DeepMPC: Learning Deep Latent Features for Model Predictive Control

Ian Lenz, Ross Knepper, and Ashutosh Saxena
Department of Computer Science, Cornell University.
Email: {ianlenz, rak, asaxena}@cs.cornell.edu

***Abstract*—Designing controllers for tasks with complex non-linear dynamics is extremely challenging, time-consuming, and in many cases, infeasible. This difficulty is exacerbated in tasks such as robotic food-cutting, in which dynamics might vary both with environmental properties, such as material and tool class, and with time while acting. In this work, we present DeepMPC, an online real-time model-predictive control approach designed to handle such difficult tasks. Rather than hand-design a dynamics model for the task, our approach uses a novel deep architecture and learning algorithm, learning controllers for complex tasks directly from data. We validate our method in experiments on a large-scale dataset of 1488 material cuts for 20 diverse classes, and in 450 real-world robotic experiments, demonstrating significant improvement over several other approaches.**

I. Introduction

Most real-world tasks involve interactions with complex, non-linear dynamics. Although practiced humans are able to control these interactions intuitively, developing robotic controllers for them is very difficult. Several common household activities fall into this category, including scrubbing surfaces, folding clothes, interacting with appliances, and cutting food. Other applications include surgery, assembly, and locomotion. These interactions are characterized by hard-to-model effects, involving friction, deformation, and hysteresis. The compound interaction of materials, tools, environments, and manipulators further alters these effects. Consequently, the design of controllers for such tasks is highly challenging.

In recent years, "feed-forward" model-predictive control (MPC) has proven effective for many complex tasks, including quad-rotor control [36], mobile robot maneuvering [20], full-body control of humanoid robots [14], and many others [26, 18, 11]. The key insight of MPC is that an accurate predictive model allows us to optimize control inputs for some cost over both inputs and predicted *future* outputs. Such a cost function is often easier and more intuitive to design than completely hand-designing a controller. The chief difficulty in MPC lies instead in designing an accurate dynamics model.

Let us consider the dynamics involved in cutting food items, as shown in Fig. 1 for the wide range of materials shown in Fig. 2. An effective cutting strategy depends heavily on properties of the food, including its coefficient of friction with the knife, elastic modulus, fracture effects, and hysteretic effects such as plastic deformation [29]. These parameters lead humans to such diverse cutting strategies as slicing, sawing, and chopping. In fact, they can even vary within a single material; compare cutting through the skin of a lemon to cutting its flesh. Thus, a major challenge of this work is to design a model which can estimate and make use of global environmental properties such as the material and tool in question and temporally-changing properties such as the current rate of motion, depth of cutting, enclosure of the knife by the material, and layer of the material the knife is in contact with. While some works [15] attempt to define these properties, it is very difficult to design a set that truly captures all these complex inter- and intra-material variations.

Fig. 1: **Cutting food:** Our PR2 robot uses our algorithms to perform complex, precise food-cutting operations. Given the large variety of material properties, it is challenging to design appropriate controllers.

Hand-designing features and models for such tasks is infeasible and time-consuming, so here, we take a learning approach. In the recent past, feature learning methods have proven effective for learning latent task-specific features across many domains [3, 19, 24, 9, 28]. In this paper, we give a novel deep architecture for physical prediction for complex tasks such as food cutting. When this model is used for predictive control, it yields a DeepMPC controller which is able to learn task-specific controls. Deep learning is an excellent choice as a model for real-time MPC because its learned models are easily and efficiently differentiable with respect to their inputs using the same back-propagation algorithms used in learning, and because network sizes can simply be adjusted to trade off between prediction accuracy and computational speed.

Our model, optimized for receding-horizon prediction, learns latent material properties directly from data. Our architecture uses multiplicative conditional interactions and temporal recurrence to model both inter-material and time-dependent intra-material variations. We also present a novel learning

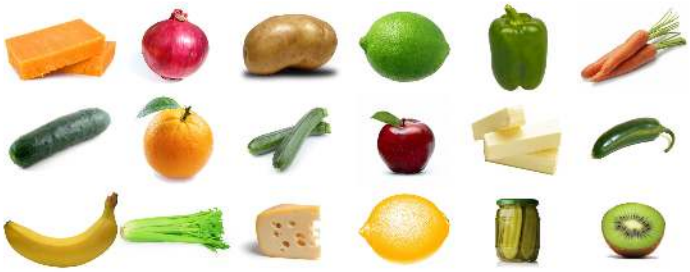

Fig. 2: **Food materials:** Some of the 20 diverse food materials which make up our material interaction dataset.

algorithm for this recurrent network which avoids overfitting and the "exploding gradient" problem commonly seen when training recurrent networks. Once learned, inference for our model is extremely fast - when predicting to a time-horizon of 1s (100 samples) in the future, our model and its gradients can be evaluated at a rate of 1.2kHz.

In extensive experiments on our large-scale dataset comprising 1488 examples of robotic cutting across 20 different material types, we demonstrate that our feature-learning approach outperforms other state-of-the-art methods for physical prediction. We also implement an online real-time model-predictive controller using our model. In a series of over 450 real-world robotic trials, we show that our controller gives extremely strong performance for robotic food-cutting, even compared to methods tuned for specific material classes.

In summary, the contributions of this paper are:

- We combine deep learning and model-predictive control in a DeepMPC controller that uses learned task dynamics.
- We propose a novel deep architecture which is able to model dynamics conditioned on learned latent properties and a multi-stage pre-training algorithm that avoids common problems in training recurrent neural networks.
- We implement a real-time model predictive control system using our learned dynamics model.
- We demonstrate that our model and controller give strong performance for the difficult task of robotic food-cutting.

II. Related Work

Reactive feedback controllers, where a control signal is generated based on error from current state to some set-point, have been applied since the 19^{th} century [4]. Stiffness control, where error in robot end-effector pose is used to determine end-effector forces, remains a common technique for compliant, force-based activities [5, 2, 15]. Such approaches are limited because they require a trajectory to be given beforehand, making it difficult to adapt to different conditions.

