A Decision Tree Framework for Spatiotemporal Sequence Prediction

Taehwan Kim
Toyota Technological Institute at Chicago
taehwan@ttic.edu

Yisong Yue
California Institute of Technology
yyue@caltech.edu

Sarah Taylor
Disney Research Pittsburgh
sarah.taylor@disneyresearch.com

Iain Matthews
Disney Research Pittsburgh
iainm@disneyresearch.com

ABSTRACT

We study the problem of learning to predict a spatiotemporal output sequence given an input sequence. In contrast to conventional sequence prediction problems such as part-of-speech tagging (where output sequences are selected using a relatively small set of discrete labels), our goal is to predict sequences that lie within a high-dimensional continuous output space. We present a decision tree framework for learning an accurate non-parametric spatiotemporal sequence predictor. Our approach enjoys several attractive properties, including ease of training, fast performance at test time, and the ability to robustly tolerate corrupted training data using a novel latent variable approach. We evaluate on several datasets, and demonstrate substantial improvements over existing decision tree based sequence learning frameworks such as SEARN and DAgger.

Categories and Subject Descriptors: I.2.6 Artificial Intelligence: Learning - Induction

General Terms: Algorithms; Experimentation.

Keywords: Decision Trees; Sequence Prediction.

1. INTRODUCTION

Contextual sequence prediction is an important problem in many domains, ranging from natural language processing tasks such as part-of-speech tagging and named-entity recognition [1, 6, 17], to computational biology tasks such as sequence alignment [10, 28, 38]. The basic setting can be defined as generating a sequential output given a sequential input. For example, in part-of-speech tagging, the input can be a sequence of words (i.e., a sentence), and the output is the corresponding sequence of part-of-speech tags.

In this paper, we study the problem of spatiotemporal sequence prediction. In contrast to conventional sequence prediction where output sequences are typically selected using a relatively small set of discrete labels (e.g., a set of part-of-speech tags), our goal is to predict sequences that lie within a high-dimensional continuous output space. One example is visual speech generation, where the goal is to predict a sequence of face configurations given a sequence of audio or phonetic inputs [33, 40]. Other example applications include speech synthesis [34] and human motion prediction [15].

Predicting spatiotemporal sequences presents several technical challenges. First, outputs can vary continuously, which leads to a high-dimensional sequential regression problem. Most notably, specific spatial patterns can develop over multiple frames. For example, accurate visual speech animation requires capturing a wide range of "temporal curvature" in lip motions, from smoothly to sharply varying. A second challenge stems from the fact that, for many applications, it can be difficult to specify a semantically meaningful feature representation. As such, conventional sequence prediction approaches (cf. [1, 6, 17]) are unlikely to produce accurate models, since they typically are designed to predict over discrete output labels, make strong Markovian assumptions, and are linear models with respect to the feature representation.

In this paper, we propose a simple yet effective decision tree framework for learning an accurate non-parametric spatiotemporal sequence predictor. Decision trees are an attractive model class due their ability to capture near-arbitrary non-linear predictors based on the input features [21, 24]. One challenge with decision trees is that they cannot be naturally applied to sequence prediction problems. We present an approach to decompose spatiotemporal sequence prediction into a series of overlapping fixed-length multivariate regression problems, which can be naturally trained using decision trees. For example, in our experiments on visual speech animation, our base decision tree model is a 150-dimensional regressor. Our approach also enjoys attractive computational properties, including ease of training and fast performance at test time.

Our approach is complementary to existing decomposition approaches such as SEARN [7] and DAgger [25]. The main difference is that our decomposition focuses on capturing the local temporal curvature of spatiotemporal sequences, whereas SEARN and DAgger focus on controlling for cascading error effects due to longer range dependencies. We show empirically that our approach consistently outperforms or is competitive with SEARN and DAgger, while being significantly faster to train.

A third challenge of working with spatiotemporal sequence data is that such data is often partially corrupted, e.g., with missing values or misalignments [20, 36]. We further propose a latent variable extension to our decision tree framework that can robustly tolerate such corrupted data. Both our main framework as well as the latent variable extension are compatible with using ensemble methods such as random forests [3] as well as single decision trees.

Our contributions can be summarized as follows:

- We propose a discriminative learning framework based on decision trees for spatiotemporal sequence prediction. Our approach decomposes the prediction task into a series of over-

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783356.

lapping fixed-length multivariate regression problems, which can be naturally trained using decision trees. Our approach enjoys attractive computational properties, including ease of training and fast performance at test time.

- We propose a novel latent variable extension that can robustly tolerate corrupted training data such as missing values and misalignments. This extension is compatible with using random forests as well as decision trees.
- We evaluate our approach using several benchmark datasets and demonstrate competitive or substantially better performance over existing decomposition approaches such as SEARN [7] and DAgger [25], as well as state-of-the-art baselines based on HMMs [40] and semi-Markov models [33].
- We showcase the practicality of our approach for an application in data-driven visual speech animation. In a user study, we found that our approach generates significantly more realistic animations when compared to several strong baselines.
- We evaluate the robustness of our latent variable extension in dealing with partially corrupted training data such as missing values and misalignments. Our experiments demonstrate that our latent variable extension is resilient to a substantial amount of corrupted training data.

2. RELATED WORK

The study of sequence prediction enjoys a long history within the machine learning and related communities. Early work in sequence labeling centered around the use of generative models such as hidden Markov models (HMMs) and stochastic grammars [10, 22]. However, many sequence prediction tasks require conditioning on an input sequence or some other context. For example, in part-of-speech tagging, one must predict a sequence of part-of-speech tags conditioned on an input sequence of words (i.e., a sentence). Indeed, contextual or conditional sequence prediction problems are pervasive across the sciences, ranging from natural language processing [1, 6, 17] to computation biology [28, 38], and can be difficult to accurately model using generative models such as HMMs.

The need to better tackle contextual sequence prediction has led to the development of discriminative learning methods such as structured perceptrons [6], Conditional Random Fields [17, 26], Max-Margin Markov Networks [32], and structural Support Vector Machines [1, 35], as well methods for modeling more complex output spaces [9, 37] (e.g., higher order sequential models). These approaches typically learn linear models over sequence-based features to directly maximize the accuracy of resulting contextual sequence predictor. From the perspective of our work, two important limitations of this line of research are the inability to naturally deal with the continuous nature of spatiotemporal outputs, and the reliance on having a semantically rich feature representation.

Spatiotemporal sequence modeling is an area of increasing interest due to the growing availability of spatiotemporal sequence data. Example domains include motion & pose tracking [15, 27, 33, 40], speech synthesis [34], player tracking in sports [4, 39], and other behavioral tracking areas [11, 42]. We are especially interested in settings where the goal is to predict a spatiotemporal output sequence given (i.e., conditioned on) an input sequence or some other context. For example, in visual speech animation, the goal is to predict an animation sequence of a face given an audio or phonetic input sequence [27, 33, 40]. Because existing discriminative approaches for sequence prediction are largely limited to predicting discrete sequences, existing state-of-the-art approaches to visual speech animation typically resort to continuous variants of generative approaches such as HMMs [40] and semi-Markov models [33].

Existing discriminative approaches for spatiotemporal modeling typically focus on predicting a discrete label over an entire spatiotemporal input sequence, such as predicting whether a given sequence belongs to a certain class (cf. [11, 14, 19]). In contrast, we are interested in the "reverse" problem of predicting a spatiotemporal sequence, rather than classifying one.

