Deep Multimodal Embedding: Manipulating Novel Objects with Point-clouds, Language and Trajectories

Jaeyong Sung, Ian Lenz, and Ashutosh Saxena

***Abstract*— A robot operating in a real-world environment needs to perform reasoning with a variety of sensing modalities. However, manually designing features that allow a learning algorithm to relate these different modalities can be extremely challenging. In this work, we consider the task of manipulating novel objects and appliances. To this end, we learn to embed point-cloud, natural language, and manipulation trajectory data into a shared embedding space using a deep neural network. In order to learn semantically meaningful spaces throughout our network, we introduce a method for pre-training its lower layers for multimodal feature embedding and a method for fine-tuning this embedding space using a loss-based margin. We test our model on the Robobarista dataset [22], where we achieve significant improvements in both accuracy and inference time over the previous state of the art.**

I. Introduction

Consider a robot manipulating a new appliance in a home kitchen in Figure 2. Even though the robot may never have seen this object before, it should be able to use an instruction manual in natural language along with its observations of the object to infer how the object should be manipulated. This ability to fuse information from different input modalities and map them to actions is extremely useful to a household robot.

If the robot has prior experience with similar object parts and instructions, it should understand how these correspond to actions [22], and be able to extend this to new environments. For example, it should understand that all "rotate clockwise" operations on vertical knobs attached to a vertical surface map to similar manipulation motions.

Even though similar concepts might appear very differently in different sensor modalities, humans are able to understand that they map to the same concept. For example, we are able to correlate the appearance with feel of a banana, or a natural language instruction with a real-world action. There is a strong evidence that we do so through a *common representation* between different modalities [3].

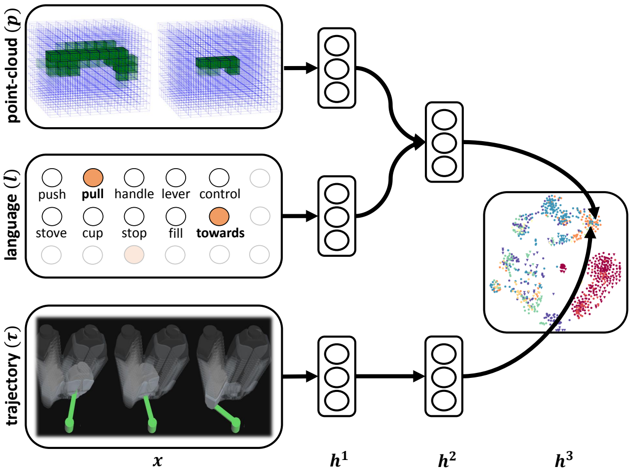

Fig. 1. **Deep Multimodal Embedding:** Our deep neural network learns to embed both point-cloud/natural language instruction combinations and manipulation trajectories in the same space. This allows for fast selection of a new trajectory by projecting a new environment/instruction pair and choosing its nearest-neighbor trajectory in this space.

Obtaining a good common representation between different modalities is challenging for two main reasons. First, each modality might intrinsically have very different statistical properties – for example, most trajectory representations are inherently dense, while a bag-of-words representation of language is inherently sparse. This makes it challenging to apply algorithms designed for unimodal data, as one modality might overpower the others. Second, even with expert knowledge, it is extremely challenging to design joint features between such disparate modalities [22]. Designing features which map different sensor inputs and actions to the same space, as required here, is particularly challenging.

In this work, we use a deep neural network to learn a shared embedding between the combination of object parts in the environment and natural language instructions, and manipulation trajectories (Figure 1). This means that all three modalities are projected to the *same* feature space. We introduce an algorithm that learns to pull semantically similar environment/language pairs and their corresponding trajectories to the same regions, and push environment/language pairs away from irrelevant trajectories based on how irrelevant these trajectories are. Our algorithm allows for efficient inference because, given a new instruction and point-cloud, we need only find the nearest trajectory to the projection of this pair in the learned embedding space, which can be done using fast nearest-neighbor algorithms [15].

In the past, deep learning methods have shown impressive results for learning features for a wide variety of domains [8, 18, 5] and even learning cross-domain embeddings for, for example, language and image features [20]. In contrast to these existing methods, here we present a new pre-training algorithm for initializing networks to be used for joint embedding of different modalities. This algorithm trains each layer to map similar cases to similar areas of its feature

Authors are with Department of Computer Science, Cornell University. Email: {jysung,ianlenz,asaxena}@cs.cornell.edu Jaeyong Sung is also a visiting student at Stanford University. Ashutosh Saxena is also with Brain of Things, Inc.

space, as opposed to other methods which either perform variational learning [6] or train for reconstruction [4].

In order to validate our approach, we test our model on a large manipulation dataset, the Robobarista Dataset [22], which focuses on learning to infer manipulation trajectories for *novel* objects. In summary, the key contributions of this work are:

- We present a novel approach to manipulation planning via joint embedding of multimodal data with deep neural networks.
- We present an algorithm which learns an embedding space while enforcing a varying and loss-based margin which pushes less-relevant cases further away.
- We present an algorithm for unsupervised pre-training of features to be used for embedding which outperforms standard pre-training algorithms [4].
- Our approach allows for fast inference of manipulation trajectories for novel objects, roughly 170x faster than previous work [22] while also improving accuracy.

II. Related Work

A. Robotic Manipulation

Most works in robotic manipulation focus on task-specific manipulation of *known* objects—for example, folding towels [12], or baking cookies with known tools [2]. Others [21, 13] focus on sequencing manipulation tasks assuming perfect manipulation primitives such as *pour* are available. Instead, here, we use learning to generalize to manipulating *novel* objects never seen before by the robot.