Feed-forward model-predictive control allows controls to adapt online by optimizing some cost function over predicted future states. These approaches have gained increased attention as modern computing power makes it feasible to perform optimization in real time. Shim et al. [36] used MPC to control multiple quad-rotors in simulation, while Howard et al. [20] performed intricate maneuvers with real-world mobile robots. Erez et al. [14] used MPC for full-body control of a humanoid robot. These approaches have been extended to many other tasks, including underwater vehicle control [26], visual servoing [18], and even heart surgery [11]. However, all these works assume the task dynamics model is fully specified.

Model learning for robot control has also been a very active area, and we refer the reader to a review of work in the area by Nguyen-Tuong and Peters [31]. While early works in model learning [1, 30] fit parameters of some hand-designed task-specific model to data, such models can be difficult to design and may not generalize well to new tasks. Thus, several recent works attempt to learn more general dynamics models such as Gaussian mixture models [7, 21] and Gaussian processes [22]. Neural networks [8, 6] are another common choice for learning general non-linear dynamics models. The highly parameterized nature of these models allows them to fit a wide variety of data well, but also makes them very susceptible to overfitting.

Modern deep learning methods retain the advantages of neural networks, while using new algorithms and network architectures to overcome their drawbacks. Due to their effectiveness as general non-linear learners [3], deep learning has been applied to a broad spectrum of problems, including visual recognition [19, 24], natural language processing [9], acoustic modeling [28], and many others. Recurrent deep networks have proven particularly effective for time-dependent tasks such as text generation [37] and speech recognition [17]. Factored conditional models using multiplicative interactions have also been shown to work well for modeling short-term temporal transformations in images [27]. More recently Taylor and Hinton [38] applied these models to human motion, but did not model any control inputs, and treated the conditioning features as a set of fully-observed "motion styles".

Several recent approaches to control learning first learn a dynamics model, then use this model to learn a policy which maps from system state to control inputs. These works often iteratively use this policy to collect more data and re-learn a new policy in an online learning approach. Levine and Abbeel [25] use a Gaussian mixture model (GMM) where linear models are fit to each cluster, while Deisenroth and Rasmussen [10] use a Gaussian process (GP.) Experimentally, both these models gave less accurate predictions than ours for robotic food-cutting. The GP also had very long inference times (roughly 10^6 times longer than ours) due to the large amount of training data needed. For details, see Section VII-B. This weak performance is because they use only temporally-local information, while our model uses learned recurrent features to integrate long-term information and model unobserved system properties such as materials.

These works focus on online policy search, while here we focus on modeling and application to real-time MPC. Our model could be used along with them in a policy-learning approach, allowing them to model dynamics with environmental and temporal variations. However, our model is efficient enough to optimize for predictive control at run-time. This avoids the possibility of learned policies overfitting the training data and allows the cost function and its parameters to be changed online. It also allows our model to be used with other algorithms which use its predictions directly.

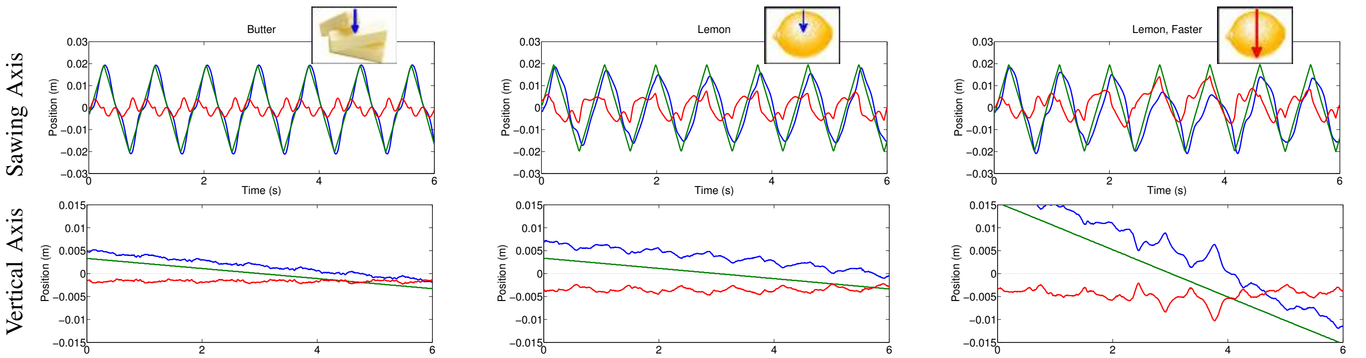

Fig. 3: **Variation in cutting dynamics:** plots showing desired (green) and actual (blue) trajectories, along with error (red) obtained using a stiffness controller while cutting butter (left) and a lemon at low (middle) and high (right) rates of vertical motion.

Gemici and Saxena [15] presented a learning system for manipulating deformable objects which infers a set of material properties, then uses these properties to map objects to a latent set of haptic categories which are used to determine how to manipulate the object. However, their approach requires a predefined set of properties (*plasticity*, *brittleness*, etc.), and chooses between a small set of discrete actions. By contrast, our approach performs continuous-space real-time control, and uses *learned* latent features to model material properties and other variations, avoiding the need for hand-design.

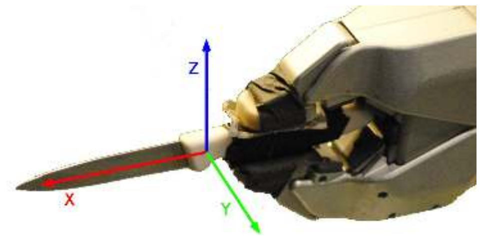

Fig. 4: **Gripper axes:** PR2's gripper with knife grasped, showing the axes used in this paper. The X ("sawing") axis points along the blade of the knife, Y points normal to the blade, and Z points vertically.