Our approach is a decomposition or reduction approach that decomposes the spatiotemporal sequence prediction problem into a series of overlapping "sliding window" prediction problems. Decomposition approaches are attractive since they allow for utilizing powerful non-parametric base models such as decision trees [21, 24]; such base models are difficult to apply directly to sequence prediction problems. Existing decomposition approaches for sequence prediction, such as SEARN [7] and DAgger [25], are typically designed to control for cascading error effects from long range dependencies, and are complementary to our approach. Our approach instead focuses on accurately capturing the local temporal curvature of spatiotemporal sequences. Because of their self-recurrent definition, SEARN and DAgger require iterative training of the base model, which can be quite slow. As we shall see in our experiments, our approach can dramatically outperform SEARN and DAgger, while being substantially faster to train.

Our work bears affinity to structured decision tree methods for tasks such as edge detection [8] and image labeling [16], which also employ a sliding window approach. The main difference is that our output space is a continuous spatiotemporal sequence, with prediction goals such as generating realistic facial animations to match an accompanying audio track; this leads to a different choice of decision tree base models. We also consider settings with corrupted training data, as described below.

One important challenge when dealing with spatiotemporal sequence data is the fact that the training data can often be partially corrupted. The two most common types of corruption are missing values [12, 15] and misalignments [5, 18, 20, 30, 41]. Missing values commonly occur when the spatiotemporal data is generated from tracking data that has occlusions, such as in human motion capture and articulatory measurement datasets [15, 36]. The typical approach to resolving missing values is data imputation, possibly using a low-rank assumption [36]. Misalignments can arise due to imperfections in the tracking technology for generating the spatiotemporal training data [30], or from natural temporal variability in the phonomenon being studied [18, 20, 41], or both. Common techniques for resolving misalignment include variants of dynamic time warping [20] as well as curve alignment and clustering [13].

For our setting, we consider the case where the output sequence (i.e., the spatiotemporal sequence to be predicted) is corrupted in the training data, either due to missing values or misalignments. We propose a latent-variable extension to our basic decomposition framework to jointly estimate a "cleaner" version of training data while learning a contextual spatiotemporal sequence predictor. Our extended framework can naturally incorporate many existing techniques for missing value imputation and misalignment correction.

Other notable sequence modeling problems studied recently include machine translation [31], and semantically aware sentiment analysis [29]. Such problems typically require modeling very long-range dependencies within the input sequence (e.g. words far apart in the input sentence), and so are not well suited for our approach.

3. THE LEARNING PROBLEM

Let $\mathbf{x} = \langle x_1, \ldots, x_{|\mathbf{x}|} \rangle$ denote an input sequence, and let $\mathbf{y} = \langle y_1, \ldots, y_{|\mathbf{y}|} \rangle$ denote a spatiotemporal output sequence. We use bold face **x** and **y** to denote input and output sequences, respectively, and use unbolded x and y to refer to individual entries in the sequences, which we also refer to as tokens or frames. Each output frame $y \in \Re^D$ is represented as a point in some D-dimensional space, and we use superscripts $y^{(d)}$ to refer to individual dimensions in the output frame. We often think of the sequences as time-varying, i.e., that frame y_t temporally preceeds frame y_{t+1}. For example, in visual speech animation, **x** could correspond to an audio sequence, and **y** could correspond to an animation sequence of a face model with D degrees of freedom. Figure 1 depicts an illustration of **x** and **y**, which corresponds to a phonetic input sequence and a one-dimensional spatiotemporal output sequence corresponding to one of the parameters of a face model animating to the word "prediction".

Following the standard machine learning setup, our goal is to a learn a function $h(\mathbf{x}) := \mathbf{y}$ that maps input sequences to spatiotemporal output sequences. We restrict ourselves to the supervised learning scenario, where input/output pairs $(\mathbf{x}, \mathbf{y})$ are available for training and are assumed to come from some fixed distribution $P(\mathbf{x}, \mathbf{y})$. The goal is to find a predictor h such that the risk (i.e., expected loss),

$$L_P(h) = \int \ell(\mathbf{y}, h(\mathbf{x})) dP(\mathbf{x}, \mathbf{y}), \tag{1}$$

is minimized. In this paper, we take the view of spatiotemporal sequence prediction as a high-dimensional regression problem, and thus use the squared L2 error,

$$\ell(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|^2_{Fro},$$

to measure imperfections in the predicton $h(\mathbf{x})$ when the true output sequence is **y**.[1]

Of course, $P(\mathbf{x}, \mathbf{y})$ is unknown. But given a training set of input/output pairs drawn from $P(\mathbf{x}, \mathbf{y})$,

$$S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N, \tag{2}$$

we can instead approximately minimize (1) by minimizing the empirical risk,

$$L_S(h) = \sum_{(\mathbf{x},\mathbf{y}) \in S} \ell(\mathbf{y}, h(\mathbf{x})), \tag{3}$$

which is equivalent to finding an h that minimizes the training loss.

3.1 Corrupted Training Data

We also consider the case where the output sequence (i.e., the training label) may be corrupted in the training data. In particular, we can now rewrite our training set as

$$S = \{(\mathbf{x}_i, \tilde{\mathbf{y}}_i)\}_{i=1}^N, \tag{4}$$

where each $\tilde{\mathbf{y}}_i$ is a potentially corrupted version of $\mathbf{y}_i$. Despite training on corrupted $\tilde{\mathbf{y}}$, our goal is to still learn a predictor that minimizes the risk on the original test distribution (1). The two most common types of corruption are missing values [12, 36] and misalignments [18, 20, 30, 41].

3.1.1 Missing Values

Missing values commonly occur when the spatiotemporal training data is generated from tracking data that has occlusions, such as in human motion and articulatory datasets [15, 36]. For example, if **y** corresponds to an animation sequence of a hand performing fingerspelling, then each dimension in an output frame y can correspond to a specific tracked marker (e.g., the tip of a finger). Such markers naturally become occluded during the course of fingerspelling, which leads to missing values in the resulting **y**.

Input speech: " P R E D I C T I O N "

	Frame	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
(a) **x**	Token	-	p	p	r	ih	ih	d	d	ih	ih	ih	ih	k	k	sh	sh	sh	sh	uh	uh	n	-

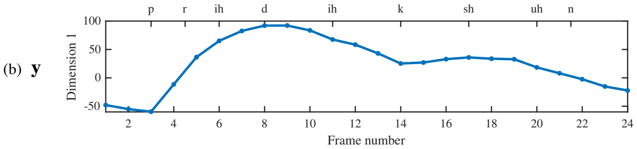

Figure 1: Depicting an example (a) input x and (b) output y for the application of visual speech animation. Each dimension of y corresponds to a parameter of a face model. Only the first dimension of y is depicted.

For any specific output frame y, the corresponding (partially) corrupted $\tilde{y}$ can be defined element-wise as:

$$\tilde{y}^{(d)} = \begin{cases} ? & \text{if } y^{(d)} \text{ is missing} \\ y^{(d)} & \text{otherwise} \end{cases},$$

where ? denotes a missing value that could take on any real value.

More generally, one could also consider cases where the measurements for the output frames have been corrupted by noise (e.g., due to technology limitations), which leads to $\tilde{y}$ being defined as:

$$\tilde{y}^{(d)} = y^{(d)} + \epsilon,$$

for independently distributed random noise variables ϵ.

3.1.2 Misalignments

Misalignments can arise due to imperfections in the tracking technology for generating the spatiotemporal training data [30], or from natural temporal variability in the phenomenon being studied [18, 20, 41], or both. For simplicity, we restrict ourselves to non-warping misalignments of the output spatiotemporal sequences. For example, if **x** corresponds to an audio sequence and **y** corresponds to the associated animation sequence, then **x** and **y** may not be perfectly aligned frame-by-frame.

