechanisms were proposed such as Gated Recurrent Units (GRU) [11].

**Autonomous trajectory execution.** One of the most effective approaches for teaching robots to execute a desired task is Learning from Demonstration (LfD). This technique extracts policies from example state to action mappings [12]. In this method, sample trajectories for performing a task is demonstrated by the user. The challenge is to extend these demonstrations to unseen situations. Successful LfD applications include autonomous helicopter maneuvers [13], playing table tennis [14], [15], object manipulation [16], and making coffee [17].

Acquiring a sufficient number of demonstrations is the common challenge of LfD systems. To overcome this problem, some researchers proposed cloud-based and crowd-sourced data collection techniques [18], [19], [20]. Another way to reduce the number of necessary demonstrations is to hand-engineer task-specific features [21], [22]. Using neural networks eliminates the need to hand-engineer features. Therefore, deep neural networks were recently applied to complex robotic tasks and achieved interesting results. For instance, [23] used feed-forward neural networks to map robot’s visual input to control commands. In this approach, the visual input is processed using a convolutional neural

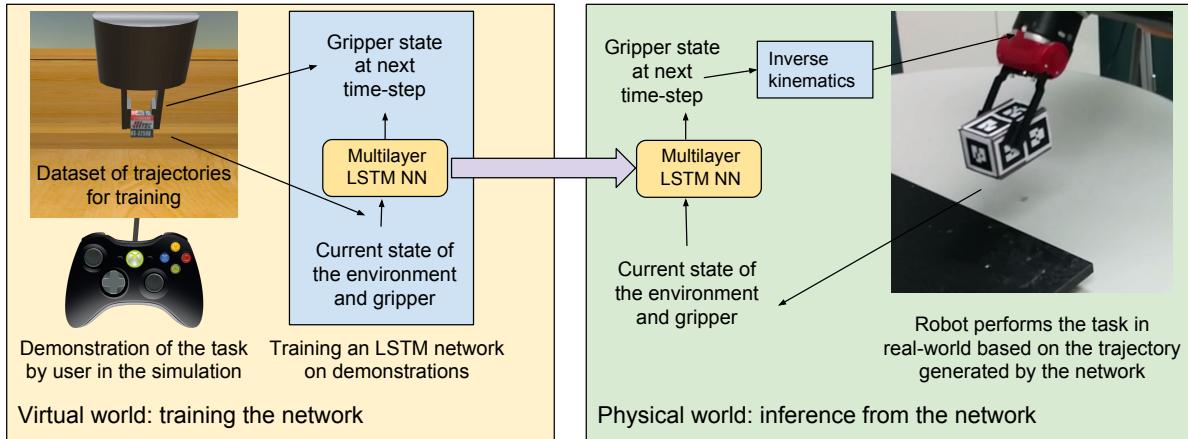


Fig. 1: The general flow of our approach. The demonstrations of the ADL manipulation tasks are collected in a virtual environment. The collected trajectories are used to train the neural network controller. The trained controller is then transferred to the physical robot.

network (CNN) to extract 2D feature points, then it is aggregated with the robot's current joint configuration, and fed into a feed-forward neural network. The resulting neural network will predict the next joint configuration of the robot. A similar neural network architecture is designed to address the grasping problem [24].

In this paper, we use recurrent neural networks and show that they outperform feedforward networks in complex tasks. We show that using recurrent neural networks instead of feedforward networks mitigates the need for using more advanced algorithms to change the data distribution to achieve good results at test time [25].

### III. METHOD

#### A. Collecting demonstrations in a virtual environment

1) *The virtual environment:* The first step of our workflow is for disabled users to be able to demonstrate their ADLs to the robot in a virtual environment. To enable this we designed in the Unity3D game engine a virtual environment modeling a table with an attached shelf that can hold various objects. The virtual environment also contains a simple two-finger gripper that can be opened and closed to grasp and carry an object. The user can use the mouse and keyboard to open/close the gripper, as well as to move and rotate it in the 3D Cartesian space. Alternatively, a joystick can be used for the same tasks. Thus, the gripper has 7 degrees of freedom. The environment can also contain one or more movable objects, which have their own position and orientation. Unity3D simulates the basic physics of the real world including gravity, collision between objects and friction.

2) *Trajectory representation:* We represent the state of the virtual environment as the collection of the poses of the  $M$  movable objects  $q = \{o_1 \dots o_M\}$ . The pose of an object is represented by the vector containing the position and rotation quaternion with respect to the origin  $o = [p_x, p_y, p_z, r_x, r_y, r_z, r_w]$ . During each step of a demonstration, at time step  $t$  we record the state of the environment

$q_t$  and the pose of the end-effector augmented with the open/close status of the gripper  $e_t$ . Thus a full demonstration can be recorded as a list of pairs  $d = \{(q_1, e_1) \dots (q_T, e_T)\}$ . The duration of a trajectory  $T$  is determined by the moment when the user successfully finishes or abandons the experiment and it varies from demonstration to demonstration. The temporal resolution at which the states are recorded depend on the requirements of the experiment.