III. Problem Definition and System

In this work, we focus on the task of cutting a wide range of food items. This problem is a good testbed for our algorithms because of the variety of dynamics involved in cutting different materials. Designing individual controllers for each material would be very time-consuming, and hand-designing accurate dynamics models for each would be nearly infeasible.

For the task of cutting, we define gripper axes as shown in Fig. 4, such that the X axis points out of the point of the knife, Y axis normal to the blade, and Z axis vertically. Here, we consider linear cutting, where the goal is to make a cut of some given length along the Z axis. The control inputs to the system are denoted as $u^{(t)} = (F_x^{(t)}, F_y^{(t)}, F_z^{(t)})$, where $F_x^{(t)}$ represents the force, in Newtons, applied along the end-effector X axis at time t. The physical state of the system is $x^{(t)} = (P_x^{(t)}, P_y^{(t)}, P_z^{(t)})$ where $P_x^{(t)}$ is the X coordinate of the end-effector's position at time t.

A simple approach to control for this problem might use a fixed-trajectory stiffness controller, where control inputs are proportional to the difference between the current state $x^{(t)}$ and some desired state $x^{*(t)}$ taken from a given trajectory.

Fig. 3 shows some examples which demonstrate the difficulties inherent in this approach. While some materials, such as the butter shown on the left, offer very little resistance, allowing a stiffness controller to accurately follow a given trajectory, others, such as the lemon shown in the remaining two plots, offer more resistance, causing significant deviation from the desired trajectory. When cutting a lemon, we can also see that the dynamics change with time, resisting the knife more as it cuts through the skin, then less once it enters the flesh of the lemon. The dynamics of the sawing and vertical axes are also coupled - increasing the rate of vertical motion increases error along the sawing axis, even though the same controls are used for that axis.

In our approach, we fix the orientation of the end-effector, as well as the position of the knife along its Y axis, using stiffness control to stabilize these. However, even though our primary goal is to move the knife along its Z axis, as shown in Fig. 3, the X and Z axes are strongly coupled for this problem. Thus, our algorithm performs control along both the X and Z axes. This allows "sawing" and "slicing" motions in which movement along the X axis is used to break static friction along the Z axis and enable forward progress. We use a nonlinear function f to predict future states:

$$\hat{x}^{(t+1)} = f(x^{(t)}, u^{(t+1)}) \tag{1}$$

We can apply this formula recurrently to predict further into the future, e.g. $\hat{x}^{(t+2)} = f(\hat{x}^{(t+1)}, u^{(t+2)})$.

A. Model-Predictive Control: Background

In this work, we use a model-predictive controller to control the cutting hand. Such controllers have been shown to work extremely well for a wide variety of tasks for which hand-defined controllers are either difficult to define or simply cannot suffice [20, 14, 26, 11]. Defining $X_{t:k}$ as the system state from time t through time k, and $U_{t:k}$ similarly for system inputs, a model-predictive controller works by finding a set of optimal inputs $U^*_{t+1:t+T}$ which minimize some cost function $C(\hat{X}_{t+1:t+T}, U_{t+1:t+T})$ over predicted state $\hat{X}$ and control inputs U for some finite time horizon T:

$$U^*_{t+1:t+T} = \underset{U_{t+1:t+T}}{\arg\max}\; C(\hat{X}_{t+1:t+T}, U_{t+1:t+T}) \tag{2}$$

This approach is powerful, as it allows us to leverage our knowledge of task dynamics $f(x, u)$ directly, predicting future interactions and proactively avoiding mistakes rather than simply reacting to past observations. It is also versatile, as we have the freedom to design C to define optimality for some task. The chief difficulty lies in modeling the task dynamics $f(x, u)$ in a way that is both differentiable and quick to evaluate, to allow online optimization.

IV. Modeling Time-Varying Non-Linear Dynamics with Deep Networks

Hand-designing models for the entire range of potential interactions encountered in complex tasks such as cutting food would be nearly impossible. Our main challenge in this work is then to design a model capable of *learning* non-linear, time-varying dynamics. This model must be able to respond to short-term changes, such as breaking static friction, and must be able to identify and model variations caused by varying materials and other properties. It must be differentiable with respect to system inputs, and the system outputs and gradients must be fast to compute to be useful for real-time control.

We choose to base our model on deep learning algorithms, a strong choice for our problem for several reasons. They have been shown to be general non-linear learners [3], but remain differentiable using efficent back-propagation algorithms. When time is an issue, as in our case, network sizes can be scaled down to trade accuracy for computational performance.

Although deep networks can learn any non-linear function, care must still be taken to design a task-appropriate model. As shown in Fig. 7, a simple deep network gives relatively weak performance for this problem. Thus, one major contribution of this work is to design a novel deep architecture for modeling dynamics which might vary both with the environment, material, etc., and with time while acting. In this section, we describe our architecture, shown in Fig. 5 and motivate our design decisions in the context of modeling such dynamics.

Dynamic Response Features: When modeling physical dynamics, it is important to capture short-term input-output responses. Thus, rather than learning features separately for system inputs u and outputs x, the basic input features used in our model are a concatenation of both. It is also important to capture high-order and delayed-response modes of interaction. Thus, rather than considering only a single timestep, we consider blocks thereof when computing these features, so that for block b, with block size B, we have visible input features $v^{(b)} = (X_{b*B:(b+1)*B-1}, U_{b*B:(b+1)*B-1})$.

Conditional Dynamic Responses: For tasks such as material cutting, our local dynamics might be conditioned on both time-invariant and time-varying properties. Thus, we must design a model which operates conditional on past information. We do so by introducting factored conditional units [27], where features from some number of inputs modulate multiplicatively and are then weighted to form network outputs. Intuitively, this allows us to blend a set of models based on features extracted from past information. Since our model needs to incorporate both short- and long-term information, we allow three sets of features to interact – the current control inputs, the past block's dynamic response, and latent features modeling long-term observations, described below. Although the past block's response is also included when forming latent features, including it directly in this conditional model frees our latent features from having to model such short-term dependencies.