For any **y**, the corresponding $\tilde{\mathbf{y}}$ would be

$$\tilde{\mathbf{y}} = \texttt{shift}_k(\mathbf{y}),$$

where $\texttt{shift}_k(\mathbf{y})$ is a shift operator that simply shifts the frames of **y** such that $\tilde{y}_i = y_{i-k}$. We deal with boundary cases by padding the start and end of the spatiotemporal sequence **y**.[2]

More generally, one could also consider cases where the output sequences have been warped due to natural human variation or imperfections in performing certain actions [5, 20]. For example, different people may form somewhat different lip shapes while speaking the same sentence. In that sense, one can consider all such observed trackings $\tilde{\mathbf{y}}$ as some warping of an unobservable gold standard animation sequence **y**.

4. DECISION TREE FRAMEWORK

Sequence prediction problems are distinguished from unstructured prediction problems (e.g., univariate regression or classification) due to the assumption that there are salient dependencies

[1] In general, one could employ any convex error function without significant modification to our approach.

[2] Such practices are common in, e.g., animation (where a still pose is maintained at the start and end of the tracked sequence) and audio synthesis (where silence is maintained at the start and end of the output sequence).

between (often nearby) frames in the input and output sequences. For example, in part-of-speech tagging, a good sequence predictor should directly model the fact that two verbs rarely occur in adjacent frames in the output sequence. Similarly for visual speech animation, a good spatiotemporal sequence predictor should directly model the fact that certain vowels or consonant-vowel transitions correspond to particular animation trajectories of a face model.

Our goal is to utilize powerful non-parametric predictors such as decision trees [21, 24] for spatiotemporal sequence prediction. However, decision trees are difficult to apply directly to sequence prediction because they cannot naturally model variable-length prediction problems. As such, we utilize a decompositon or reduction approach that decomposes the spatiotemporal sequence prediction problem to a series of simpler prediction problems, each of which can be naturally modeled using a base decision tree predictor.

For the remainder of this section, we will first describe in Section 4.1 the specifics of the decomposition, and in Section 4.2 the base decision tree model class we used. We present in Section 4.3 an extension of our framework to deal with corrupted training data using latent variables, and discuss in Section 4.4 how to extend our approach to use random forests instead of single decision trees.

4.1 Sliding Window Decomposition

We decompose the spatiotemporal sequence prediction problem into a series of overlapping fixed-length prediction problems. We first make the following observations:

OBSERVATION 1. *Spatiotemporal sequences can exhibit a wide range of context-dependent curvature along the temporal domain (i.e., a wide range of context-dependent "temporal curvature"). For instance, in Figure 1(b), the output sequence can vary smoothly or sharply depending on the input phonetic sequence.*

OBSERVATION 2. *For many domains, one only need to model a sufficiently large local neighborhood in order to produce accurate spatiotemporal output sequences. For instance, in Figure 1(b), the temporal curvature of* **y** *in frames 1-5 do not depend on frames 16-20 in neither* **x** *nor* **y**.

Observation 1 suggests that our base predictor should directly model multiple output frames jointly. Observation 2 suggests that one need not model long-range dependencies in order to produce accurate predictions. We thus define our base model as a decision tree h that maps length-K_x inputs $\hat{\mathbf{x}}_j$ to length-K_y outputs $\hat{\mathbf{y}}_j$, i.e., $\hat{\mathbf{y}}_j \equiv h(\hat{\mathbf{x}}_j)$. I.e., if each frame $y_j \in \Re^D$ is a D-dimensional output, then h is a $(D \times K_y)-$variate regression model.

For input **x**, our prediction procedure can be described as:

1. Decompose **x** into a series of overlapping fixed-length inputs $\langle \hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_T \rangle$ by running Algorithm 1 using **x** and window size K_x.
2. For each $\hat{\mathbf{x}}_j$, predict a fixed-length $\hat{\mathbf{y}}_j$ using a decision tree base model h, resulting in a series of overlapping length-K_y outputs $\langle \hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_T \rangle$ where each $\hat{\mathbf{y}}_j \equiv h(\hat{\mathbf{x}}_j)$.
3. Construct the final output **y** by blending together $\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_T$ using the frame-wise mean.

Figure 2 depicts the prediction procedure for the setting of visual speech animation for $K_x = 5$ and $K_y = 5$. The key remaining step is to decide how to specify (i.e., train) the decision tree model h that maps length-K_x inputs $\hat{\mathbf{x}}_j$ to length-K_y outputs $\hat{\mathbf{y}}_j$, which we discuss in Section 4.2.

The sliding window decomposition offers three key benefits. First, using fixed-length prediction results in dramatically more tractable machine learning problems compared to directly modeling variable-length predictions, and allows us to employ highly complex models such as decision trees (see Section 4.2). Second, so long as the fixed-length subsequences are sufficiently large, one can still make very accurate predictions. The main requirements are that the input length K_x be large enough to capture the salient context, and that the output length K_y be large enough to capture the salient local curvature of **y**. However, the larger that K_x and K_y are, the more training data is required to learn an accurate model, since the intrinsic complexity of the decision tree model class (and thus risk of overfitting to a finite training set) increases with K_x and K_y. Finally, prediction is very efficient, since computationally intensive decoding procedures such as dynamic programming are avoided.

Algorithm 1 Creating Overlapping Fixed-Length Subsequences

```
input: $\mathbf{a}$, $K$                       // a sequence and a window size
$R \leftarrow (K-1)/2$                        // K is assumed to be odd
$T \leftarrow |\mathbf{a}|$
$\hat{S} \leftarrow \emptyset$
for $t = R+1, \ldots, T-R$ do
    Create fixed-length output $\hat{\mathbf{a}} \leftarrow \langle \mathbf{a}_{t-R}, \ldots, \mathbf{a}_{t+R} \rangle$
    $\hat{S} \leftarrow \hat{S} \cup \{\hat{\mathbf{a}}\}$
end for
return: $\hat{S}$
```

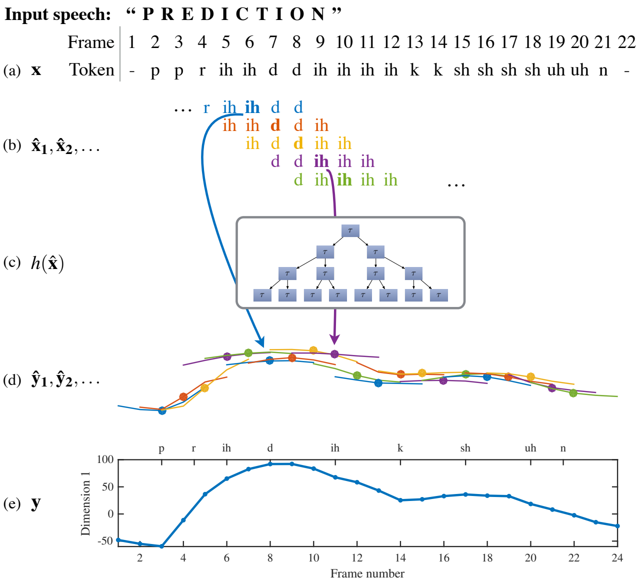

Figure 2: Depicting decision tree prediction pipeline. (a) We start with a frame-by-frame input sequence x (e.g., a phonetic sequence). (b) We convert x into a sequence of overlapping fixed-length inputs $(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots)$. (c) We apply our learned decision tree to predict on each $\hat{\mathbf{x}}_i$, which (d) results in a sequence of overlapping fixed-length outputs $(\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots)$. (e) We finally blend $(\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots)$ by averaging frame-wise to arrive at our final output sequence y. Only the first dimension of y is shown for clarity.