A few recent works use deep learning approaches for robotic manipulation. Levine et al. [10] use a Gaussian mixture model to learn system dynamics, then use these to learn a manipulation policy using a deep network. Lenz et al. [9] use a deep network to learn system dynamics for real-time model-predictive control. Both these works focus on learning low-level controllers, whereas here we learn high-level manipulation trajectories.

Our previous work [22] introduces the idea of object part-based transfer of manipulation trajectories for novel objects. In Section V-E, we show that our approach gives better accuracy than this prior work, while also running 171x faster.

B. Metric Embedding

Several works in machine learning make use of the power of shared embedding spaces. LMNN [26] learns a max-margin Mahalanobis distance for a unimodal input feature space. Weston et al. [27] learn linear mappings from image and language features to a common embedding space for automatic image annotation. Moore et al. [14] learn to map songs and natural language tags to a shared embedding space. However, these approaches learn only a shallow, linear mapping from input features, whereas here we learn a deep non-linear mapping which is less sensitive to input representations.

C. Deep Learning

Multimodal data: Ngiam et al. [16] use deep learning to learn features incorporating both video and audio modalities. Sohn et al. [19] propose a new generative learning algorithm for multimodal data which improves robustness to missing modalities at inference time. In these works, similar to our previous work [22], a single network takes all modalities as inputs, whereas here we perform joint embedding of multiple modalities using multiple networks.

Joint embedding: Several previous works use deep networks for joint embedding between different feature spaces. Mikolov et al. [11] map different languages to a joint feature space for translation. Srivastava and Salakhutdinov [20] map images and natural language "tags" to the same space for automatic annotation and retrieval. While these works use conventional pre-training algorithms, here we present a new pre-training approach for learning embedding spaces and show that it outperforms these existing methods (Sec. V-E.)

Hu et al. [7] also use a deep network for metric learning for the task of face verification. Similar to LMNN [26], Hu et al. [7] enforces a constant margin between distances among inter-class objects and among intra-class objects. In Sec. V-E, we show that our approach, which uses a loss-dependent variable margin, produces better results for our problem.

III. Overview

Our goal is to build an algorithm that allows a robot to infer a manipulation trajectory when it is introduced to a new appliance and its natural language instruction manual. From our prior work [22], we build on the key idea of *object part-based transfer* – i.e. that many completely different objects share parts that are operated similarly. For example, the motion required to operate the handle of the espresso machine in Figure 2 is almost identical to the motion required to flush a urinal with a handle. By identifying and transferring trajectories from prior experience with parts of other objects, robots can manipulate even objects they have never seen before.

As shown in Figure 2, given a point-cloud for each part of an espresso machine and a natural language instruction such as 'Push down on the handle to add hot water', our algorithm outputs a trajectory which executes the task, using a pool of prior motion experience. This trajectory includes how to approach, grasp, and press the handle.

This is a challenging problem because the object is entirely new to the robot, and because it must jointly consider the point-cloud, natural language instruction, and each potential trajectory. Moreover, manually designing useful features from these three modalities is extremely challenging [22].

A. Our Approach

In this work, we learn a joint embedding of point-cloud, language, and trajectory data into the same low dimensional space. We learn non-linear embeddings using a deep learning approach which maps raw data from these three different modalities to a joint embedding space.

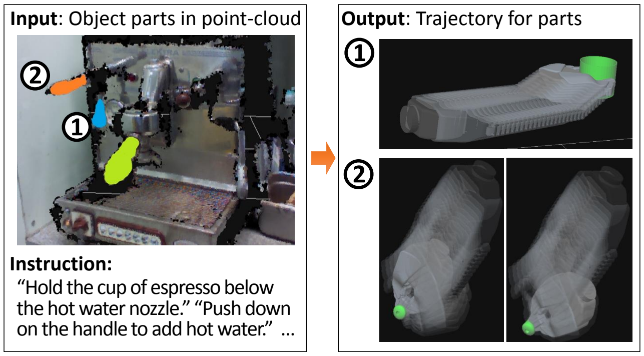
Input: Object parts in point-cloud
②
①
Instruction:
"Hold the cup of espresso below the hot water nozzle." "Push down on the handle to add hot water." ...
Output: Trajectory for parts
①
②

Fig. 2. **Inferring trajectories:** Given a point-cloud and a language instruction, our goal is to output an appropriate trajectory that would manipulate the object according to the instruction.

We then use this space to find known trajectories which are good matches for new combinations of object parts and instructions. Compared to previous work that exhaustively runs a full network over all these combinations [22], our approach allows us to pre-embed all candidate trajectories into this common feature space. Thus, the most appropriate trajectory can be identified by embedding only a new point-cloud/language pair and then finding its nearest neighbor.

In our joint feature space, proximity between two mapped points should reflect how relevant two data-points are to each other, even if they are from completely different modalities. We train our network to bring demonstrations that would manipulate a given object according to some language instruction closer to the mapped point for that object/instruction pair, and to push away demonstrations that would not correctly manipulate the object. Trajectories which have no semantic relevance to the object are pushed much further away than trajectories that have some relevance, even if the latter would not manipulate the object according to the instruction.

Prior to learning a full joint embedding of all three modalities, we pre-train embeddings of subsets of the modalities to learn semantically meaningful embeddings for these modalities, leading to improved performance as shown in Section V-E.

B. Problem Formulation

Given a point-cloud $p \in \mathcal{P}$ and a free-form natural language $l \in \mathcal{L}$, our goal is to output a manipulation trajectory $\tau \in \mathcal{T}$:

$$f : \mathcal{P} \times \mathcal{L} \rightarrow \mathcal{T} \tag{1}$$

The manipulation trajectory τ represents a trajectory that manipulates the object p according to the natural language instruction l. This approach to manipulation planning is defined the same as in Sung et al. [22]. We also take the same strategy of finding an appropriate trajectory by transferring trajectories from prior experience.