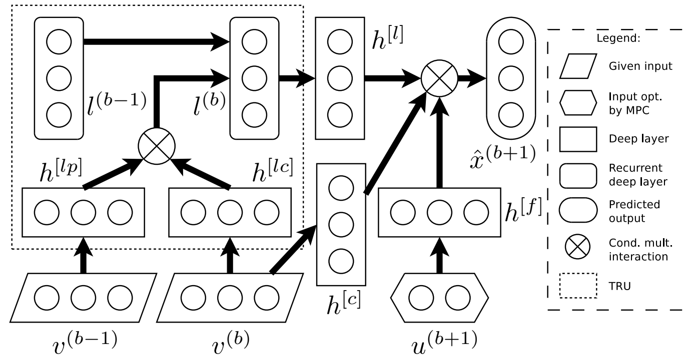

Fig. 5: **Deep predictive model:** Architecture of our recurrent conditional deep predictive dynamics model.

We use c to denote the current timeblock, f to denote the immediate future one, l for latent features, and o for outputs. Take N_v as the number of features v, N_x as the number of states x, and N_u as the number of inputs u in a block, and N_l as the number of latent features l. With $h^{[c](b)} \in \mathbb{R}^{N_{oh}}$ as the hidden features from the current timestep, formed using weights $W^{[c]} \in \mathbb{R}^{N_v \times N_{oh}}$ (similar for f and l), and $W^{[o]} \in \mathbb{R}^{N_{oh} \times N_x}$ as the output weights, our predictive model is then:

$$h_j^{[c](b)} = \sigma\left(\sum_{i=1}^{N_v} W_{i,j}^{[c]} v_i^{(b)}\right) \tag{3}$$

$$h_j^{[f](b)} = \sigma\left(\sum_{i=1}^{N_u} W_{i,j}^{[f]} u_i^{(b+1)}\right) \tag{4}$$

$$h_j^{[l](b)} = \sigma\left(\sum_{i=1}^{N_l} W_{i,j}^{[l]} l_i^{(b)}\right) \tag{5}$$

$$\hat{x}_j^{(b+1)} = \sum_{i=1}^{N_{oh}} W_{i,j}^{[o]} h_i^{[c](b)} h_i^{[f](b)} h_i^{[l](b)} \tag{6}$$

Long-Term Recurrent Latent Features: Another major challenge in modeling time-dependent dynamics is integrating long-term information while still allowing for transitions in dynamics, such as moving from the skin to the flesh of a lemon. To this end, we introduce transforming recurrent units (TRUs). To retain state information from previous observations, our TRUs use temporal recurrence, where each latent unit has weights to the previous timestep's latent features. To allow this state to transition based on locally observed transformations in dynamics, they use the paired-filter behavior of multiplicative interactions to detect transitions in the dynamic response of the system and update the latent state accordingly. In previous work, multiplicative factored conditional units have been shown to work well in modeling transformations in images [27] and physical states [38], making them a good choice here. Each TRU thus takes input from the previous TRU's output and the short-term response features for the current and previous time-blocks. With ll denoting recurrent weights, lc denoting current-step for the latent features, lp previous-step, and lo output, and N_{lh} as the number of TRU hidden units, our latent feature model is then:

$$h_j^{[lc](b)} = \sigma\left(\sum_{i=1}^{N_v} W_{i,j}^{[c]} v_i^{(b)}\right) \quad (7)$$

$$h_j^{[lp](b)} = \sigma\left(\sum_{i=1}^{N_v} W_{i,j}^{[f]} v_i^{(b-1)}\right) \quad (8)$$

$$l_j^{(b)} = \sigma\left(\sum_{i=1}^{N_{lh}} W_{i,j}^{[lo]} h_i^{[lc](b)} h_i^{[lp](b)} + \sum_{k=1}^{N_l} W_{k,j}^{[ll]} l_k^{(b-1)}\right) \quad (9)$$

Finally, Fig. 5 shows the full architecture of our deep predictive model, as described above.

V. Learning and Inference

In this section, we define the learning and inference procedures for the model defined above. The online inference approach is a direct result of our model. However, there are many possible approaches to learning its parameters. Neural networks require a huge number of parameters (weights) to be learned, making them particularly susceptible to overfitting, and recurrent networks often suffer from instability in future predictions, causing large gradients which make optimization difficult (the "exploding gradient" problem).

To avoid these issues, we define a new three-stage learning approach which pre-trains the network weights before using them for recurrent modeling. Deep learning methods are non-convex, and converge to only a *local* optimum, making our approach important in ensuring that a good optimum which does not overfit the training data is reached.

Inference: During inference for MPC, we are currently at some time-block b with latent state $l^{(b)}$, known system state $x^{(b)}$ and control inputs $u^{(b)}$. Future control inputs $U_{t+1:t+T}$ are also given, and our goal is then to predict the future system states $\hat{X}_{t+1:t+T}$ up to time-horizon T, along with the gradients $\partial X/\partial U$ for all pairs of x and u. We do so by applying our model recurrently to predict future states up to time-horizon T, using predicted states $\hat{x}$ and latent features l as inputs to our predictive model for subsequent timesteps, e.g. when predicting $x^{(b+2)}$, we use the known $x^{(b)}$ along with the predicted $\hat{x}^{(b+1)}$ and $l^{(b+1)}$ as inputs.

Our model's outputs ($\hat{x}$) are differentiable with respect to all its inputs, allowing us to take gradients $\partial X/\partial U$ using an approach similar to the backpropagation-through-time algorithm used to optimize model parameters during learning. We can in turn use these gradients with any gradient-based optimization algorithm to optimize $U_{t+1:t+T}$ with respect to some differentiable cost function $C(X, U)$. No online optimization is necessary to perform inference for our model.