4.2 Decision Tree Base Model

Decision trees are amongst the most widely used methods for discriminative classification and regression tasks [21, 24]. Decision trees are typically specified as binary tree-structured models, and predictions are made via traversing from the root node to a leaf node. Each internal node is associated with a binary query on the input features (e.g., is the fifth input feature non-negative?), with a positive query response leading to traversing one subtree, and a negative response leading to the other subtree. Each leaf node is associated with a static prediction. One can thus think of decision trees as a piece-wise static (w.r.t. the input features) function class that can approximate almost any prediction function.

For our setting, our base decision tree model h is a sliding window predictor that predicts $(D \times K_y)$ outputs in each leaf node (D outputs for each of K_y frames). In essence, h is a $(D \times K_y)-$variate regression model. For example, in our application in visual speech animation, $D = 30$ and $K_y = 5$, leading to a 150-variate regression problem.

We train h by creating a new training set $\hat{S}$ of fixed-length input/output pairs. For a specified input length K_x and output length K_y, we run Algorithm 1 over every $\mathbf{x}$ and $\mathbf{y}$ in S (2) to yield:

$$\hat{S} = \{(\hat{\mathbf{x}}_j, \hat{\mathbf{y}}_j)\}_{j=1}^{\hat{N}}. \tag{5}$$

Note that if the average length of $(\mathbf{x}, \mathbf{y}) \in S$ is T, then $\hat{N} \approx TN$. We can now rephrase our original learning goal (3) as finding an h to minimize the error on the new training set $\hat{S}$ (5):

$$\text{argmin}_{h\in\mathcal{H}}\, L_{\hat{S}}(h) \equiv \text{argmin}_{h\in\mathcal{H}} \sum_{(\hat{\mathbf{x}},\hat{\mathbf{y}})\in\hat{S}} \ell(h(\hat{\mathbf{x}}), \hat{\mathbf{y}}), \tag{6}$$

where $\mathcal{H}$ is the class of $(D \times K_y)-$dimensional multivariate regression trees, subject to some regularization constraint. Since decision trees define a partitioning over the training data $\hat{S}$, in practice, we regularize by enforcing a minimal leaf-node size N_{min}. Decision trees are typically trained via top-down induction (cf. [21, 24]) to greedily minimize the training loss.

Let $\hat{\mathbf{Y}}$ denote a set of training labels $\hat{\mathbf{y}}$ that was partitioned to the same leaf node in h, and $\hat{\mathbf{X}}$ the corresponding inputs. Thus, we have that $h(\hat{\mathbf{x}})$ is the same for every $\hat{\mathbf{x}} \in \hat{\mathbf{X}}$, and the prediction of that leaf node is exactly the mean of $\hat{\mathbf{Y}}$ (in order to minimize squared loss). As such, one can rewrite the squared loss over any given leaf node of h with training labels $\hat{\mathbf{Y}}$ as the unnormalized variance:

$$\begin{aligned} L^2(\hat{\mathbf{Y}}) &= |\hat{\mathbf{Y}}|\texttt{variance}(\hat{\mathbf{Y}}) \\ &= |\hat{\mathbf{Y}}| \sum_{t=1}^{K_y}\sum_{d=1}^{D} \texttt{variance}\left(\left\{ \hat{y}_{i,t}^{(d)} \middle| \hat{\mathbf{y}}_i \in \hat{\mathbf{Y}} \right\}\right), \end{aligned} \tag{7}$$

where $\hat{y}_{i,t}^{(d)}$ is the d-th dimension of t-th frame of $\hat{\mathbf{y}}_i$, and $|\hat{\mathbf{Y}}|$ corresponds to the number of training labels in $\hat{\mathbf{Y}}$. Our learning goal (6) then can be rewritten as:

$$\text{argmin}_{h\in\mathcal{H}}\, L^2_{\hat{S}}(h) \equiv \text{argmin}_{h\in\mathcal{H}} \sum_{\hat{\mathbf{Y}}\in\texttt{partitioning}(\hat{S},h)} L^2(\hat{\mathbf{Y}}), \tag{8}$$

where $\texttt{partitioning}(\hat{S}, h)$ corresponds to a partitioning of the training labels according to the leaf nodes of h. The variance formulation defined in (7) and (8) will be more convenient when developing our latent variable extension in Section 4.3.

4.3 Latent Variable Extension

We now describe a latent variable extension to accommodate corrupted training data as described in Section 3.1. Each corrupted piece of training data is associated with a latent variable z that corresponds to the missing or corrupted information. In the case of missing values, each z corresponds to one specific missing value. In the case of misalignment, each z corresponds to the correct alignment of each complete input/output sequence pair $(\mathbf{x}, \mathbf{y})$.

Algorithm 2 Training Procedure for Latent Variable Framework

```
 1: input: $S$                          // with missing values
 2: input: $K_x, K_y$                    // input and output window size
 3: input: $N_{min}$            // minimum leaf node size of base model
 4: init: $\hat{S}$ using Algorithm 1
 5: init: $Z$ to default values
 6: repeat
 7:     train $h$ to minimize (9)
 8:     repeat
 9:         for $z \in Z$ do
10:             infer $z$ to minimize (9)
11:         end for
12:     until convergence
13: until convergence
14: return: $h$, $Z$
```

The goal of the latent variable extension is to jointly infer the z's while training a good decision tree h. Let $\hat{S}$ denote the decomposed version of the original training set S into overlapping fixed-length subsequences, and let Z denote the set of all latent variables. Given Z, one can produce a "cleaned" version of the training set, $\Phi(\hat{S}, Z)$. We instantiate Φ for missing values in Section 4.3.1 and for misalignments in Section 4.3.2. We can now extend our decomposed learning objective (8) to include latent variables Z:

$$\text{argmin}_{Z,h\in\mathcal{H}}\, L^2_{\Phi(\hat{S},Z)}(h), \tag{9}$$

where the goal now is to jointly learn h and infer Z. Note that when the training set S is not corrupted, then (9) reduces to (8) since there are no latent variables (i.e., $Z = \emptyset$), and $\Phi(\hat{S}, Z) = \hat{S}$ is just the identity function.

In practice, we solve (9) via alternating optimization, as described in Algorithm 2. We first set Z to some default value (Line 4), and use the resulting $\Phi(\hat{S}, Z)$ to train a decision tree h as decribed in Section 4.2 (Line 6).[3] Given a trained h, we then infer a better Z via coordinate descent on each $z \in Z$ by finding a "cleaner" $\Phi(\hat{S}, Z)$ that minimizes the loss of the leaf nodes of h (Lines 7-11).

The main novelty of our latent variable approach is that we can exploit a specific property that arises from combining our sliding window decomposition with a decision tree base model. In particular, our decomposition essentially creates K_y copies of each output frame y, which results in the same piece of corrupted training data z being placed into many different leaf nodes of the decision tree h. As a consequence, we can utilize the other training data in these leaf nodes to estimate a better z that can improve the purity (i.e., decrease the variance) of the leaf nodes in h. We elaborate on this point below for missing values and misalignments.

4.3.1 Missing Values

In the missing values setting, each $z \in Z$ corresponds to a specific missing measurement of some output frame $\tilde{y}^{(d)}$ in the original training set S. At a high level, we infer z by averaging over all the labels that belong to the same leaf nodes of h that z belongs to (see (11) below). Figure 3 shows an illustration of this procedure.

For this section, we use the index i to refer to a specific partially corrupted training example $\tilde{\mathbf{y}}_i$ from the original training set S, index t to refer to a specific frame $\tilde{y}_{i,t}$ of $\tilde{\mathbf{y}}_i$, and index d to refer to a specific dimension of the output frame $\tilde{y}_{i,t}^{(d)}$. We use the notation $z_{i,t}^{(d)}$ to refer to the z that corresponds to $\tilde{y}_{i,t}^{(d)}$. In other words,

[3] In the first iteration, we often train an overly regularized h (i.e., with larger N_{min}) to use for inferring the first set of "cleaner" Z.