To solve this problem, in this work, our goal is to learn two different mapping functions that map to a common space—one from a point-cloud/language pair and the other from a trajectory. More formally, we want to learn $\Phi_{\mathcal{P},\mathcal{L}}(p,l)$ and $\Phi_{\mathcal{T}}(\tau)$ which map to a joint feature space $\mathbb{R}^M$:

$$\Phi_{\mathcal{P},\mathcal{L}}(p,l) : (\mathcal{P},\mathcal{L}) \rightarrow \mathbb{R}^M$$
$$\Phi_{\mathcal{T}}(\tau) : \mathcal{T} \rightarrow \mathbb{R}^M$$

Here, we represent these mappings with a deep neural network, as shown in Figure 1.

The first, $\Phi_{\mathcal{P},\mathcal{L}}$, which maps point-clouds and trajectories, is defined as a combination of two mappings. The first of these maps to a joint point-cloud/language space $\mathbb{R}^{N_{2,pl}}$ — $\Phi_{\mathcal{P}}(p) : \mathcal{P} \rightarrow \mathbb{R}^{N_{2,pl}}$ and $\Phi_{\mathcal{L}}(l) : \mathcal{L} \rightarrow \mathbb{R}^{N_{2,pl}}$. Once each is mapped to $\mathbb{R}^{N_{2,pl}}$, this space is then mapped to the joint space shared with trajectory information: $\Phi_{\mathcal{P},\mathcal{L}}(p,l) : ((\mathcal{P},\mathcal{L}) \rightarrow \mathbb{R}^{N_{2,pl}}) \rightarrow \mathbb{R}^M$.

C. Model

We use two separate multi-layer deep neural networks, one for $\Phi_{\mathcal{P},\mathcal{L}}(p,l)$ and one for $\Phi_{\mathcal{T}}(\tau)$. Take N_p as the size of point-cloud input p, N_l as similar for natural language input l, $N_{1,p}$ and $N_{1,l}$ as the number of hidden units in the first hidden layers projected from point-cloud and natural language features, respectively, and $N_{2,pl}$ as the number of hidden units in the combined point-cloud/language layer. With W's as network weights, which are the learned parameters of our system, and $a(\cdot)$ as a rectified linear unit (ReLU) activation function [29], our model for projecting from point-cloud and language features to the shared embedding h^3 is as follows:

$$h_i^{1,p} = a\left(\sum_{j=0}^{N_p} W_{i,j}^{1,p} p_j\right)$$
$$h_i^{1,l} = a\left(\sum_{j=0}^{N_l} W_{i,j}^{1,l} l_j\right)$$
$$h_i^{2,pl} = a\left(\sum_{j=0}^{N_{1,p}} W_{i,j}^{2,p} h_j^{1,p} + \sum_{j=0}^{N_{1,l}} W_{i,j}^{2,l} h_j^{1,l}\right)$$
$$h_i^3 = a\left(\sum_{j=0}^{N_{2,pl}} W_{i,j}^{3,pl} h_j^{2,pl}\right)$$

The model for projecting from trajectory input τ is similar, except it takes input only from a single modality.

D. Inference.

Once all mappings are learned, we solve the original problem from equation (1) by choosing, from a library of prior trajectories, the trajectory that gives the highest similarity (closest in distance) to the given point-cloud p and language l in our joint embedding space $\mathbb{R}^M$. As in previous work [27], similarity is defined as $sim(a,b) = a \cdot b$, and the trajectory is selected as the trajectory that maximizes the magnitude of similarity: $\text{argmax}_{\tau \in \mathcal{T}}\, sim(\Phi_{\mathcal{P},\mathcal{L}}(p,l), \Phi_{\mathcal{T}}(\tau))$.

The previous approach to this problem [22] required projecting the combination of the current point-cloud and natural language instruction with *every* trajectory in the training set through the network during inference. Here, we pre-compute the representations of all training trajectories in h^3, and need only project the new point-cloud/language pair to h^3 and find its nearest-neighbor trajectory in this embedding space. As shown in Section V-E, this significantly improves both the runtime and accuracy of our approach and makes it much more scalable to larger training datasets like those collected with crowdsourcing platforms [22].

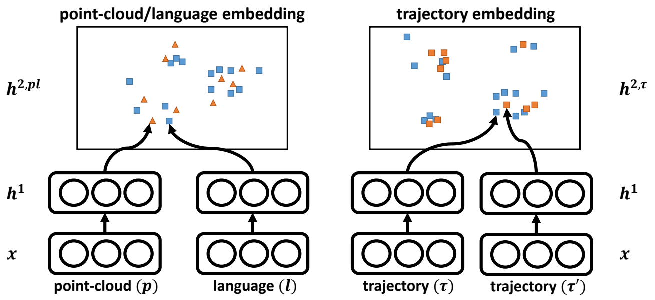

Fig. 3. **Pre-training lower layers:** Visualization of our pre-training approaches for $h^{2,pl}$ and $h^{2,\tau}$. For $h^{2,pl}$, our algorithm pushes matching point-clouds and instructions to be more similar. For $h^{2,\tau}$, our algorithm pushes trajectories with higher DTW-MT similarity to be more similar.

IV. Learning Joint Point-cloud/Language/Trajectory Model

The main challenge of our work is to learn a model which maps three disparate modalities – point-clouds, natural language, and trajectories – to a single semantically meaningful space. We introduce a method that learns a common point-cloud/language/trajectory space such that all trajectories relevant to a given task (point-cloud/language combination) should have higher similarity to the projection of that task than task-irrelevant trajectories. Among these irrelevant trajectories, some might be less relevant than others, and thus should be pushed further away.