3) *Manipulation tasks considered:* The majority of ADLs involve object manipulation, and a very large majority of these involve objects located on horizontal surfaces such as tables. For our experiments we considered two manipulation tasks that are regularly found as components of ADLs: pick and place and pushing to a desired pose.

The *pick and place* task involves picking up a small box located on top of the table, and placing it into a shelf above the table. The robot needs to move the gripper from its initial random position to a point close to the box, open the gripper, position the fingers around the box, close the gripper, move towards the shelf in an orientation where it will not collide with the shelf, enter the shelf, and finally open the gripper to release the box.

The *pushing to desired pose* task involves moving and rotating a box of size  $10 \times 7 \times 7$  cm to a desired area only by pushing it on the tabletop. In this task, the robot is not allowed to grasp the object. The box is initially positioned in a way that needs to be rotated by  $90^\circ$  to fit inside the desired area which is 3cm wider than the box in each direction. The robot starts from an initial gripper position, moves the gripper close to the box and pushes the box at specific points at its sides to rotate and move it. If necessary, the gripper needs to circle around the box to find the next contact point.

4) *Collected dataset:* Using the virtual environment described above, we collected a series of demonstrations for both manipulation tasks. The demonstrations were collected from a single user, in the course of multiple sessions. In each session, the user performed a series of demonstrations

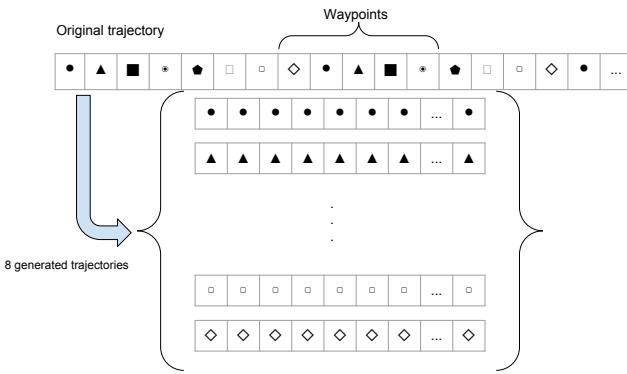


Fig. 2: Creating multiple trajectories from a demonstration recorded at a higher frequency.

for each task. The quality of demonstrations varied: in some of them, the user could finish the task only after several tries. For instance, sometimes the grasp was unsuccessful, or the user dropped the object in an incorrect position and had to pick it up again. After finishing a demonstration, the user was immediately presented with a new instance of the problem, with randomly generated initial conditions. All the experiments had been recorded in the trajectory representation format presented above, at a recording frequency of 33Hz. However, we have found that the neural network controller can be trained more efficiently if the trajectories are sampled at a lower rate. We found a sampling rate of 4Hz to give the best results.

Although recording in the virtual environment allowed us to record significantly more demonstrations than it would have been possible with a physical robot, we have found that we needed ways to improve the number of training trajectories. We have done this by exploiting both the properties of the individual tasks and the capabilities of our trajectory recording technique.

First, we noticed that in the pick and place task the user can put the object to any location on the shelf. Thus we were able to generate new synthetic training data by shifting the existing demonstration trajectories parallel to the width of the shelf. As the pushing to desired pose task requires a specific coordinate and pose to succeed, this approach is not possible for the second task.

The second observation was that by recording the demonstration at 33Hz but presenting the training trajectories at only 4Hz, we have extra trajectory points. These trajectory points can be used to generate multiple independent trajectories at a lower resolution. The process of the trajectory generation by frequency reduction is shown in Figure 2.

Table I describes the size of the final dataset. As the table shows, the average number of waypoints in a trajectory for the pick and place task was 20.68, while the same number was 28.61 for the push to desired pose task. This data, based on the human input, shows that the second task was more difficult than the first *for the human operator*.

Task	Pick and place	Push to pose
Raw demonstrations	650	1614
No. of demonstrations after shift	3900	-
No. of demonstrations after frequency reduction	31,200	12,912
Total no. of waypoints	645,198	369,477
Average demonstration length (waypoints)	20.68	28.61

TABLE I: The size of the datasets for the two studied tasks

### B. The neural network based robot controller

The next step of our workflow is to design and train a neural network based robot controller that is able to generate robot trajectories. This controller takes as input the pose of the objects involved and the pose and open/close status of the gripper at time  $t$ . The output of the controller is a prediction of the pose and the open/closed status of the gripper at time  $t + 1$ . During training, this prediction is used to generate the error signal. During the deployment of the trained network, the prediction represents the desired pose of the end actuator which the robot needs to achieve through its inverse kinematics calculations.