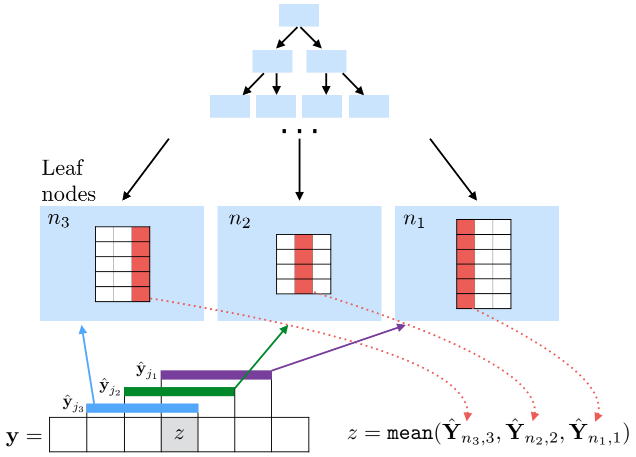

Figure 3: Depicting latent variable inference in the case of missing values. Each latent variable z is associated with a specific missing entry. Thus, each z is associated with multiple fixed-length subsequences, which then become partitioned to different leaf nodes. Estimating the latent variable z associated with the missing value is simply the mean over all the training labels in those leaf nodes. See Section 4.3.1 for more details.

following the definition in Section 3.1.1, we have a $z_{i,t}^{(d)}$ whenever $\tilde{y}_{i,t}^{(d)}$ =? in the training set S.

The sliding window decomposition described in Section 4.1 creates K_y copies of each $\tilde{y}_{i,t}^{(d)}$ when generating the new training set $\hat{S}$ (5) composed of overlapping fixed-length subsequences. In particular for some set of indices $j_1^i, \ldots, j_{K_y}^i$ on the decomposed training set $\hat{S}$, $\tilde{y}_{i,t}$ will correspond to the 1st frame of $\hat{\mathbf{y}}_{j_1^i}$, the 2nd frame of $\hat{\mathbf{y}}_{j_2^i}$, and so forth. Thus, the K_y copies of $\tilde{y}_{i,t}^{(d)}$ are located at:

$$\tilde{y}_{i,t}^{(d)} \equiv \hat{y}_{j_1^i,1}^{(d)} \equiv \hat{y}_{j_2^i,2}^{(d)} \equiv \ldots \equiv \hat{y}_{j_{K_y}^i,K_y}^{(d)}. \qquad (10)$$

We define $\Phi(\hat{S}, Z) \rightarrow \{(\hat{\mathbf{x}}_j, \hat{\mathbf{y}}_j)\}_{i=1}^{\hat{N}}$ as imputing the missing values in $\hat{S}$ using the corresponding latent variables, i.e.,

$$\hat{y}_{j_t^i,t}^{(d)} = \begin{cases} z_{i,t}^{(d)} & \text{if } \tilde{y}_{i,t}^{(d)} = \ ? \\ \tilde{y}_{i,t}^{(d)} & \text{otherwise} \end{cases} .$$

For the remainder of this section, we focus on optimizing (9) for a single latent variable $z_{i,t}^{(d)}$ (i.e, Lines 9-11 in Algorithm 2). We observe that inferring the $z_{i,t}^{(d)}$ that minimizes (9) only depends on the leaf nodes affected by $z_{i,t}^{(d)}$, which can easily be seen via the definition of (9) as the sum of (7) over each leaf node.

Following the notation in Section 4.2, we use $\hat{\mathbf{Y}}$ to refer to the training labels from $\hat{S}$ that is partitioned to a specific leaf node in the current decision tree h. Let $n_1, \ldots, n_{K_y}$ denote the indices of the leaf nodes that data points $\hat{\mathbf{y}}_{j_1^i}, \ldots, \hat{\mathbf{y}}_{j_{K_y}^i}$ were partitioned to in h. In other words, $\hat{y}_{j_1}$ was partitioned to leaf node n_1, $\hat{y}_{j_2}$ was partitioned to leaf node n_2, and so forth. We will refer to the training labels partitioned to each of these leaf nodes as $\hat{\mathbf{Y}}_{n_1}, \ldots, \hat{\mathbf{Y}}_{n_{K_y}}$.

Choosing $z_{i,t}^{(d)}$ to minimize the unnormalized variance (7) of the leaf nodes $n_1, \ldots, n_{K_y}$ yields:

$$z_{i,t}^{(d)} = \texttt{mean}\left(\cup_{k=1}^{K_y} \left\{ \hat{y}_{m,k}^{(d)} \middle| \hat{\mathbf{y}}_m \in \hat{\mathbf{Y}}_{n_k} \setminus \hat{\mathbf{y}}_{j_k^i} \right\}\right). \qquad (11)$$

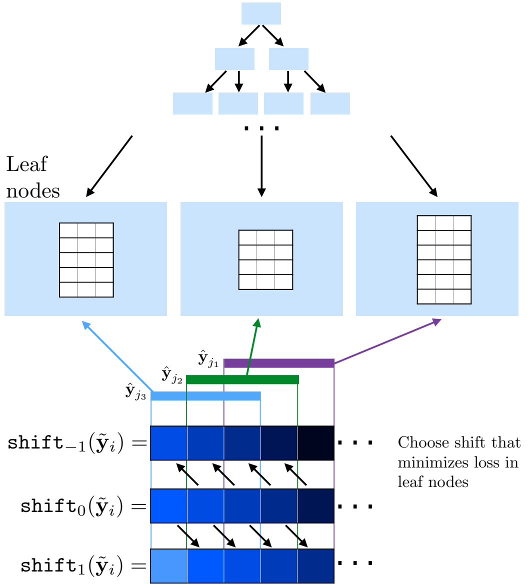
Leaf nodes
$\hat{\mathbf{y}}_{j_1}$
$\hat{\mathbf{y}}_{j_2}$
$\hat{\mathbf{y}}_{j_3}$
$\texttt{shift}_{-1}(\tilde{\mathbf{y}}_i) =$
$\texttt{shift}_{0}(\tilde{\mathbf{y}}_i) =$
$\texttt{shift}_{1}(\tilde{\mathbf{y}}_i) =$
Choose shift that minimizes loss in leaf nodes

Figure 4: Depicting latent variable inference in the case of misalignments. Each latent variable z defines a global shift of an original complete output sequence y. Latent variable inference corresponds to choosing the best value of z such that the resulting sliding window subsequences have lowest training loss in the affected leaf nodes. See Section 4.3.2 for more details.

In other words, $z_{i,t}^{(d)}$ is the average over all other training labels in the leaf nodes that contain a copy of $y_{i,t}^{(d)}$ (10) . Note for a fixed decision tree h, imputing the missing value of $y_{i,t}^{(d)}$ does not change which leaf node each $\hat{y}_{j_t^i}$ belongs to, since that is determined based solely on the input $\hat{\mathbf{x}}_{j_t^i}$ which was not modified.

In practice, we down-weight the importance of training labels imputed using latent variables when computing the loss function (7), so that they have smaller influence relative to training labels that are observed. In Line 4 of Algorithm 2, we initialize the weights of the missing values to 0 so that they have no influence when training the first decision tree (i.e., the first trained decision tree ignores missing values). After one round of missing value imputation (Lines 7-11), we set the weights of the training labels imputed using latent variables Z to some value between 0 and 1.