For example, given a door knob that needs to be grasped normal to the door surface and an instruction to rotate it clockwise, a trajectory that correctly approaches the door knob but rotates counter-clockwise should have higher similarity to the task than one which approaches the knob from a completely incorrect angle and does not execute any rotation.

For every training point-cloud/language pair (p_i, l_i), we have a set of demonstrations $\mathcal{T}_i$ and the most optimal demonstration trajectory $\tau_i^* \in \mathcal{T}_i$. Using the optimal demonstration and a loss function $\Delta(\tau, \bar{\tau})$ for comparing demonstrations, we find a set of trajectories $\mathcal{T}_{i,S}$ that are relevant (similar) to this task and a set of trajectories $\mathcal{T}_{i,D}$ that are irrelevant (dissimilar.) We use the DTW-MT distance function (described later in Sec. V-C) for our loss function $\Delta(\tau, \bar{\tau})$, but it could be replaced by any function that computes the loss of predicting $\bar{\tau}$ when τ is the correct demonstration. Using a strategy previously used for handling noise in crowd-sourced data [22], we can use thresholds (t_S, t_D) to generate two sets from the pool of trajectories:

$$\mathcal{T}_{i,S} = \{\tau \in \mathcal{T} | \Delta(\tau_i^*, \tau) < t_S\}$$
$$\mathcal{T}_{i,D} = \{\tau \in \mathcal{T} | \Delta(\tau_i^*, \tau) > t_D\}$$

For each pair of (p_i, l_i), we want all projections of $\tau_j \in \mathcal{T}_{i,S}$ to have higher similarity to the projection of (p_i, l_i) than $\tau_k \in \mathcal{T}_{i,D}$. A simple approach would be to train the network to distinguish these two sets by enforcing a finite distance (safety margin) between the similarities of these two sets [26], which can be written in the form of a constraint: $sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_j)) \geq 1 + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_k))$

Rather than simply being able to distinguish two sets, we want to learn semantically meaningful embedding spaces from different modalities. Recalling our earlier example where one incorrect trajectory for manipulating a door knob was much closer to correct than another, it is clear that our learning algorithm should drive some of incorrect trajectories to be more dissimilar than others. The difference between the similarities of τ_j and τ_k to the projected point-cloud/language pair (p_i, l_i) should be at least the loss $\Delta(\tau_j, \tau_k)$. This can be written as a form of a constraint:

$$\begin{aligned} &\forall \tau_j \in \mathcal{T}_{i,S}, \forall \tau_k \in \mathcal{T}_{i,D} \\ &sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_j)) \\ &\qquad\qquad \geq \Delta(\tau_j, \tau_k) + sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_k)) \end{aligned}$$

Intuitively, this forces trajectories with higher DTW-MT distance from the ground truth to embed further than those with lower distance. Enforcing all combinations of these constraints could grow exponentially large. Instead, similar to the cutting plane method for structural support vector machines [23], we find the most violating trajectory $\tau' \in \mathcal{T}_{i,D}$ for each training pair of $(p_i, l_i, \tau_i \in \mathcal{T}_{i,S})$ at each iteration. The most violating trajectory has the highest similarity augmented with the loss scaled by a constant α:

$$\tau_i' = \arg \max_{\tau \in \mathcal{T}_{i,D}} (sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau)) + \alpha\Delta(\tau_i, \tau))$$

The cost of our deep embedding space h^3 is computed as the hinge loss of the most violating trajectory.

$$\begin{aligned} L_{h^3}(p_i, l_i, \tau_i) = |\Delta(\tau_i', \tau_i) &+ sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i')) \\ &- sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i, l_i), \Phi_{\mathcal{T}}(\tau_i))|_+ \end{aligned}$$

The average cost of each minibatch is back-propagated through all the layers of the deep neural network using the AdaDelta [28] algorithm.

A. Pre-training Joint Point-cloud/Language Model

One major advantage of modern deep learning methods is the use of unsupervised pre-training to initialize neural network parameters to a good starting point before the final supervised fine-tuning stage. Pre-training helps these high-dimensional networks to avoid overfitting to the training data.

Our lower layers $h^{2,pl}$ and $h^{2,\tau}$ represent features extracted exclusively from the combination of point-clouds and language, and from trajectories, respectively. Our pre-training method initializes $h^{2,pl}$ and $h^{2,\tau}$ as semantically meaningful embedding spaces similar to h^3, as shown later in Section V-E.

First, we pre-train the layers leading up to these layers using spare de-noising autoencoders [25, 29]. Then, our process for pre-training $h^{2,pl}$ is similar to our approach to fine-tuning a semantically meaningful embedding space for h^3 presented above, except now we find the most violating language l' while still relying on a loss over the associated optimal trajectory:

$$l' = \underset{l \in \mathcal{L}}{\operatorname{argmax}} (sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l)) + \alpha\Delta(\tau, \tau_i^*))$$

$$\begin{aligned} L_{h^{2,pl}}(p_i, l_i, \tau_i) = |\Delta(\tau_i, \tau') &+ sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l')) \\ &- sim(\Phi_{\mathcal{P}}(p_i), \Phi_{\mathcal{L}}(l_i))|_+ \end{aligned}$$

Notice that although we are training this embedding space to project from point-cloud/language data, we guide learning using trajectory information.

After the projections $\Phi_{\mathcal{P}}$ and $\Phi_{\mathcal{L}}$ are tuned, the output of these two projections are added to form the output of layer $h^{2,pl}$ in the final feed-forward network.