In this series of experiments, we have used a separate controller for the pick and place and the push to desired pose tasks<sup>1</sup>. These controllers have a similar network architecture but had been trained on the specific tasks.

This leads us to the choice of the architecture of the neural network. In particular, there are two important decision points: the nature of the network layers and the cost function used to train the network.

Our choice for this architecture is to use an LSTM recurrent neural network and rely on mixture density networks (MDNs) to predict the probability density of the output. The error signal, in this case, is based on the negative logarithm likelihood of the next target waypoint given the probability density implied by the MDN.

In order to determine whether our proposed approach represents a progress over the current state of the art, we need to compare it with current state of the art solutions. For instance, we can compare our solution with the more popular choice of using the mean squared error (MSE) as an error signal. Another term of comparison can be obtained by comparing our approach to other researchers solving similar tasks. For instance, [23] uses convolutional layers (for extracting poses from images) followed by feedforward layers to give a single prediction about the next waypoint in the trajectory. In this work we do not deal with robot vision, thus as the basis of comparison, we have also created an implementation that matches the structure of the network from [23] which follow the convolutional layers.

Thus we have implemented four choices for the controller structure: LSTM with MDN, LSTM with MSE, feedforward network with MDN and feedforward network with MSE. Due to limited space, we will only describe the LSTM with MDN controller. For the other controllers, all the comparable choices had been similar.

<sup>1</sup>Training a multi-task controller is an ongoing work for our group.

### C. The LSTM-MDN robot controller

Let us first justify the intuition behind the use of LSTM and MDN technologies in the implementation of our robot controller.

One of the first insights is that the solution to both manipulation tasks contain a series of individual movements which need to be executed in a specific sequence. Although both tasks can be solved in several different ways, the individual movements in them cannot be randomly exchanged. In order to successfully solve the task, the robot needs to choose and commit to a certain solution. While it is technically possible that this commitment will be encoded in the environment outside the robot, we conjecture that a robot controller that has a memory that can store these commitments will perform better. The requirement of a controller with a memory leads us to the choice of recurrent neural networks, in particular, one of the most widely used model, the LSTM [10].

LSTM can encode the useful information of the past in a single or multiple layers. Each layer accepts as input the output of the previous layer concatenated with the network input  $x_t = \{e_t, q_t\}$ . Each LSTM layer predicts its output based on its current input and its internal state. In addition, the LSTM updates its internal state at each time-step to be used in the next prediction. The concatenation of outputs of all layers will be used to predict the output of the network  $y_t$ . In our controller we are using three LSTM layers with 50 nodes each as shown in Figure 3.

The second intuition applies to the choice of the output layer and error signal used for the training of the neural network. For both tasks we are considering there can be multiple solutions. For instance, for the push to pose task, the robot might need to push the box in a direction parallel with its diagonal. This can be achieved by either (a) first pushing the shorter side of the box followed by a push on the longer side or (b) the other way around. However, by averaging these two solutions we reach a solution where the grasper would try to push the corner of the box, leading to an unpredictable result. This leads us to the conjecture that a multi-modal error function would perform better than the unimodal MSE. The approach we chose is based on Mixture Density Networks (MDN) [26] which use the output of the network to predict the parameters of a mixture distribution. Once we have the parameters of the mixture distribution, we can draw a sample and use it as the output of the network (which, in our case is the next pose of the gripper). Unlike the model with MSE cost which is deterministic, this approach can model stochastic behaviors to be executed by the robot. The probability density of the next waypoint can be modeled using a linear combination of Gaussian kernel functions

$$p(y|x) = \sum_{i=1}^m \alpha_i(x) g_i(y|x) \quad (1)$$

where  $\alpha_i(x)$  is the mixing coefficient,  $g_i(y|x)$  is a multivariate Gaussian, and  $m$  is the number of kernels. Note that both the mixing coefficients and the Gaussian kernels are conditioned on the complete history of the inputs till current

timestep  $x = \{x_1 \dots x_t\}$ . This is because the concatenation of the output of all layers which is used to estimate the mixing coefficients and Gaussian kernels is itself a function of  $x$ . The Gaussian kernel is of the form

$$g(y|x) = \frac{1}{(2\pi)^{c/2} \sigma_i(x)} \exp \left\{ -\frac{\|y - \mu_i(x)\|^2}{2\sigma_i(x)^2} \right\} \quad (2)$$

where the vector  $\mu_i(x)$  is the center of  $i$ th kernel. We do not calculate the full covariance matrices for each component, since this form of Gaussian mixture model is general enough to approximate any density function [27].