Learning: During learning, our objective is to use our training data to learn a set of model parameters $\Theta = (W^{[f]}, W^{[c]}, W^{[l]}, W^{[o]}, W^{[lp]}, W^{[lc]}, W^{[ll]}, W^{[lo]})$ which minimize prediction error while avoiding overfitting. A naive approach to learning might randomly initialize Θ, then optimize the entire recurrent model for prediction error. However, random weights would likely cause the model to make inaccurate predictions, which will in turn be fed forwards to future timesteps. This could cause huge errors at time-horizon T, which will in turn cause large gradients to be back-propagated, resulting in instability in the learning and overfitting to the training data. To remedy this, we propose a multi-stage pre-training approach which first optimizes some subsets of the weights, leading to much more accurate predictions and less instability when optimizing the final recurrent network. We show in Fig. 7 that our learning algorithm significantly outperforms random initialization.

Phase 1: Unsupervised Pre-Training: In order to obtain a good initial set of features for l, we apply an unsupervised learning algorithm similar to the sparse auto-encoder algorithm [16] to train the non-recurrent parameters of the TRUs. This algorithm first projects from the TRU inputs up to l, then uses the projected l to reconstruct these inputs. The TRU weights are optimized for a combination of reconstruction error and sparsity in the outputs of l.

Phase 2: Short-term Prediction Training: While we could now use these parameters as a starting point to optimize a fully recurrent multi-step prediction system, we found that in practice, this lead to instability in the predicted values, since inaccuracies in initial predictions might "blow up" and cause huge deviations in future timesteps.

Instead, we include a second pre-training phase, where we train the model to predict a single timestep into the future. This allows the model to adjust from the task of reconstruction to that of physical prediction, without risking the aforementioned instability. For this stage, we remove the recurrent weights from the TRUs, effectively setting all $W^{[ll]}$ to zero and ignoring them for this phase of optimization.

Taking $x^{(m,k)}$ as the state for the k^{th} time-block of training case m, M as the number of training cases, and B_m as the number of timeblocks for case m, this stage optimizes:

$$\Theta^* = \arg\min_{\Theta} \sum_{m=1}^{M} \sum_{b=2}^{B_m-1} ||\hat{x}^{(m,b+1)} - x^{(m,b+1)}||_2^2 \quad (10)$$

Phase 3: Warm-Latent Recurrent Training: Once Θ has been pre-trained by these two phases, we use them to initialize a recurrent prediction system which performs inference as described above. We then optimize this system to minimize the sum-squared prediction error up to T timesteps in the future, using a variant of the backpropagation-through-time algorithm commonly used for recurrent neural networks [34].

When run online, our model will typically have some amount of past information, as we allow a short period where we optimize forces while a stiffness controller makes an initial inwards motion. Thus, simply initializing the latent state "cold" from some intial state and immediately penalizing prediction error does not match well with the actual use of the network, and might in fact introduce overfitting by forcing the model to rely more heavily on short-term information. Instead, we train our model for a "warm" start. For some number of initial time-blocks B_{W}, we propagate latent state l, but do not predict or penalize system state $\hat{x}$, only doing so after this warm-up phase. We still back-propagate errors from future timesteps through the warm-up latent states as normal.

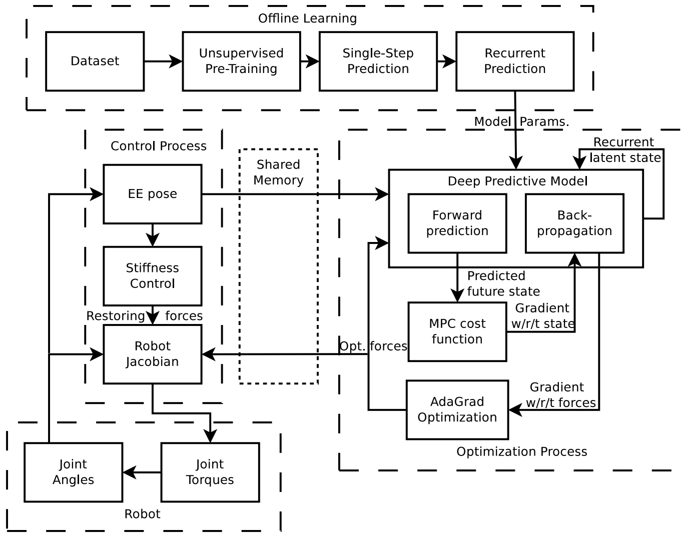

Fig. 6: **Online system:** Block diagram of our DeepMPC system.

VI. System Details

Learning System: We used the L-BFGS algorithm, shown to give strong results for deep learning methods [23], to optimize our model during learning. While larger network sizes gave slightly (∼10%) less error, we found that setting N_{lh} = 50, N_l = 50, and N_{oh} = 100 was a good tradeoff between accuracy and computational performance. We found that block size B = 10, giving blocks of 0.1s, gave the best performance. When implemented on the GPU in MATLAB, all phases of our learning algorithm took roughly 30 minutes to optimize.

Robotic Platform: For both data collection and online evaluation of our algorithms, we used a PR2 robot. The PR2 has two 7-DoF manipulators with parallel-plate grippers, and a reach of roughly 1m. For safety reasons, we limit the forces applied by PR2's arms to 30N along each axis, which was sufficient to cut every material tested. PR2 natively runs the Robot Operating System (ROS) [33]. Its arm controllers recieve robot state information in the form of joint angles and must publish desired motor torques at a hard real-time rate of 1KHz.

Online Model-Predictive Control System: The main challenge in designing a real-time model-predictive controller for this architecture lies in allowing prediction and optimization to run continuously to ensure optimality of controls, while providing the model with the most recent state information and performing control at the required real-time rate. As shown in Fig. 6, we solve this by separating our online system into two processes (ROS nodes), one performing continuous optimization, and the other real-time control. These processes use a shared memory space for high-rate inter-process communication. This approach is modular and flexible - the optimization process is generic to the robot involved (given an appropriate model), while the control process is robot-specific, but generic to the task at hand. In fact, models for the optimization process do not even need to be learned locally, but could be shared using an online platform [35].