B. Pre-training Trajectory Model

For our task of inferring manipulation trajectories for novel objects, it is especially important that similar trajectories τ map to similar regions in the feature space defined by $h^{2,\tau}$, so that trajectory embedding $h^{2,\tau}$ itself is semantically meaningful and they can in turn be mapped to similar regions in h^3. Standard pretraining methods, such as sparse denoising autoencoder [25, 29] would only pre-train $h^{2,\tau}$ to reconstruct individual trajectories. Instead, we employ pretraining similar to Sec. IV-A, except now we pre-train for only a single modality – trajectory data.

As shown on right hand side of Fig. 3, the layer that embeds to $h^{2,\tau}$ is duplicated. These duplicated embedding layers are treated as if they were two different modalities, but all their weights are shared and updated simultaneously. For every trajectory $\tau \in \mathcal{T}_{i,S}$, we can again find the most violating $\tau' \in \mathcal{T}_{i,D}$ and the minimize a similar cost function as we do for $h^{2,pl}$.

V. Experiments

A. Dataset

We test our model on the Robobarista dataset [22]. This dataset consists of 249 segmented point-clouds, 250 free-form natural language instructions, and 1225 crowd-sourced manipulation trajectories which are demonstrated for 250 point-cloud/language pairs. The point-clouds were collected by stitching multiple views using Kinect Fusion. The manipulation trajectories were collected from 71 non-experts on Amazon Mechanical Turk through the Robobarista crowd-sourcing platform. This platform allows non-experts to demonstrate manipulation trajectories in their own web browser without the presence of an expert, the robot, or the target object.

Preprocessing: We utilize the same preprocessing of point-cloud, language, and trajectory data as Sung et al. [22] to fit raw data from each modality into a fixed-length vector. Each point-cloud is converted into two occupancy grid-like structures of size $10 \times 10 \times 10$ voxels, with each voxel spanning a cube of $1 \times 1 \times 1$ cm and $2.5 \times 2.5 \times 2.5$ cm, respectively. Natural language instructions are represented using a bag-of-words model. Finally, trajectories, which are given as sequence of waypoints including gripper actions such as 'opening', 'closing', and 'holding', are converted to a smooth trajectory through linear interpolation of translations and spherical interpolation of rotational quaternions. Then, each smoothed trajectory is normalized to the same length while preserving the gripper actions.

B. Evaluation

For each point-cloud/language pair in each fold's test set, each algorithm selects one trajectory from the training set which best suits this pair. For evaluation purposes, the dataset contains a separate expert demonstration for each point-cloud/language pair, which is not included in the training phase [22]. Every transferred trajectory is evaluated against these expert demonstrations. All algorithms are evaluated using *five-fold cross-validation*, with 10% of the data kept out as a validation set.

C. Metrics

The main measure we use is dynamic time warping for manipulation trajectories (DTW-MT) [22]. This metric compares trajectories by non-linearly warping two trajectories of different lengths while preserving weak ordering of matched trajectory waypoints. Since the DTW-MT measure itself is not intuitive, Sung et al. [22] also reports the percentage of transferred trajectories that have a DTW-MT measure of less than 10 from the ground-truth trajectory. From an expert survey, a DTW-MT measure of less than 10 indicates that a trajectory will most likely correctly manipulate the given object according to the given instruction. We also plot the accuracy per varying threshold.

As our main evaluation, we report three metrics – DTW-MT per manual, DTW-MT per instruction, and Accuracy (DTW-MT < 10) per instruction. Instruction here refers to every point-cloud/language pair, and manual refers to list of instructions which comprises a set of sequential tasks, which we average over.

D. Baselines

We compare our model against several baselines:

1) *Random Transfers (chance).* Trajectories are randomly transferred to the new object.
2) *Sung et al. [22].* State-of-the-art result on this dataset. Authors train a deep neural network to predict how likely each known trajectory matches the given point-cloud and language.
   We also report several baselines from this work which rely on more traditional approaches such as classifying point-clouds into labels like 'handle', 'lever' and so on (*object part classifier*), or hand-designing features for multi-modal data (*LSSVM + kinematic structure* and *task similarity + weighting*).
3) *LMNN [26]-like cost function.* For all top layer fine-tuning and lower layer pre-training, we define the cost function without loss augmentation. Similar to LMNN [26], we give a finite margin between similarities. For example, as cost function for h^3:
$$L_{h^3}(p_i,l_i,\tau_i) = |1+sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i,l_i),\Phi_{\mathcal{T}}(\tau')) \\ -sim(\Phi_{\mathcal{P},\mathcal{L}}(p_i,l_i),\Phi_{\mathcal{T}}(\tau_i))|_+$$
4) *Our Model without Pretraining.* Our full model fine-tuned without any pre-training of lower layers.
5) *Our Model with SDA.* Our full model without pre-training $h^{2,pl}$ and $h^{2,\tau}$ as defined in Sections IV-A

TABLE I

Result on our dataset with *5-fold cross-validation*. Rows list models we tested including our model and baselines. Each column shows a different metric used to evaluate the models. For the DTW-MT metric, lower values are better. For accuracy, higher is better.

	per manual	**per instruction**	
Models	**DTW-MT**	**DTW-MT**	**Accu. (%)**
chance	28.0 (±0.8)	27.8 (±0.6)	11.2 (±1.0)
object part classifier [22]	-	22.9 (±2.2)	23.3 (±5.1)
LSSVM + kinematic [22]	17.4 (±0.9)	17.5 (±1.6)	40.8 (±2.5)
similarity + weights [22]	13.3 (±1.2)	12.5 (±1.2)	53.7 (±5.8)
Sung et al. [22]	13.0 (±1.3)	12.2 (±1.1)	60.0 (±5.1)
LMNN-like cost func. [26]	15.4 (±1.8)	14.7 (±1.6)	55.5 (±5.3)
without any pretraining	13.2 (±1.4)	12.4 (±1.0)	54.2 (±6.0)
Our Model with SDA	11.5 (±0.6)	11.1 (±0.6)	62.6 (±5.8)
Our Model	**11.0** (±0.8)	**10.5** (±0.7)	**65.1** (±4.9)

and IV-B. Instead, we pre-train each layer as stacked de-noising autoencoders [25, 29].