4.3.2 Misalignments

In the misalignment setting, each $z \in Z$ corresponds to a shift (e.g., 2 frames to the left) of a potentially misaligned spatiotemporal output sequence $\tilde{\mathbf{y}}$ in the original training set S. At a high level, we infer each z by trying all possible shifts of $\tilde{\mathbf{y}}$ in order to minimize the unnormalized variance of the leaf nodes of h that z belongs to. Figure 4 shows an illustration of this procedure.

Similar to Section 4.3.1, we use the index i to refer to a specific training example $\mathbf{y}_i$ in the original training set S. Following the definition in Section 3.1.2, we use the notation z_i to refer to the z that corresponds to i-th training label sequence $\tilde{\mathbf{y}}_i$ in S. Let T_i denote the length (i.e., the number of frames) of $\tilde{\mathbf{y}}_i$. We use the indices $j_1^i, \ldots, j_{T_i}^i$ on the decomposed training set $\hat{S}$ to refer to the $\hat{\mathbf{y}}$ that correspond to some subsequence of $\mathbf{y}_i$.

We define $\Phi(\hat{S}, Z) \rightarrow \{(\hat{\mathbf{x}}_j, \hat{\mathbf{y}}_j)\}$ as shifting the output subsequences in $\hat{S}$ according to the corresponding latent variables. Pro-

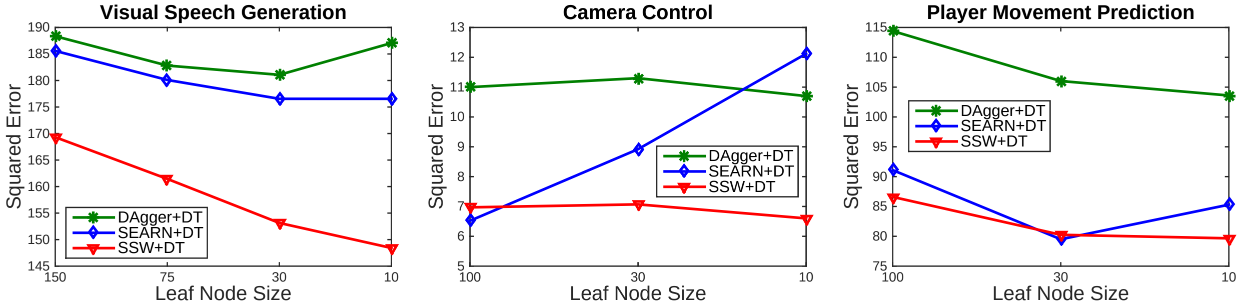

Figure 5: Showing a comparison of SSW+DT with SEARN and DAgger over a range of minimum leaf node sizes N_{min} for the three benchmark datasets. We see that SSW+DT consistently outperforms or is competitive with SEARN and DAgger.

cedurally, $\Phi(\hat{S}, Z)$ is generated by running Algorithm 1 on the shifted training set $\{(\mathbf{x}_i, \texttt{shift}_{z_i}(\tilde{\mathbf{y}}_i))\}_{i=1}^{N}$.

For the remainder of this section, we focus on optimizing (9) for a single z_i (i.e., Line 9 in Algorithm 2). Similar to Section 4.3.1, inferring z_i only depends on the leaf nodes affected by z_i. Thus, we simply choose the value of z_i that minimizes the unnormalized variance of the affected leaf nodes. Each choice of z_i generates a new set of subsequences $\hat{\mathbf{y}}_{j^i_1}, \ldots, \hat{\mathbf{y}}_{j^i_{T_i}}$ that is a shifted version of the default. Note for a fixed decision tree h, shifting the output subsequences $\hat{\mathbf{y}}_{j^i_1}, \ldots, \hat{\mathbf{y}}_{j^i_{T_i}}$ does not change which leaf node each $\hat{\mathbf{y}}_{j^i_t}$ belongs to, since that is determined based solely on the input $\hat{\mathbf{x}}_{j^i_t}$ which was not modified.

In essence, the latent variables Z are trying to correct any frame-wise misalignments between the input and output sequences. We chose a default of $z_i = 0$, which is used in Line 4 in Algorithm 2.

4.4 Using Random Forests

Ensembles of decision trees, such as Bagging [2] and Random Forests [3], can often improve upon the accuracy of a single decision tree. Furthermore, Random Forests are particularly useful for inputs that contain real-valued attributes, since randomly sampling the splitting criteria is an attractive alternative to exhaustively iterating over all possible splitting criteria (which tends to grow proprotional to the training set size for real-valued inputs).

Our framework extends in a straightforward manner to ensemble decision tree base models. In the case without latent variables, one simply replaces the decision tree base model with an ensemble. In the case with latent variables, inferring the latent variables Z simply requires considering more leaf nodes from multiple trees.

5. BENCHMARK EXPERIMENTS

We evaluate our approach, which we refer to as SSW+DT ("Spatiotemporal Sliding Window with Decision Trees"), using a number of benchmark datasets. We take the view of spatiotemporal sequence prediction as a high-dimensional regression problem, and thus evaluate primarily using squared error.

5.1 Datasets

Visual Speech Animation. Our main benchmark dataset is the KB-2k visual speech dataset from [33]. KB-2k is a large audio-visual dataset containing an actor speaking approximately 2500 sentences in a neutral tone while having his face tracked. The inputs **x** are phoneme sequences (sampled at 30 Hz), and the outputs **y** are parameter sequences of a 30-dimensional Active Appearance Model [23] of the actor's lower face (also sampled at 30 Hz). Following the setup in [33], we used 50 sentences for testing and the rest for training. For SSW+DT, we use $K_x = 11$ and $K_y = 5$, which results in a 150-variate base regression model. The total size of the decomposed training set is $|\hat{S}| \approx 200K$.

Automated Camera Control. Our next dataset is the camera control dataset from [4]. The input sequences **x** are noisy detections of basketball players on a basketball court. The output sequences **y** are the tracked pan-angle states of a broadcast camera operated by a human expert to track the interesting action on the court. The dataset comprises seventeen minutes worth of frames sampled at 25 Hz. Following the setup in [4], we use 16 minutes for training and the remaining minute for testing. For SSW+DT, we use $K_x = 11$ and $K_y = 5$, which results in a 5-variate base regression model. The total size of the decomposed training set is $|\hat{S}| \approx 34k$.

Team Sports Player Movement Prediction. Our final dataset is a player position prediction dataset derived from [39]. The input sequences **x** are the tracked positions of nine basketball players on the basketball court. The output sequences **y** are the tracked position of the remaining basketball player (who is the ballhandler). The data comprises 2600 half court possessions sampled at 5 Hz. We use 2000 posssesions for training and 600 possesions for testing. For SSW+DT, we use $K_x = 11$ and $K_y = 5$, which results in a 10-variate base regression model. The total size of the decomposed training set is $|\hat{S}| \approx 77K$.

5.2 Evaluating Prediction Quality

We first evaluate the effectiveness of SSW+DT on uncorrupted training data (see Section 5.3 for experiments on corrupted training data). We primarily compare against existing decision tree based decomposition approaches for sequence prediction, such as SEARN [7] and DAgger [25].

SEARN and DAgger are designed to control for cascading error effects from long range dependencies, and are complementary to our SSW+DT approach which instead focuses on accurately capturing the local temporal curvature of spatiotemporal output sequences. SEARN and DAgger both define a self-recurrent base model that predicts a single frame y using both the standard input subsequence $\hat{\mathbf{x}}$ as well as the previously predicted frames for the entire input sequence **x**. For both SEARN and DAgger, the base decision tree model takes as input the exact same K_x input subsequence as SSW+DT, as well as the previous K_y predicted frames. Note that, because of their self-recurrent definition, SEARN and DAgger require iterative training of the base model, and thus take an order of magnitude longer to train than SSW+DT. All three approaches have the same computational costs at test time.[4]

Figure 5 shows the test results. We see that SSW+DT consistently outperforms or is competitive with DAgger and SEARN

[4] Although the prediction procedure of SSW+DT is more easily parallelized than that of SEARN and DAgger.