The parameters of the Gaussian kernels  $\mu_i(x)$ ,  $\sigma_i(x)$  and mixing coefficients  $\alpha_i(x)$  are represented by the layer M in Figure 3. To accomplish this, layer M needs to have one neuron for each parameter. Thus layer M will have a width  $(c+2) \times m$ , containing  $c \times m$  neurons for  $\mu_i(x)$ ,  $m$  neurons for  $\sigma_i(x)$ , and another  $m$  neurons for  $\alpha_i(x)$ . This layer is fully connected to the concatenation of layers  $H_1$ ,  $H_2$  and  $H_3$ .

To satisfy the constraint  $\sum_{i=1}^m \alpha_i(x) = 1$ , the corresponding neurons are passed through a softmax function. The neurons corresponding to the variances  $\sigma_i(x)$  are passed through an exponential function and the neurons corresponding to the means  $\mu_i(x)$  are used without any further changes. Finally, we can define the error in terms of negative logarithm likelihood

$$E_{MDN} = -\ln \left\{ \sum_{i=1}^m \alpha_i(x) g_i(y|x) \right\} \quad (3)$$

*Training the controller:* The network model is implemented in Blocks [28] framework that is built on top of Theano [29]. The network is unrolled for 50 time steps. All the parameters are initialized uniformly between -0.08 to 0.08 following the recommendation by [6]. Stochastic gradient descent with mini-batches of size 10 is used to train the network. RMSProp [30] with initial learning rate of 0.001 and decay of 0.99-0.999 (based on number of examples) is used to divide the gradients by a running average of their recent magnitude. In order to overcome the exploding gradients problem, the gradients are clipped in the range [-1, 1]. We use 80% of the data for training and keep the remaining 20% for validation. We stop the training when the validation error does not change for 20 epochs.

### D. Transferring the controller to the real robot

The last step of the process is to transfer the trained controller to a real robot. We have used a Rethink Robotics Baxter robot for this purpose. Transferring the controller to the physical robot opened several new challenges. As the controller provides only the next pose of the end effector, the controller had been augmented with inverse kinematics calculations to calculate the trajectory in the Baxter robot arms' joint space.

Another challenge is that while in the virtual world we had perfect knowledge of the pose of the effector and all the objects in the environment, we needed to acquire this

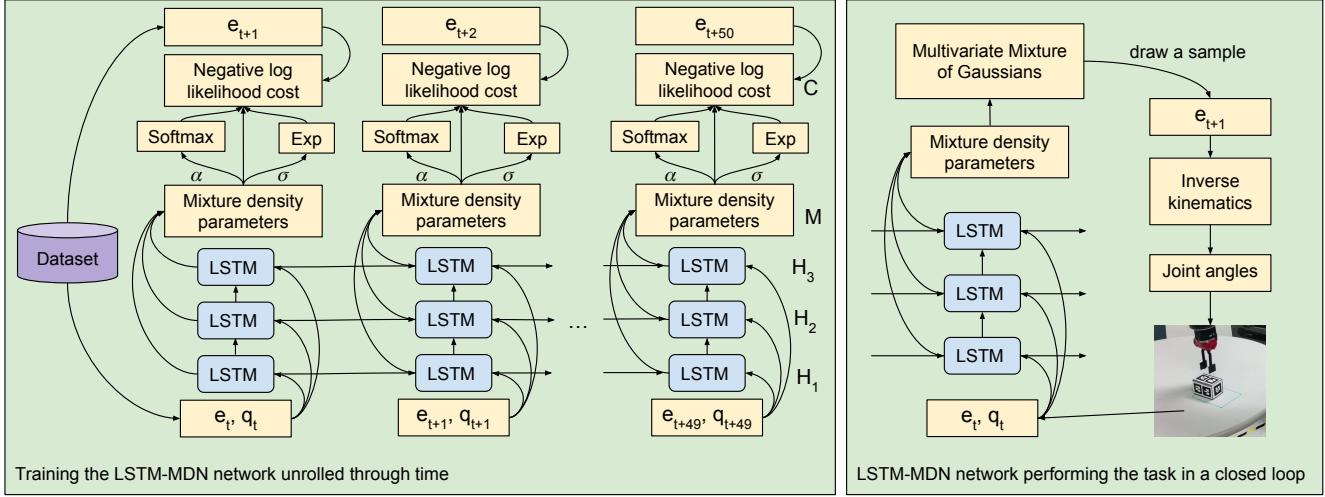


Fig. 3: The training and evaluation phase. During the training the LSTM network is unrolled for 50 time-steps. The gripper pose and status (*open/close*)  $e_t$  and the pose of relevant objects  $q_t$  at time-step  $t$  is used as input and output of the network to calculate and backpropagate the error to update the weights. During the evaluation phase, the mixture density parameters are used to form a mixture of Gaussians and draw a sample from it. The sample is used to control the robot arm.