E. Results

We present the results of our algorithm and the baseline approaches in Table I. Additionally, Fig. 4 shows accuracies obtained by varying the threshold on the DTW-MT measure.

The state-of-the-art result on this dataset has a DTW-MT measure of 13.0 per manual and a DTW-MT measure and accuracy of 12.2 and 60.0% per instruction [22]. Our full model based on joint embedding of multimodal data achieved 11.0, 10.5, and 65.1% respectively. This means that when a robot encounters a *new* object it has never seen before, our model gives a trajectory which would correctly manipulate it according to a given instruction approximately 65.1% of the time. From Fig. 4, we can see that our algorithm consistently outperforms both our prior work and an LMNN-like cost function for all thresholds on the DTW-MT metric.

What does learned deep embedding space represent? Figure 5 shows a visualization of the top layer h^3, the joint embedding space. This visualization is created by projecting all training data (point-cloud/language pairs and trajectories) of one of the cross-validation folds to h^3, then embedding them to 2-dimensional space using t-SNE [24]. Although previous work [22] was able to visualize several nodes in the top layer, most were difficult to interpret. With our model, we can embed all our data and visualize all the layers (see Figs. 5 and 6).

One interesting result is that our system was able to naturally learn that "nozzle" and "spout" are effectively synonyms for purposes of manipulation. It clustered these together in the lower-right of Fig. 5 based solely on the fact that both are associated with similar point-cloud shapes and manipulation trajectories. At the same time, it also identified one exception, a small cluster of "nozzles" in the center of Fig. 5 which require different manipulation motions.

In addition to the aforementioned cluster in the bottom-right of Fig. 5, we see several other logical clusters. Importantly, we can see that our embedding maps vertical and horizontal rotation operations to very different regions of the space – roughly 12 o'clock and 8 o'clock in Fig. 5, respectively. Even though these have nearly identical language instructions, our algorithm learns to map them differently based on their point-clouds, mapping nearby the appropriate manipulation trajectories.

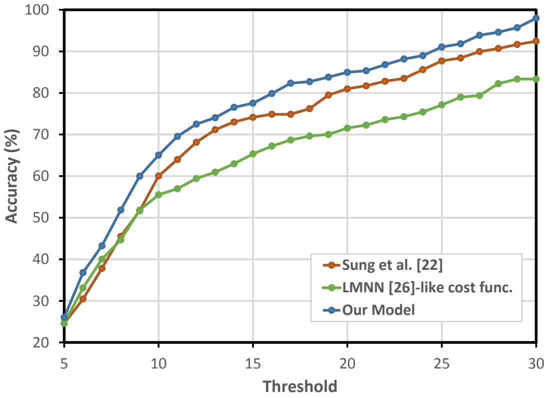

Fig. 4. **Thresholding Accuracy:** Accuracy-threshold graph showing results of varying thresholds on DTW-MT scores. Our algorithm consistently outperforms the previous approach [22] and an LMNN-like cost function [26].

Should cost function be loss-augmented? When we changed the cost function for pre-training h^2 and fine-tuning h^3 to use a constant margin of 1 between relevant $\mathcal{T}_{i,S}$ and irrelvant $\mathcal{T}_{i,D}$ demonstrations [26], performance drops to 52.0%. This loss-augmentation is also visible in our embedding space. Notice the purple cluster around the 6 o'clock region of Fig. 5, and the lower part of the cluster in the 5 o'clock region. The purple cluster represents tasks and demonstrations related to pushing a bar (often found on soda fountains), and the lower part of the red cluster represents the task of holding a cup below the nozzle. Although the motion required for one task would not be replaceable by the other, the motions and shapes are very similar, especially compared to most other motions e.g. turning a horizontal knob.

Is pre-embedding important? As seen in Table I, without any pre-training our model gives an accuracy of only 54.2%. Pre-training the lower layers with the conventional stacked de-noising auto-encoder (SDA) algorithm [25, 29] increases performance to 62.6%, still significantly underperforming our pre-training algorithm, which gives 65.1%. This shows that our metric embedding pre-training approach provides a better initialization for an embedding space than SDA.

Fig. 6 shows the joint point-cloud/language embedding $h^{2,pl}$ after the network is initialized using our pre-training algorithm and then fine-tuned using our cost function for h^3. While this space is not as clearly clustered as h^3 shown in Fig. 5, we note that point-clouds tend to appear in the more general center of the space, while natural language instructions appear around the more-specific edges. This makes sense because one point-cloud might afford many possible actions, while language instructions are much more specific.

Does embedding improve efficiency? The previous model [22] had 749,638 parameters to be learned, while our model has only 418,975 (and still gives better performance.)