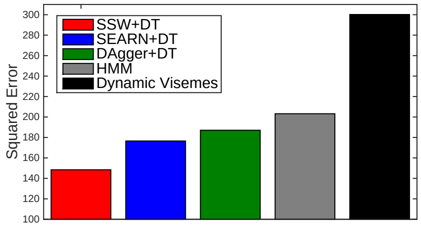

Figure 6: Showing a comparison with previous state-of-the-art methods for visual speech generation. For decision tree based approaches (SSW+DT, DAgger+DT, SEARN+DT), the base model was trained using minimum leaf node size $N_{min} = 10$. We see that SSW+DT significantly outperforms Dynamic Visemes[33] and HMM-based approaches [40].

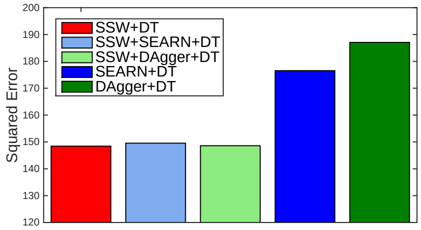

Figure 7: Evaluating the performance of combining SSW+DT with DAgger or SEARN on the visual speech dataset. For all methods, the base decision tree model was trained using minimum leaf node size $N_{min} = 10$. We see that combining SSW+DT with other decomposition approaches does not improve performance.

(while being significantly faster to train). The gains of SSW+DT are particularly notable on our main benchmark dataset for visual speech generation, where even a heavily regularized SSW+DT outperforms every version of SEARN and DAgger.

Figure 6 shows a comparison on the visual speech dataset with existing state-of-the-art visual speech approaches, such as Dynamic Visemes [33] and HMM-based approaches [40]. We observe that SSW+DT substantially outperforms all baselines. In Section 6, we provide additional evidence of the practicality of our approach via a user preference study on the generated animation sequences.

5.2.1 Combining Decomposition Frameworks

Since SEARN and DAgger employ complementary decomposition approaches to the decomposition employed by SSW+DT, we also evaluate combining SSW+DT with SEARN and DAgger. We combine in the straightforward way: the decision tree base model is a recurrent sliding window predictor that predicts a length-K_y subsequence using both the standard input subsequence $\hat{\mathbf{x}}$ as well as the previous K_y predicted frames. In other words, combining SSW+DT with SEARN or DAgger results in a multi-frame extension of conventional SEARN or DAgger, respectively.

Figure 7 shows the results for our main benchmark setting of visual speech. We used base models trained with minimum leaf node size $N_{min} = 10$. We observe that combining SSW+DT with SEARN or DAgger does not improve performance.

5.3 Evaluating Robustness to Corrupted Training Data

We now evaluate the robustness of our latent variable extension, which we refer to as SSWL+DT, to corrupted training data. We evaluate for both missing values and misalignments, as described in Section 3.1. We focus on the visual speech dataset for this analysis.

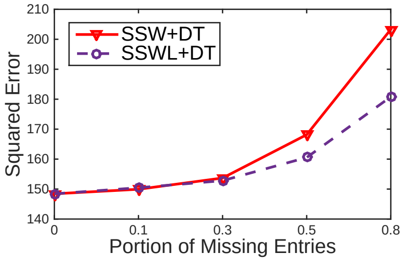

Figure 8: Showing squared error on the visual speech dataset with missing entries in the training labels. We see that SSWL+DT can robustly tolerate many missing values.

Table 1: Evaluating the ability of SSWL+DT to infer the missing entries in the training set. The table below shows the mean squared error of SSW+DT and SSWL+DT when predicting on every missing entry in the training set.

Frac. w/ Missing Entries	0.1	0.3	0.5	0.8
SSW+DT	184.42	191.32	206.97	235.82
SSWL+DT	185.21	186.02	189.24	199.24

5.3.1 Missing Values

For the missing values setting, we randomly remove a fraction of the training labels from the training set. Specifically, each output frame y is a 30-dimensional output, and each dimension of each frame is independently eligible for being missing. We instantiate SSWL+DT as described in Section 4.3.1.

Figure 8 shows the results comparing SSWL+DT with SSW+DT (which ignores missing values during training). We observe that SSWL+DT can more robustly tolerate a substantial amount of missing entries in the training set. Table 1 analyzes how well SSWL+DT can infer the missing entries in the training set, where we again observe that SSWL+DT is more robust than SSW+DT.

5.3.2 Misalignments

For the misalignments setting, we randomly choose training sentences to misalign, and we randomly shift a misaligned training sentence by one of $\{-3, -2, -1, +1, +2, +3\}$ frames. We instantiate SSWL+DT as described in Section 4.3.2, and we specify the range of each latent variable as $z \in [-3, +3]$.[5]

Figure 9 shows the results comparing SSWL+DT with SSW+DT (which assumes that all sentences are correctly aligned during training). We observe that SSWL+DT is surprisingly robust to misalignments in the training set, and actually achieves slightly better performance than SSW+DT on the uncorrupted training set (although the difference is not statistically significant). One possible interpretation is that the visual speech dataset from [33] actually does suffer from a small degree of misalignment in the data generation process. Table 2 analyzes how well SSWL+DT can infer the correct alignments in the training set (assuming that the gold standard is properly aligned), and we see that SSWL+DT is able to recover a substantial fraction of the correct alignments.

6. VISUAL SPEECH USER STUDY

While squared error is a standard measure of prediction quality, it may not be fully indicative of which method achieves better performance in the target application domain. For instance, one

[5] When the range of misalignments is not known exactly, one can conservatively overspecify the the range of z, e.g., $z \in [-5, +5]$.

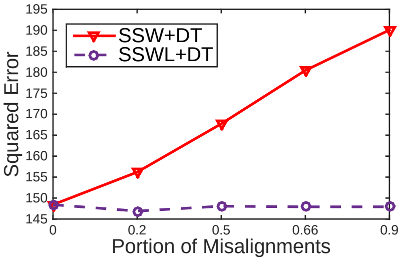

Figure 9: Showing squared error on the visual speech dataset with misalignments in the training labels. We see that SSWL+DT is extremely robust to misalignments.

Table 2: Showing the fraction of training sentences in visual speech dataset whose alignment z_i was correctly inferred by SSWL+DT. We see that SSWL+DT is able to correctly infer a substantial fraction of correct alignments in the training data.

Fraction with Misalignments	0.2	0.5	0.66	0.9
Fraction Correctly Aligned	0.78	0.78	0.79	0.78

animation sequence might suffer higher squared error than another animation sequence, but is more appealing visually. In this section, we provide a complementary evaluation of the visual speech animation setting via a user preference study.

We asked human raters to provide preference judgements between two animations for a common audio input sequence. The two animations are placed side-by-side, thus allowing raters to make a paired frame-by-frame comparison of how well the two animations align with input audio. Figure 10 depicts a screenshot of this setup.

We considered five experimental conditions. Each experiment condition compares our SSW+DT method with an alternative approach, in particular the four baselines evaluated in Section 5.2 as well as the ground truth animations. Each experimental condition was evaluated over all 50 test sentences with five judgements per sentence. The left/right placements were selected randomly.

Table 3 shows the results. A win indicates that our SSW+DT approach accumulated at least 3/5 votes for a test sentence, and a loss indicates the opposite. The vote difference indicates the average difference in votes for SSW+DT and the alternative approach. We see that SSW+DT is significantly preferred over all baselines, which showcases the effectiveness of our approach in a practical setting.[6] The comparison between SSW+DT and the ground truth indicates that there is still significant room for improvement. Interestingly, it appears that the preference gap between SSW+DT and the ground truth is smaller than the preference gap between SSW+DT and the other baselines.

