PROPS: Probabilistic personalization of black-box sequence models

Michael Thomas Wojnowicz and Xuan Zhao
Department of Research and Intelligence
Cylance, Inc.
Irvine, California 92612
{mwojnowicz, xzhao}@cylance.com

***Abstract*—We present PROPS , a lightweight transfer learning mechanism for sequential data. PROPS learns probabilistic perturbations around the predictions of one or more arbitrarily complex, pre-trained black box models (such as recurrent neural networks). The technique pins the black-box prediction functions to "source nodes" of a hidden Markov model (HMM), and uses the remaining nodes as "perturbation nodes" for learning customized perturbations around those predictions. In this paper, we describe the PROPS model, provide an algorithm for online learning of its parameters, and demonstrate the consistency of this estimation. We also explore the utility of PROPS in the context of personalized language modeling. In particular, we construct a baseline language model by training a LSTM on the entire Wikipedia corpus of 2.5 million articles (around 6.6 billion words), and then use PROPS to provide lightweight customization into a personalized language model of President Donald J. Trump's tweeting. We achieved good customization after only 2,000 additional words, and find that the PROPS model, being fully probabilistic, provides insight into when President Trump's speech departs from generic patterns in the Wikipedia corpus. All code (both the PROPS training algorithm as well as reproducible experiments) are available as a pip-installable Python package.**

I. Introduction

Suppose one has access to one, or possibly more, pre-trained sequence models $\{\mathcal{M}_k\}$ which have been trained in one context, but whose knowledge should be transferred to a related context. Further suppose that one requires the transfer learning to be lightweight and streaming, and that after training, one would like to have a **fully probabilistic sequence model** (i.e one with stochastic hidden variables). The fully probabilistic sequence model would provide real-time probabilistic statements about membership in the previous, reference context versus membership in the new, customized context.

One motivating example is a situation where one can train a single, arbitrarily complex, black-box sequence model in the cloud on a large public corpus, but then would like to "customize" or "personalize" that model to a particular user's private data on an endpoint (laptop, mobile device, etc.). One can imagine this scenario at play for machine learning models for automatic message generation or text-to-speech. In such contexts, a lightweight, streaming transfer learning mechanism would minimize computation and memory footprint at deployment. At the same time, the ability to do inference on hidden state variables would allow one to 'draw attention' to subsequences which deviate from the population-level "normal" behavior, even if they are normal for that particular user.

To solve this problem, we present PROPS , an acronym derived from *PRObabilistically Perturbed Sequence modeling*. PROPS generalizes a streaming training algorithm for a HMM to learn customized perturbations around the predictions of pre-trained black-box sequence models. In particular, the PROPS model takes K_s pre-trained black-box sequence models and pins them at K_s nodes of a HMM wrapper. By "pinning", we mean that the emissions distributions at those K_s **source states** are not updated during training. These source states can provide knowledge from a reference or baseline corpus about arbitrarily complex functional relationships between the sequence observed so far and upcoming observations, because the $\{\mathcal{M}_k\}$'s could be recurrent neural networks (RNNs), such as Long Short-Term Memory (LSTM) networks [1], variations thereof, or anything desired. Indeed, the only constraint is that each such model produces predictive output probabilities, $P(y_{t+1} \mid y_{1:t})$ at each time step.

The remaining K_p nodes serve as the **perturbation states**. For instance, Figure 1 shows a probability simplex supporting the latent state distribution of a PROPS model with 3 latent states. Suppose the top node (colored red) is a single source node, whereas the bottom nodes (colored blue) are the perturbation nodes. The training of PROPS causes the blue nodes to converge to stable emission distributions (just as in a standard HMM), even while the red node is tied to a stochastic process of distributions that varies (potentially wildly) from timestep to timestep. Moreover, the model learns transition dynamics across the full heterogeneous system.

A. Summary of contributions

The PROPS model is a lightweight transfer learning mechanism for sequence data. PROPS learns probabilistic perturbations around the predictions of an arbitrarily complex, pre-trained black box model trained on a reference corpus. The key properties of the PROPS model and training algorithm are:

- **Streaming** - Both training and scoring can be done in a streaming (i.e. online) manner. Among other things, this reduces memory footprint, and keeps the model's scores up-to-date with potential behavioral non-stationaries.

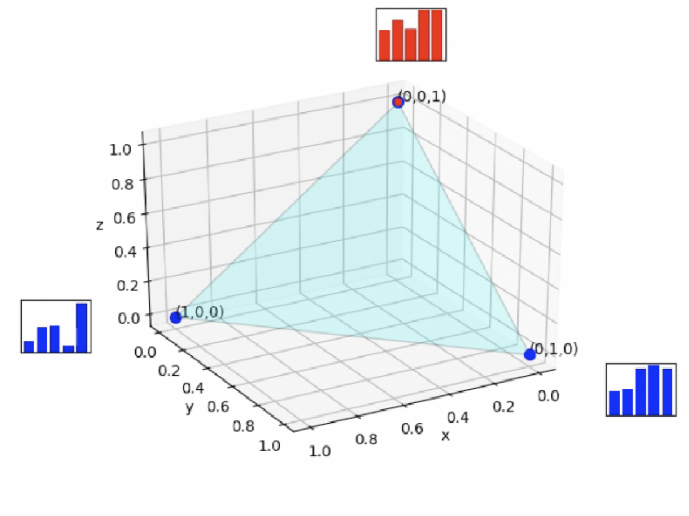

Fig. 1: *A probability simplex supporting the latent state distribution of a* PROPS *model with 3 latent states*