The **control** process is designed to perform minimal computation so that it can be called at a rate of 1KHz. It recieves robot state information in the form of joint angles, and control information from the optimization process as end-effector forces. It performs forward kinematics to determine end-effector pose, transmits it to the optimization process, and uses it to determine restoring forces for axes not controlled by MPC. It translates the combination of these forces and those recieved from MPC to a set of joint torques sent to the arm. All operations performed by the control process are at most quadratic in terms of the number of degrees of freedom of the arm, allowing each call to run in roughly 0.1 ms on PR2.

The **optimization** process runs as a continuous loop. When started, it loads model parameters (network weights) from disk. Cost function parameters are loaded from a ROS parameter server, allowing them to be changed online. The optimization loop first uses past robot states (received from the control process) and control inputs along with past latent state and the future forces being optimized to predict future state using our model. It then uses this state to compute the gradients of the MPC cost function and back-propagates these through our model, yielding gradients with respect to the future forces. It optimizes these forces using a variant of the AdaGrad algorithm [12], a form of gradient descent in which gradient contributions are scaled by the L2 norm of past gradients, chosen because it is efficient in terms of function evaluations while avoiding scaling issues. This process is implemented using the Eigen matrix library [13], allowing the optimization loop to run at a rate of over 1.2kHz.

MPC Cost Function: In order to perform MPC, we need to define a cost function $C(X, U)$ for our task. For food cutting, we design a cost function with two main components, with β defining the weighting between them:

$$C(X, U) = C_{\text{cut}}(X) + \beta C_{\text{saw}}(X) \tag{11}$$

The first, C_{cut}, drives the controller to move the knife downwards. It penalizes the height of the knife at each timestep, with an additional penalty at the final timestep allowing a tradeoff between immediate and eventual downwards motion:

$$C_{\text{cut}}(X) = \sum_{k=t}^{t+T} P_z^{(k)} + \gamma P_z^{(t+T)} \tag{12}$$

The second term, C_{saw}, keeps the tip of the knife inside some reasonable "sawing range" along the X axis, ensuring that it actually cuts through the food. Since any valid position is acceptable, this term will be zero inside some margins from this range, then become quadratic once it passes those margins. Taking P_x^* as the center point of the sawing range, d_{S} as the range, and λ as the margin, we define this term as:

$$C_{\text{saw}}(X) = \sum_{k=t}^{t+T} \left(\max\left\{0, |P_x^{(k)} - P_x^*| - d_{\text{S}} + \lambda \right\} \right)^2 \tag{13}$$

We also include terms performing first- and second-order smoothing and L2 regularization on the control forces.

VII. Experiments

In order to evaluate our algorithm as compared to other baseline and state-of-the-art approaches, we performed a series of experiments, both offline on our large dataset of material interactions, and online on our PR2 robot.

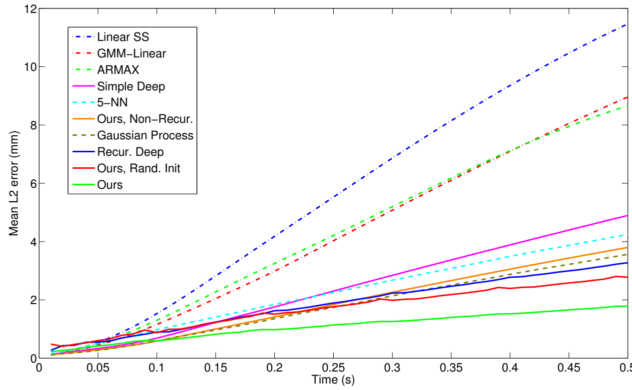

Fig. 7: **Prediction error:** Mean L2 distance (in mm) from predicted to ground-truth trajectory from 0.01s to 0.5s in the future.

A. Dataset

Our material interaction dataset contains 1488 examples of robotic food-cutting for 20 different materials (Fig. 2). We collected data from three different settings. First, a fixed-parameter setting in which trajectories as shown in the leftmost two columns of Fig. 3 were used with a stiffness controller. Second, for 8 of the 20 materials in question, data was collected while a human tuned a stiffness controller to improve cutting rate. This data was not collected for all materials to avoid giving the learning algorithm and controller near-optimal cases for all materials. Third, a randomized setting where most parameters of the controller, including cutting and sawing rate and stiffnesses, but excluding sawing range (still fixed at 4cm) were randomized for each stroke of the knife. This helped to obtain data spanning the entire range of possible interactions.

B. Prediction Experiments

Setting: In order to test our model, we examine its predictive accuracy compared to several other approaches. Each model was given 0.7s worth of past trajectory information (forces and known poses) and 0.5s of future forces and then asked to predict the future end-effector trajectory. For this experiment, we used 70% of our data for training, 10% for validation, and 20% for testing, sampling to keep each set class-balanced.

Baselines: For prediction, we include a linear state-space model, an ARMAX model which also weights a range of past states, and a K-Nearest Neighbors (K-NN) model (5-NN gave the best results) as baseline algorithms. We also include a GMM-linear model [25], and a Gaussian process (GP) model [10], trained using the GPML package [32]. Additionally, we compare to standard recurrent and non-recurrent two-layer deep networks, and versions of our approach without recurrence and without pre-training (randomly initializing weights, then training for recurrent prediction).

Results: Fig. 7 shows performance for each model as mean L2 distance from predicted to ground truth trajectory vs. prediction time in the future. Temporally-local (piecewise-) linear methods (linear SS, GMM-linear, and ARMAX) gave weak performance for this problem, each yielding an average error of over 8mm at 0.5s. This shows, as expected, that linear models are a poor fit for our highly non-linear problem.