The previous model had to compute joint point-

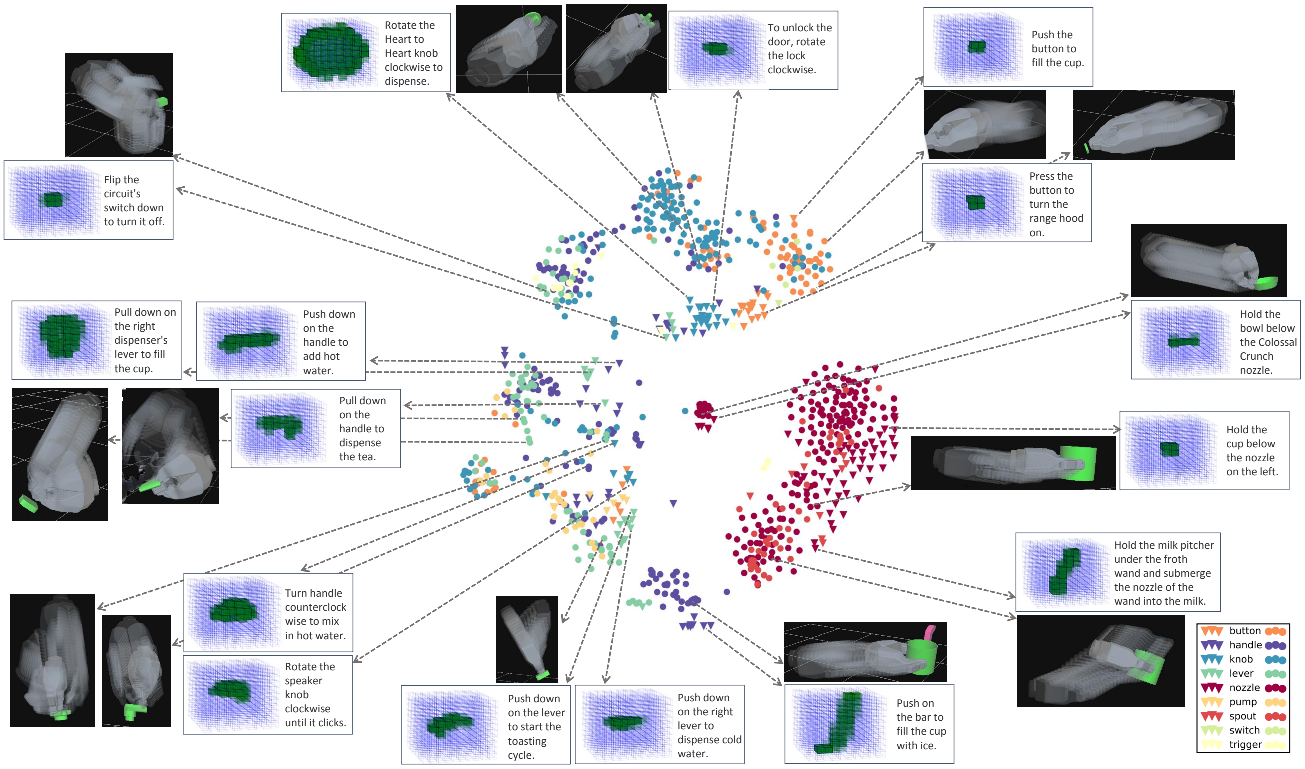

Fig. 5. **Learned Deep Point-cloud/Language/Trajectory Embedding Space:** Joint embedding space h^3 after the network is fully fine-tuned, visualized in 2d using t-SNE [24] . Inverted triangles represent projected point-cloud/language pairs, circles represent projected trajectories. The occupancy grid representation of object part point-clouds is shown in green in blue grids. Among the two occupancy grids (Sec. V-A), we selected the one that is more visually parsable for each object. The legend at the bottom right shows classifications of object parts by an expert, collected for the purpose of building a baseline. As shown by result of this baseline (object part classifier in Table I), these labels do not necessarily correlate well with the actual manipulation motion. Thus, full separation according to the labels defined in the legend is not optimal and will not occur in this figure or Fig. 6. These figures are best viewed in color.

cloud/language/trajectory features for all combinations of the current point-cloud/language pair with *each* candidate trajectory (i.e. all trajectories in the training set) to infer an optimal trajectory. This is inefficient and does not scale well with the number of training datapoints. However, our model pre-computes the projection of all trajectories into h^3. Inference in our model then requires only projecting the new point-cloud/language combination to h^3 once and finding the trajectory with maximal similarity in this embedding.

In practice, this results in a significant improvement in efficiency, decreasing the average time to infer a trajectory from 2.3206ms to 0.0135ms, a speed-up of about 171x. Time was measured on the same hardware, with a GPU (GeForce GTX Titan X), using the Theano library [1]. We measured inference times 10000 times for first test fold, which has a pool of 962 trajectories. Time to preprocess the data and time to load into GPU memory was not included in this measurement. We note that the only part of our algorithm's runtime which scales up with the amount of training data is the nearest-neighbor computation, for which there exist many efficient algorithms [15]. Thus, our algorithm could be scaled to much larger datasets, allowing it to handle a wider variety of tasks, environments, and objects.

VI. Conclusion

In this work, we approach the problem of inferring manipulation trajectories for novel objects by jointly embedding point-cloud, natural language, and trajectory data into a common space using a deep neural network. We introduce a method for learning a common representation of multimodal data with loss-augmented cost function, which produces semantically meaningful embedding. We also introduce a method for pre-training the network's lower layers, learning embeddings for subsets of modalities, and show that it outperforms standard pre-training algorithms. Learning such an embedding space allows efficient inference by comparing the embedding of a new point-cloud/language pair against pre-embedded demonstrations. We test our algorithm on the Robobarista Dataset [22] and show that our approach improves accuracy, despite having only half of the number of learned parameters and being much more computationally efficient (about 171x faster) than the state-of-the-art result. In future work, we plan to make the knowledge learned in the joint embedding space available to other robots using RoboBrain [17].

Acknowledgment

We thank Emin Gün Sirer for useful discussions. We thank NVIDIA Corporation for the donation of the Tesla K40 GPU used for this research. This work was supported by NRI award 1426452, ONR award N00014-14-1-0156, and by Microsoft Faculty Fellowship and NSF Career Award to one of us (Saxena).