information through sensing. In the work described in this paper we do not deal with computer vision: our controller architecture only performs robot arm control. To supplant the missing vision component, we relied on a Microsoft Kinect sensor and objects annotated with markers to track the pose of the objects. One of the problems with this approach is that the robot arm might occlude the view of the sensor. The Kinect sensor was placed close to the table to reduce the chance of occlusion, however, occlusions may still occur if the robot's arm is placed between the object and the Kinect.

Another challenge is the fact that the waypoints generated by the controller are relatively far away, leading to a jerky motion. While generating the trajectory at each time-step, we use interpolation in joint space to fill in the gap between the current waypoint and previous one and make the robot's movement smooth and slow.

Finally, we found that the trajectory described by the controller can not always be executed by the Baxter arm at the same timestep as in the virtual environment. Sometimes it takes longer (a few seconds) for the Baxter to reach the desired next end-effector pose and sometimes it is quite fast (a few milliseconds). Therefore, we use a dynamic execution rate to wait between execution of each waypoint. Concretely, the algorithm waits for .2sec and checks if the difference between the current pose of the gripper and the predicted one is below a certain threshold. If yes, it commands the robot to go to the next waypoint, otherwise it waits in a loop until the end-effector reaches the desired pose or timeout occurs which means that the end-effector cannot reach that pose (either because inverse kinematic fails or a collision occurs).

#### IV. EXPERIMENTAL VALIDATION

In the following we describe the results of a series of experiments performed in the virtual and physical environ-

ment. Our goals with the experimentation were to verify two hypotheses:

- Hypothesis 1: We conjecture that for the manipulation tasks we consider, the choices of LSTM for the neural network architecture and MDN for the error signal confer advantages over feedforward networks and MSE respectively.
- Hypothesis 2: We conjecture that the proposed architecture allows us to transfer a controller learned in a virtual environment to the control of a physical robot executing a real-world version of the same task.

##### A. Validating hypothesis 1: comparing network architectures in the virtual world

In order to validate hypothesis 1, in addition to our proposed architecture involving LSTM and MDN we have implemented and trained all the other combinations of the proposed approaches. Thus we trained four different neural network based robot controllers of the following architectures and parametrization:

- FeedForward-MSE: 3 layers of fully connected feedforward network with 100 neurons in each layer and mean squared error as the cost function.
- LSTM-MSE: 3 layers of LSTM with 50 memory states in each layer and mean squared error as the cost function.
- FeedForward-MDN: Mixture density network containing 3 fully connected feedforward layers with 100 neurons in each layer. The mixture contains 20 Gaussian kernels.
- LSTM-MDN: Mixture density network containing 3 layers of LSTM with 50 memory states in each layer. The mixture contains 20 Gaussian kernels.