7. LIMITATIONS AND FUTURE WORK

Although our approach showed significant promise in contextual spatiotemporal sequence prediction, there is still significant room for improvement. For instance, it would be interesting to train on significantly larger and more heterogenous datasets (e.g., visual speech that accounts for a variety of tones such as angry and sad).

For many settings, the spatiotemporal output sequences must obey physical constraints. For example, visual speech is typically animated on a virtual or physical rig [33]. Typically, an end-to-end pipeline requires mapping the predicted spatiotemporal sequence to a physically feasible trajectory, which often results in degraded performance. A better approach would be develop a more holistic machine learning approach that can directly learn to predict physically feasible spatiotemporal sequences.

Figure 10: Screenshot of our user study. For each test sentence, we ask raters to view two competing visual speech animations for the same audio sequence, and judge which appears more natural. The left/right placement is randomly chosen.

Table 3: User study results for visual speech animation. For each comparison, we ask five raters to view two competing animations for the same audio sequence (see Figure 10), and judge which animation appears more natural. Our SSW+DT approach significantly outperforms all competing methods, although there is still a sizeable gap versus the ground truth. All results are 99% significant using a two-tailed signed-rank test.

Comparison	Win/Loss	Vote Diff
SSW+DT vs Dynamic Visemes [33]	47 / 3	3.32
SSW+DT vs HMM [40]	50 / 0	3.76
SSW+DT vs SEARN+DT [7]	44 / 6	1.96
SSW+DT vs DAgger+DT [25]	45 / 5	2.12
SSW+DT vs Ground Truth	10 / 40	-1.68

In some sense, our latent variable approach only considered the simplest types of corrupted training data. As such, another interesting direction of research is to consider more complex forms of corrupted training data, such as large contiguous regions of occlusion [36], severe measurement error generating the output labels, and warping effects [20].

A related issue is the fact that spatiotemporal sequence labels are often generated from human demonstrations, and thus exhibit natural human variations. One interesting direction is to treat the gold standard demonstration as a latent variable, which can be inferred from imperfect demonstrations (cf. [5]).

Finally, our experiments showed that combining our SSW-DT approach with SEARN [7] and DAgger [25] did not yield improved performance. SEARN and DAgger focus on controlling for cascading error effects due to longer range dependences, and it would be interesting to identify and formally characterize settings that might enjoy improved performance from inheriting the strengths of both SSW-DT as well as SEARN & DAgger.

8. CONCLUSIONS

We have presented a discriminative learning approach for spatiotemporal sequence prediction. Our approach employs a sliding window decomposition that enables using powerful non-parametric base models such as decision trees. We also presented a latent variable extension that robustly tolerates corrupted training labels such as missing values and mis-alignments. We demonstrate empirically the benefits of our approach over existing decomposition approaches, as well as the robustness of our latent variable extension. We showcased the practicality of our approach in a user preference study for visual speech animation, where we found our approach to be significantly preferred over several state-of-the-art baselines.

[6] See sample videos at: `http://tinyurl.com/m52p64m`.

References

[1] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *International Conference on Machine Learning (ICML)*, 2003.

[2] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] P. Carr and J. Chen. Mimicking human camera operators. In *IEEE Workshop on Applications of Computer Vision (WACV)*, 2015.

[5] A. Coates, P. Abbeel, and A. Ng. Learning for control from multiple demonstrations. In *International Conference on Machine Learning (ICML)*, 2008.

[6] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[7] H. Daumé III, J. Langford, and D. Marcu. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009.

[8] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[9] J. R. Doppa, A. Fern, and P. Tadepalli. Structured prediction via output space search. *Journal of Machine Learning Research (JMLR)*, 15(1):1317–1350, 2014.

[10] R. Durbin. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[11] E. Eyjolfsdottir, S. Branson, X. Burgos-Artizzu, E. Hoopfer, J. Schor, D. Anderson, and P. Perona. Detecting social actions of fruit flies. In *European Conference on Computer Vision (ECCV)*, 2014.

[12] M. Farhadloo and M. Á. Carreira-Perpinán. Learning and adaptation of a tongue shape model with missing data. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[13] S. Gaffney and P. Smyth. Joint probabilistic curve clustering and alignment. In *Neural Information Processing Systems (NIPS)*, 2004.

[14] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Learning time-series shapelets. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2014.

[15] T. Kim, G. Shakhnarovich, and R. Urtasun. Sparse coding for learning interpretable spatio-temporal primitives. In *Neural Information Processing Systems (NIPS)*, 2010.

[16] P. Kontschieder, S. R. Bulo, H. Bischof, and M. Pelillo. Structured class-labels in random forests for semantic image labelling. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[17] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001.

[18] R. Lajugie, D. Garreau, F. Bach, and S. Arlot. Metric learning for temporal sequence alignment. In *Neural Information Processing Systems (NIPS)*, 2014.

[19] S. Lenser and M. Veloso. Non-parametric time series classification. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[20] J. Listgarten, R. Neal, S. Roweis, and A. Emili. Multiple alignment of continuous time series. In *Neural Information Processing Systems (NIPS)*, 2004.

[21] O. Maimon and L. Rokach. Chaper 9: Decision trees. In *Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

[22] C. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[23] W. Mattheyses, L. Latacz, and W. Verhelst. Comprehensive many-to-many phoneme-to-viseme mapping and its application for concatenative visual speech synthesis. *Speech Communication*, 55(7–8):857–876, 2013.

[24] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[25] S. Ross, G. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[26] S. Sarawagi and W. Cohen. Semi-markov conditional random fields for information extraction. In *Neural Information Processing Systems (NIPS)*, 2004.

[27] D. Schabus, M. Pucher, and G. Hofer. Speaker-adaptive visual speech synthesis in the HMM-framework. In *Interspeech*, 2012.

[28] B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel methods in computational biology*. MIT press, 2004.

[29] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[30] Y. Su, H. Ai, and S. Lao. Real-time face alignment with tracking in video. In *IEEE International Conference on Image Processing (ICIP)*, 2008.

[31] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems (NIPS)*, 2014.

[32] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Neural Information Processing Systems (NIPS)*, 2003.

[33] S. Taylor, M. Mahler, B.-J. Theobald, and I. Matthews. Dynamic units of visual speech. In *ACM/Eurographics Symposium on Computer Animation (SCA)*, 2012.

[34] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2000.

[35] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*, 2004.

[36] W. Wang, R. Arora, and K. Livescu. Reconstruction of articulatory measurements with smoothed low-rank matrix completion. In *Spoken Language Technology Workshop*, 2014.

[37] D. Weiss, B. Sapp, and B. Taskar. Structured prediction cascades. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[38] C.-N. J. Yu, T. Joachims, R. Elber, and J. Pillardy. Support vector training of protein alignment models. *Journal of Computational Biology*, 15(7):867–880, 2008.

[39] Y. Yue, P. Lucey, P. Carr, A. Bialkowski, and I. Matthews. Learning fine-grained spatial models for dynamic sports play prediction. In *IEEE International Conference on Data Mining (ICDM)*, 2014.

[40] H. Zen, T. Nose, J. Yamagishi, S. Sako, T. Masuko, A. Black, and K. Tokuda. The HMM-based speech synthesis system version 2.0. In *Speech Synthesis Workshop*, 2007.

[41] F. Zhou and F. De la Torre. Generalized time warping for multi-modal alignment of human motion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[42] B. Ziebart, A. Dey, and J. A. Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *International Conference on Intelligent User Interfaces (IUI)*, 2012.