Instance-based learning methods – K-NN and Gaussian processes – gave better performance, at an average of 4.25mm and 3.56mm, respectively. Both outperformed the baseline two-layer non-recurrent deep network, which gave 4.90mm error. The GP gave the best performance of any temporally-local model, although this came at the cost of extreme inference time, taking an average of 3361s (56 minutes) to predict 0.5s into the future, $1.18\text{x}10^6$ times slower than our algorithm, whose MATLAB implementation took only 3.1ms.

The relatively unimpressive performance of a standard two-layer deep network for this problem underscores the need for task-appropriate architectures and learning algorithms. Including conditional structures, as in the non-recurrent version of our model and temporal recurrence reduced this error to 3.80mm and 3.27mm, respectively. Even when randomly initialized, our model outperformed the baseline recurrent network, giving only 2.78mm error, showing the importance of using an appropriate architecture. Using our learning algorithm further reduced error to 1.78mm, 36% less than the randomly-initialized version and 46% less than the baseline recurrent model, demonstrating the need for careful pre-training.

Our approach also gave a tighter and lower 95% confidence interval of prediction error, from 0.17-7.23mm at 0.5s, a width of 7.06mm, compared to the baseline recurrent net's interval of 0.26-10.71mm, a width of 10.45mm, and the GP's interval of 0.34-15.22mm, a width of 14.88mm.

C. Robotic Experiments

Setting: To examine the actual control performance of our system, we also performed a series of online robotic experiments. In these experiments, the robot's goal was to make a linear cut between two given points. We selected a subset of 10 of the 20 materials in our dataset for these experiments, aiming to span the range of variation in material properties. We evaluated these experiments in terms of cutting rate, i.e. the vertical distance traveled by the knife divided by time taken.

Baselines: For control, we validate our algorithm against several other control methods with varying levels of class information. First, we compare to a class-generic stiffness controller, using the same controller for all classes. This controller was tuned to give a good (>90%) rate of success for all classes, meaning that it is much slower than necessary for easy-to-cut classes such as tofu. We also validate against controllers tuned separately for each of the test classes, using the same criteria as above, showing the maximum cutting rate that can be expected from fixed-trajectory stiffness control.

As a middleground, we compare to an algorithm similar to that of Gemici and Saxena [15], where a set of class-specific material properties are mapped to learned haptic categories. We found that learning five such categories assigned all data for each class to exactly one cluster, and thus used the same controller for all classes assigned to each cluster. In cases where this controller was the same as used for the class-tuned case, we used the same results for both.

Results: Figure 8 shows the results of our series of over 450 robotic experiments. For all materials except butter and tofu,

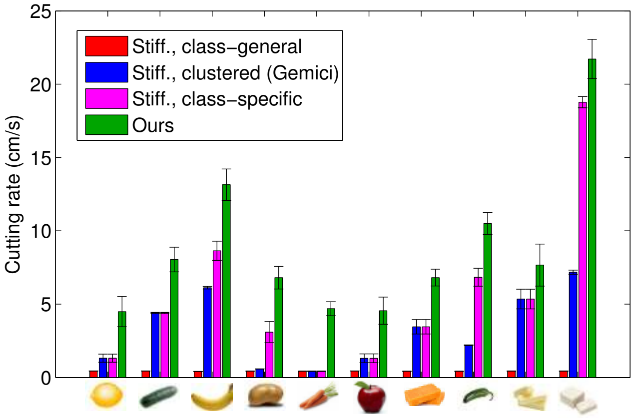

Fig. 8: **Robotic experiment results:** Mean cutting rates, with bars showing normalized standard deviation, for ten diverse materials. Red bar uses the same controller for all materials, blue bar uses the same for each cluster given by [15], purple uses a tuned stiffness controller for each, and green is our online MPC method.

a paired t-test showed that our DeepMPC controller made a statistically significant improvement in cutting rate with 95% confidence. This makes sense as butter and tofu are relatively soft and easy-to-cut materials. However, for the four materials for which stiffness control gave the weakest results – lemons, potatoes, carrots, and apples – our algorithm more than tripled the mean cutting rate, from 1.5 cm/s to 5.1 cm/s.

One major advantage our approach has over the others tested is that it treats material properties and classes as latent and continuous-valued, rather than supervised and discrete. For intra-class variations which affect dynamics, such as different varieties of apples or cheeses, different radii of carrots or potatoes, or varying material temperature, even the class-specific stiffness controllers were typically limited by the hardest-to-cut variation. However, our approach's latent material properties allowed it to adapt to these, significantly increasing cutting rates. This was particularly evident for carrots, whose thickness causes huge variations in dynamics. While all approaches were tested on both thick and thin sections of carrot, only ours was able to properly adapt, slicing easily through thin sections and more carefully through thicker ones, increasing mean cutting rate from 0.4 cm/s to 4.7 cm/s. Similar results were observed for potatoes, increasing mean rate from 3.1 cm/s to 6.8 cm/s.

Another advantage of our approach is its ability to respond to time-dependent changes in dynamics, thanks to the time-varying nature of our latent features and the online adaptation performed by MPC. Such changes occur to some degree as the knife enters and becomes enclosed by most materials, particularly in irregular shapes such as potatoes where the degree of enclosure varies throughout the cut. They are even more apparent in non-uniform materials, such as lemons, with variation between the skin and flesh, and apples, which are much denser and harder to cut closer to the core. Again, stiffness control was limited by the toughest of these dynamics, while our approach was able to adapt, typically performing more sawing for more difficult regions, and quickly moving

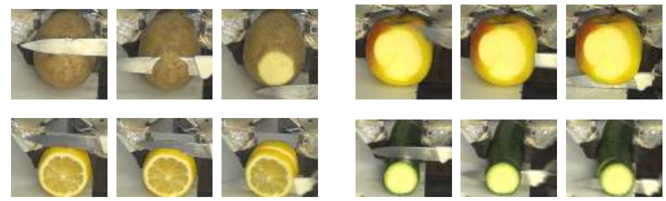

Fig. 9: **Cutting food:** Time-series of our PR2 robot using our DeepMPC controller to cut several of the food items in our dataset.

downwards for easier ones. This allowed our controller to increase the mean cutting rate for lemons from 1.3 cm/s to 4.5 cm/s, and for apples from 1.4 cm/s to 4.6 cm/s.