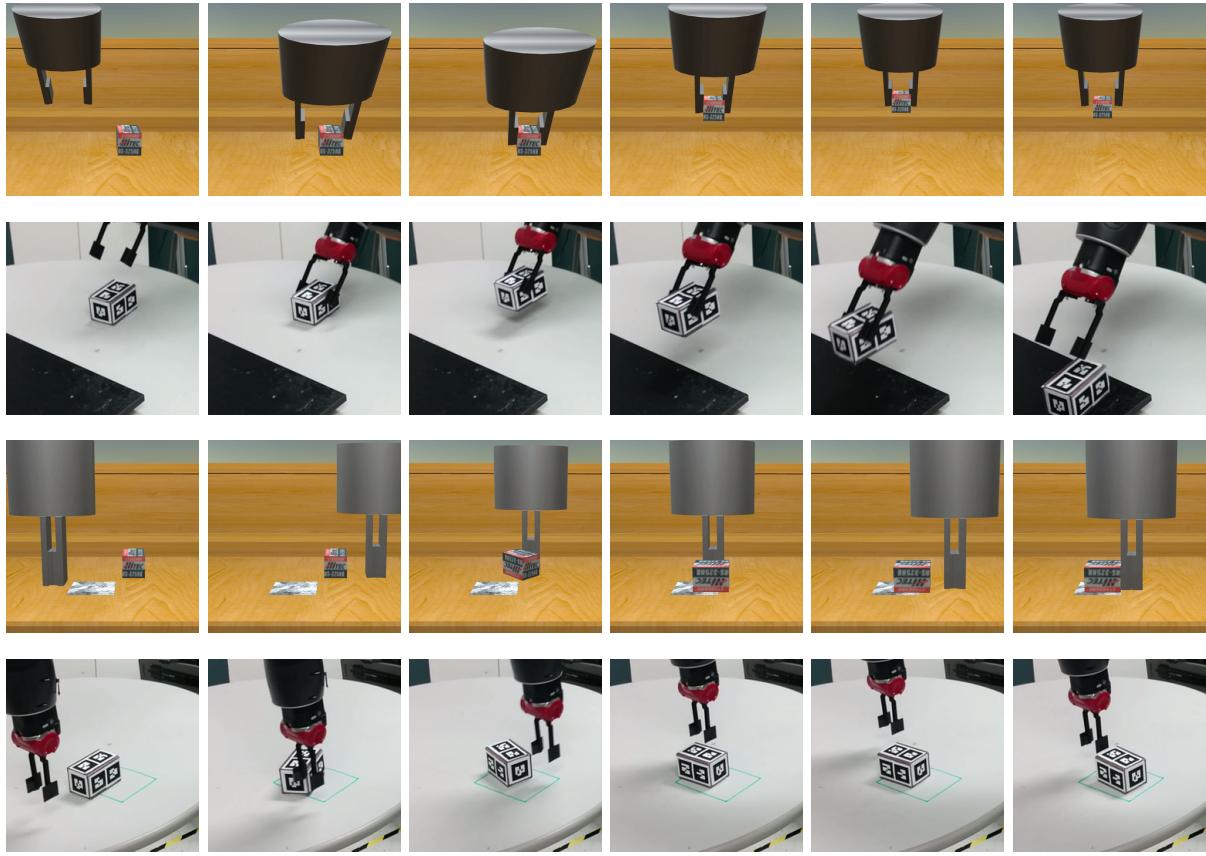


Fig. 4: A sequence of images showing the autonomous execution of pick and place in simulation (first row), pick and place in real world (second row), pushing in simulation (third row), and pushing in real world (fourth row). The robot is controlled by a mixture density network with 3 layers of LSTM.

The LSTM-MDN network was illustrated in Figure 3. The training phase of other baseline approaches are shown in Figure 5.

Each network had been separately trained for the pick and place and the push to desired pose respectively, in effect creating 8 different controllers. The resulting controllers had been tested in the virtual environment as follows. The virtual robot had to perform randomly generated tasks 20 times. If it can not complete the task in a limited time (1 minute for the first task and 2 minutes for the second one), we count the try as a failure and reset the position of the box.

Controller	Pick and place	Push to pose
Feedforward-MSE	0%	0%
LSTM-MSE	85%	0%
Feedforward-MDN	95%	15%
<b>LSTM-MDN</b>	<b>100%</b>	<b>95%</b>

*Feedforward-MSE.* The feedforward-MSE network with a Markov assumption was not able to complete the tasks even once. In the pick and place task, it learns to follow the box and sometimes close the gripper, however, it stops there and does not continue towards the shelf. The reason might be that the user usually pauses after closing the gripper to see if the grasp is successful or not. Since the gripper does not move for a few waypoints and then continues to

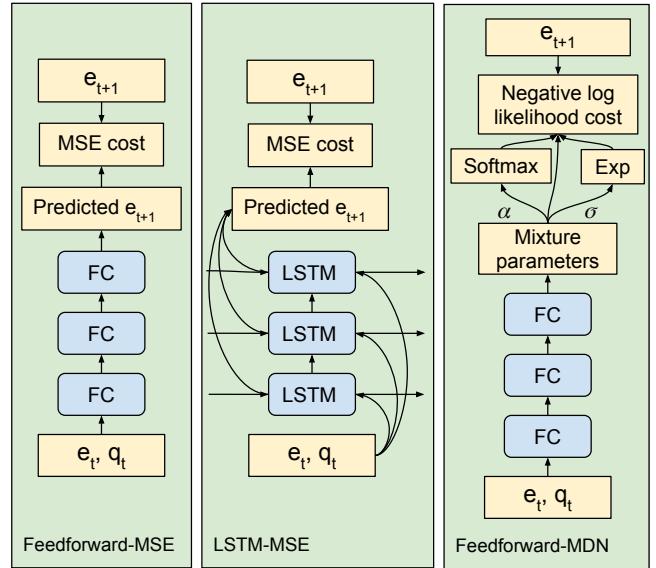


Fig. 5: Baseline network architectures compared to the LSTM-MDN approach.

move, the memoryless network fails to determine when the gripper should stop and when it should continue towards the shelf. What this model learns is basically the average of next waypoints without considering the past trajectory.