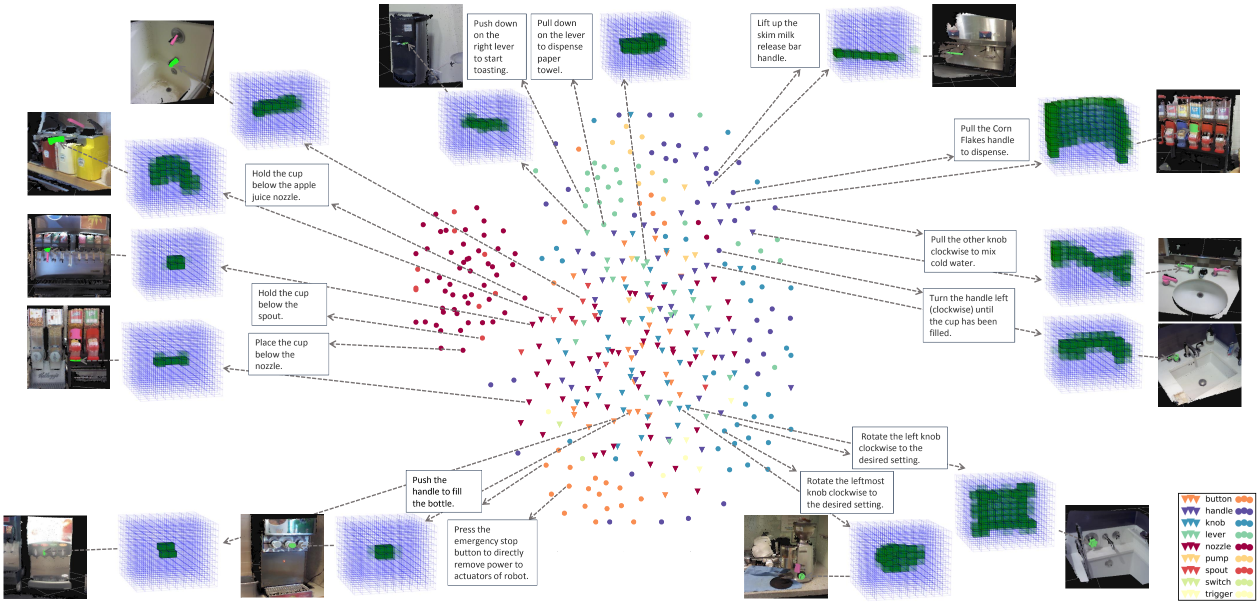

Fig. 6. **Learned Point-cloud/Language Space:** Visualization of the point-cloud/language layer $h^{2,lp}$ in 2d using t-SNE [24] after the network is fully fine-tuned. Inverted triangles represent projected point-clouds and circles represent projected instructions. A subset of the embedded points are randomly selected for visualization. Since 3D point-clouds of object parts are hard to visualize, we also include a snapshot of a point-cloud showing the whole object. Notice correlations in the motion required to manipulate the object or follow the instruction among nearby point-clouds and natural language.

References

[1] F. Bastien, P. Lamblin, R. Pascanu, et al. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] M. Bollini, J. Barry, and D. Rus. Bakebot: Baking cookies with the pr2. In *IROS PR2 Workshop*, 2011.

[3] G. Erdogan, I. Yildirim, and R. A. Jacobs. Transfer of object shape knowledge across visual and haptic modalities. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

[4] I. Goodfellow, Q. Le, A. Saxe, H. Lee, and A. Y. Ng. Measuring invariances in deep networks. In *NIPS*, 2009.

[5] R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *IROS*, pages 628–633. IEEE, 2008.

[6] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[7] J. Hu, J. Lu, and Y.-P. Tan. Discriminative deep metric learning for face verification in the wild. In *CVPR*, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[9] I. Lenz, R. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *RSS*, 2015.

[10] S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. *ICRA*, 2015.

[11] T. Mikolov, Q. V. Le, and I. Sutskever. Exploiting similarities among languages for machine translation. *CoRR*, 2013.

[12] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel. A geometric approach to robotic laundry folding. *IJRR*, 2012.

[13] D. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *RSS*, 2014.

[14] J. Moore, S. Chen, T. Joachims, and D. Turnbull. Learning to embed songs and tags for playlist prediction. In *Conference of the International Society for Music Information Retrieval (ISMIR)*, pages 349–354, 2012.

[15] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *PAMI*, 2014.

[16] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *ICML*, 2011.

[17] A. Saxena, A. Jain, O. Sener, A. Jami, D. K. Misra, and H. S. Koppula. Robo brain: Large-scale knowledge engine for robots. *Tech Report*, Aug 2014.

[18] R. Socher, J. Pennington, E. Huang, A. Ng, and C. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, 2011.

[19] K. Sohn, W. Shang, and H. Lee. Improved multimodal deep learning with variation of information. In *NIPS*, 2014.

[20] N. Srivastava and R. R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, 2012.

[21] J. Sung, B. Selman, and A. Saxena. Synthesizing manipulation sequences for under-specified tasks using unrolled markov random fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[22] J. Sung, S. H. Jin, and A. Saxena. Robobarista: Object part-based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *International Symposium on Robotics Research (ISRR)*, 2015.

[23] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, and Y. Singer. Large margin methods for structured and interdependent output variables. *JMLR*, 6(9), 2005.

[24] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605): 85, 2008.

[25] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

[26] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2005.

[27] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011.

[28] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[29] M. D. Zeiler, M. Ranzato, R. Monga, et al. On rectified linear units for speech processing. In *ICASSP*, 2013.