Optimizing its trajectory online also enabled our DeepMPC controller to exhibit a much more diverse range of behaviors. Most tuned stiffness controllers were forced to make use of high-amplitude sawing to ensure continuous motion. However, our controller was able to use more aggressive cutting strategies, typically executing smooth slicing motions until it found its progress impeded. It then used a variety of techniques to break static friction and continue motion, including high-amplitude sawing, low-amplitude "wiggle" motions, and reducing and re-applying vertical pressure, even to the point of picking up the knife slightly in some cases. The latter behavior, in particular, underscores the strength of predictive control, as it trades off short-term losses for long-term gains. Stiffness controllers sometimes became stuck in tough materials such as thick potatoes and carrots and the cores of apples, and remained so because downwards force grew as vertical error increased. Our controller, however, was able to detect and break such cases using these techniques.

Some examples of the diverse behaviors of our DeepMPC controller can be seen in Fig. 9 and in the video at `http://deepmpc.cs.cornell.edu`.

VIII. Conclusion

In this work, we presented DeepMPC, a novel approach to model-predictive control for complex non-linear dynamics which might vary both with environment properties and with time while acting. Instead of hand-designing predictive dynamics models, which is extremely difficult and time-consuming for such tasks, our approach uses a new deep architecture and learning algorithm to accurately model even such complicated dynamics. In experiments on our large-scale dataset of 1488 material cuts over 20 diverse materials, we showed that our approach improves accuracy by 46% as compared to a standard recurrent deep network. In a series of over 450 real-world robotic experiments for the challenging problem of robotic food-cutting, we showed that our algorithm produced significant improvements in cutting rate for all but the easiest-to-cut materials, and over tripled average cutting rates for the most difficult ones.

Acknowledgements

This work was supported in part by Army Research Office (ARO) award W911NF-12-1-0267, NSF National Robotics Initiative (NRI) award IIS-1426744, a Microsoft Faculty Fellowship, and an NSF CAREER Award.

References

[1] C. G. Atkeson, C. H. An, and J. M. Hollerbach. Estimation of inertial parameters of manipulator loads and links. *Int. J. Rob. Res.*, 5(3):101–119, Sept. 1986.

[2] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth. Robotic Roommates Making Pancakes. In *Humanoids*, 2011.

[3] Y. Bengio. Learning deep architectures for AI. *FTML*, 2 (1):1–127, 2009.

[4] S. Bennett. A brief history of automatic control. *Control Systems, IEEE*, 16(3):17–25, Jun 1996. ISSN 1066-033X.

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

[6] M. V. Butz, O. Herbort, and J. Hoffmann. Exploiting redundancy for flexible behavior: Unsupervised learning in a modular sensorimotor control architecture. *Psychological Review*, 114:1015–1046, 2007.

[7] S. Chernova and M. Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *AAMAS*, 2007.

[8] C.-M. Chow, A. G. Kuznetsov, and D. W. Clarke. Successive one-step-ahead predictions in multiple model predictive control. *Int. J. Systems Science*, 29(9):971–979, 1998.

[9] R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, 2008.

[10] M. P. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[11] M. Dominici and R. Cortesao. Model predictive control architectures with force feedback for robotic-assisted beating heart surgery. In *ICRA*, 2014.

[12] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159, July 2011.

[13] Eigen Matrix Library. http://eigen.tuxfamily.org.

[14] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov. An integrated system for real-time model-predictive control of humanoid robots. In *ICHR*, 2013.

[15] M. Gemici and A. Saxena. Learning haptic representation for manipulating deformable food objects. In *IROS*, 2014.

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

[17] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[18] S. Heshmati-alamdari, G. K. Karavas, A. Eqtami, M. Drossakis, and K. Kyriakopoulos. Robustness analysis of model predictive control for constrained image-based visual servoing. In *ICRA*, 2014.

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

[20] T. Howard, C. Green, and A. Kelly. Receding horizon model-predictive control for mobile robot navigation of intricate paths. In *International Conference on Field and Service Robotics*, July 2009.

[21] S. Khansari-Zadeh and A. Billard. Learning stable non-linear dynamical systems with gaussian mixture models. *Robotics, IEEE Transactions on*, 27(5):943–957, Oct 2011.

[22] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard. Gaussian process model based predictive control. In *American Control Conference*, 2004.

[23] Q. V. Le, A. Coates, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *ICML*, 2011.

[24] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, 2009.

[25] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, 2015.

[26] C. McFarland and L. Whitcomb. Experimental evaluation of adaptive model-based control for underwater vehicles in the presence of unmodeled actuator dynamics. In *ICRA*, 2014.

[27] R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, June 2010.

[28] A.-R. Mohamed, G. Dahl, and G. E. Hinton. Acoustic modeling using deep belief networks. *IEEE Trans Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[29] F. C. Moon and T. Kalmár-Nagy. Nonlinear models for complex dynamics in cutting materials. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 359 (1781):695–711, 2001.

[30] K. S. Narendra and A. M. Annaswamy. *Stable Adaptive Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-839994-8.

[31] D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. *Cognitive Processing*, 12(4), 2011.

[32] C. E. Rasmussen and H. Nickisch. GPML Matlab Code version 3.5. http://www.gaussianprocess.org/gpml/code/matlab/doc/.

[33] ROS: Robot Operating System. http://www.ros.org.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, 1986.

[35] 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.

[36] D. Shim, H. Kim, and S. Sastry. Decentralized reflective model predictive control of multiple flying robots in dynamic environment. In *Conference on Decision and Control*, 2003.

[37] I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[38] G. W. Taylor and G. E. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *ICML*, 2009.