*LSTM-MSE.* This network learns a deterministic policy to predict the next waypoint based on the past trajectory by taking the average of next waypoints in the example demonstrations. Therefore, in the pick and place task that there is usually only one solution to the problem, it works relatively fine. However, in the pushing task that usually there are multiple solutions that the user has chosen to perform arbitrarily, the model takes their average which might not be a valid solution.

*Feedforward-MDN.* This method finds a probability distribution of the next waypoint without considering the past. Therefore, the trajectory does not look smooth as in the case of using LSTM especially when there are different solutions as in the case of pushing task. However, it tries and fails multiple times until it sometimes successfully performs the task.

*LSTM-MDN.* This method learns to successfully perform the tasks in most of the cases since it does not have the limitations of the other methods, i.e. it predicts the complete probability distribution of the next waypoint considering the past. One interesting observation was that it learns to recover from failure; for instance, when the grasp fails in the pick and place task, the gripper does not continue towards the shelf without the box, instead it tries the grasp one more time. Similarly, in the pushing task, when the force to the box was not enough to put it into the desired location, it tries again. These behaviors are learned by the network based on the performance of the user in similar situations.

#### B. Validating hypothesis 2: Comparing performance in the virtual and physical world

Once we have established that the LSTM-MDN approach yields the best performance on both tasks, we proceeded to verify that the LSTM-MDN controller trained on virtual demonstrations can actually perform on the physical robot. To verify this we subjected both the virtual and the physical robots to the same tasks. The sequence of images in the Figure 4 shows the controller acting autonomously for the pick and place and pushing to pose tasks in the virtual and physical environments respectively. A video of the same experiments is available online<sup>2</sup>.

We have found that indeed, in most cases, the physical robot had been successful on executing both tasks. This is not because the virtual and physical worlds are highly similar. The size of the gripper of the Baxter robot is different from the one in the virtual world. The friction coefficients are very different, and the physics simulation in the virtual world is also of limited accuracy. Even the size and shape of the box used in the physical experiments is not an exact match of the ones in the simulation, and the physical setup suffered from

camera calibration problems. Overall, the number of things that can go wrong, is much higher in the physical world.

What we found remarkable is that, in fact, during the experiments many things went wrong or changed from the training data - however, the robot was often able to recover from them. This shows how the model is robust in handling deviation from what it has seen. The network contains different solutions for a case that can apply if others fail. For instance, in the pushing task, to rotate the box, it tries to touch the corner of the box and push it. If the box did not move since the gripper passed it without a touch, it tries again but this time from a point closer to the center of the box. This gives the network some tolerance to slight variations in the size of the box from the simulation to the real world.

After verifying that the successful completion of the task in the physical world is possible, we ran the same series of 20 experiments in the physical world as well. The success rates in the virtual and physical worlds are compared as follows:

Environment	Pick and place	Push to pose
Virtual world	100%	95%
Physical world	80%	60%

As expected, the success rate was lower in the physical world for both tasks. Some of the reasons behind the lower success rate is obvious: for instance, in the physical world there is an inevitable noise in the position of the objects and the end effector. Some of the noise is a consequence of limited sensor accuracy (such as the calibration of the Kinect sensor) and effector performance. Some of the noise is due to the way in which we acquired the positional information through a Kinect sensor: if during the manipulation the robot arm occluded the view of the object to the Kinect sensor, we temporarily lost the ability to track the object.

Another reason for the lower performance in the physical world is due to the differences in the size, shape, physical attributes such as friction, etc. of the gripper and objects between the simulation and real world. For instance, for the to push to pose task, the friction between the object and the table determines the way the object moves when pushed. This creates a bigger difference between the virtual and the physical environment compared to the pick and place task, where after a successful grasp the robot is essentially in control of the environment. Thus, the push to pose task shows a stronger decrease in success rate when moving to the physical world.

#### V. CONCLUSIONS AND FUTURE WORK

In this paper we have developed a technique through which a robotic arm can be taught to perform certain manipulation tasks. We focused on two tasks that are frequently required of robots that assist disabled users in activities of daily living: pick and place and push to desired pose. As disabled users can not generate large numbers of demonstrations on physical robots, we designed an approach where the user demonstrates the task in a virtual environment. These virtual demonstrations are used to teach a deep neural network based robot controller. Then, the controller is transferred to the physical robot. We found that the best performance was

<sup>2</sup><https://youtu.be/9vY1IG2ozaM>

obtained using a mixture density network containing 3 LSTM layers. This network reached higher than 90% success rate in the virtual environment. In the physical environment, it succeeded 80% of the time for the pick and place task, and 60% of the time for the push to pose task.

Our team is working to improve these results along several directions. First, the success rates demonstrate the validity of the method, but they fall short from practical deployability. There are several directions that can improve the success rate: improving the fidelity of the virtual environment, improving the sensor accuracy of the robot (for instance, by handling occlusions), tweaking the network structure and finding a way to acquire more demonstrations. The latter requirement is especially challenging, because at the same time we want to also *reduce* the number of demonstrations a user needs to make for a specific task. We also plan to extend the approach to more difficult and complex manipulation tasks, including full execution of activities of daily living tasks.

In our current approach we trained one neural network per task. We are currently working towards multi-task learning, an approach that, we hope, can reduce the number of demonstrations required by extracting common features among related tasks.

Finally, our current architecture trains a robot controller that takes as input the poses of the robot manipulator and the objects in the environment. These numerical values need to be obtained from sensors that are not integrated with the controller. We are working towards extending our neural network to also perform computer vision on raw pixels from cameras attached to the robot.

**Acknowledgments:** We would like to gratefully thank Sergey Levine, Navid Kardan, and Amirhossein Jabalameli for their helpful comments and suggestions. This work had been supported by the National Science Foundation under Grant Number IIS-1409823.

## REFERENCES

- [1] F. Endres, J. Trinkle, and W. Burgard, “Learning the dynamics of doors for robotic manipulation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3543–3549, 2013.
- [2] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, “A geometric approach to robotic laundry folding,” *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 249–267, 2012.
- [3] M. Bollini, J. Barry, and D. Rus, “Bakebot: Baking cookies with the PR2,” in *IROS PR2 workshop: results, challenges and lessons learned in advancing robots with a common platform*, 2011.
- [4] A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.
- [5] A. Karpathy, J. Johnson, and F.-F. Li, “Visualizing and understanding recurrent networks,” *arXiv preprint arXiv:1506.02078*, 2015.
- [6] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems (NIPS)*, pp. 3104–3112, 2014.
- [7] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649, 2013.
- [8] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, “Long-term recurrent convolutional networks for visual recognition and description,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [9] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [11] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *arXiv preprint arXiv:1409.1259*, 2014.
- [12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.
- [13] P. Abbeel, A. Coates, and A. Y. Ng, “Autonomous helicopter aerobatics through apprenticeship learning,” *International Journal of Robotics Research (IJRR)*, 2010.
- [14] J. Kober, E. Oztop, and J. Peters, “Reinforcement learning to adjust robot movements to new situations,” in *2010 Robotics: Science and Systems Conference (RSS 2010)*, pp. 33–40, MIT Press, 2011.
- [15] S. Calinon, F. D’halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, “Learning and reproduction of gestures by imitation,” *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 44–54, 2010.
- [16] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, “Learning and generalization of motor skills by learning from demonstration,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 763–768, 2009.
- [17] J. Sung, S. H. Jin, and A. Saxena, “Robobarista: Object part-based transfer of manipulation trajectories from crowd-sourcing in 3d point-clouds,” in *International Symposium on Robotics Research (ISRR)*, 2015.
- [18] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, “Cloud-based robot grasping with the Google object recognition engine,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4263–4270, 2013.
- [19] M. Forbes, M. J.-Y. Chung, M. Cakmak, and R. P. Rao, “Robot programming by demonstration with crowdsourced action fixes,” in *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.
- [20] C. Crick, S. Osentoski, G. Jay, and O. C. Jenkins, “Human and robot perception in large-scale learning from demonstration,” in *International conference on Human-robot interaction*, pp. 339–346, ACM, 2011.
- [21] S. Calinon, F. Guenter, and A. Billard, “On learning, representing, and generalizing a task in a humanoid robot,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 37, no. 2, pp. 286–298, 2007.
- [22] S. Calinon, F. D’halluin, D. G. Caldwell, and A. Billard, “Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework,” in *IEEE International Conference on Humanoid Robots (Humanoids)*, pp. 582–588, Citeseer, 2009.
- [23] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.
- [24] L. Pinto and A. Gupta, “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 763–768, 2016.
- [25] S. Ross, G. J. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proc. of International Conference on Artificial Intelligence and Statistics (AISTATS) - Volume 1*, no. 2, p. 6, 2011.
- [26] C. M. Bishop, “Mixture density networks,” *Technical Report*, 1994.
- [27] G. J. McLachlan and K. E. Basford, “Mixture models. inference and applications to clustering,” *Statistics: Textbooks and Monographs, New York: Dekker*, vol. 1, 1988.
- [28] B. Van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio, “Blocks and fuel: Frameworks for deep learning,” *arXiv preprint arXiv:1506.00619*, 2015.
- [29] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.02688, May 2016.
- [30] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural Networks for Machine Learning*, vol. 4, 2012.