- **Lightweight** - The model can be trained reliably (from being high bias/low variance), has a low computational footprint, and offers quick training times (e.g. see [2], with additional speedups due to the streaming algorithm [3]).
- **Probabilistic/Interpretable** - The model is fully probabilistic. In particular, its hidden states are random variables. Thus inference – such as filtering, smoothing, and MAP estimates – can be performed on the hidden state variables. Because the latent states can be partitioned into nodes related to the original task and nodes related to the new task, we can make probabilistic statements about when behavior drifts from the source or reference context into the customization or transfer context. (Note that anomalies for the PROPS model would be behavior that departs from both contexts, or which moves between contexts in unexpected ways.)
- **Privacy-Preserving** - Training data can be kept private while PROPS provides customization.

In this paper, we describe the PROPS model, provide an algorithm for fitting it, corroborate the consistency of its estimators, and explore its utility in the context of personalized language modeling.

II. Models

A. Hidden Markov Model

A hidden Markov model (HMM) is a bivariate stochastic process $(X_t, Y_t)_{t\geq 0}$ where:

1) $X = (X_t)_t$ is an unobserved ergodic Markov chain (often called a hidden state sequence) over $\{1, \ldots, K\}$ with transition matrix τ and initial state distribution π.
2) $Y = (Y_t)_t$ is observed data, independent conditional on X with parametric emission distribution $(Y_t \mid X_t = k) \sim \epsilon^{\lambda_k}$.

The complete data likelihood for a HMM is

$$P(X, Y \mid \theta) = P(X_1 \mid \theta)P(Y_1 \mid X_1, \theta) \prod_{t=2}^{T} P(X_t \mid X_{t-1}, \theta)P(Y_t \mid X_t, \theta) \tag{1}$$

where parameter $\theta = (\pi, \tau, (\lambda_k)_k)$.

B. PROPS Model

A probabilistically perturbed sequence (PROPS) model is a generalization of HMM where deterministic predictive probability functions $(f^k_{Y_{1:t-1}}(Y_t))_k$ are supplied as inputs and

$$(Y_t = y_t \mid Y_{1:t-1}, X_t = k) \sim \begin{cases} \epsilon^{\lambda^t_k}(y_t) = f^k_{Y_{1:t-1}}(y_t) & k \in \mathcal{K}_s \\ \epsilon^{\lambda_k}(y_t) & k \in \mathcal{K}_p \end{cases}$$

for partition $\mathcal{K}_s \sqcup \mathcal{K}_p = \mathcal{K} = \{1, \ldots, K\}$.

That is, a subset of states have emissions distributions that break the conditional independence assumption on the observations Y. However, these emissions distributions are not learned, but provided as fixed functions. This results in time-dependent (but unlearned) emission distribution parameters λ^t_k governing the distribution of $(Y_t \mid X_t = k)$ for a subset of states. Emissions distributions for the remaining states are standard, and serve to learn customized perturbations around the deterministic predictive probability functions. Note that a streaming HMM is a special case of the PROPS model, with $K_s = 0$.

III. Notation

Notation is summarized in Table I. Learned PROPS model parameters are given by $\theta = \left(\pi, \tau, \{\lambda_k\}_{k\in\mathcal{K}_p}\right)$. For simplicity of presentation (and due to the streaming context), we assume the initial state distribution π to be fixed. The non-learned model objects are given by $(K, K_s, \{f^k_\cdot(\cdot)\}_{k\in\mathcal{K}_s})$.

TABLE I: Summary of notation

$\{\mathcal{M}_k\}_{k\in\mathcal{K}_s}$	source models
$\{f^k_{y_{1:t}}(\cdot)\}_{k\in\mathcal{K}_s}$	Predictive probability functions from source models
y	observed sequence
y_t	observation at timestep t
x	hidden state sequence
x_t	hidden state at timestep t
T	number of timesteps
W	number of "words" for discrete emissions
K_s, K_p, K	# source states, # perturbation states, and total # hidden states
$\mathcal{K}_s, \mathcal{K}_p, \mathcal{K}$	source states, perturbation states, hidden states
θ	learned PROPS model parameters
π	initial state distribution
τ	state transition matrix
$\{\epsilon_k\}_{k\in\mathcal{K}}$	emissions distributions
$s(X_{t-1}, X_t, Y_t)$	complete-data sufficient statistics

IV. Streaming training for Hidden Markov Models

Here we present material on streaming (also called online, recursive, or adaptive) learning of fixed model parameters in HMMs. The purpose is two-fold: (1) Algorithm 1 requires it for implementation, and (2) the overview is helpful to motivate the argument that PROPS is consistent.

A. Classical (batch) Estimation

Estimating (or fitting) the model by maximum likelihood means finding the parameters $\arg\max_\theta P(x, y|\theta)$. Yet because the HMM has latent variables, the sufficient statistics for the complete-data log likelihood are not observed. The **Expectation Maximization** (EM) algorithm handles this by iteratively computing expected sufficient statistics for the latent variables (given the current parameter estimates) before maximizing the parameters. That is, we iteratively compute

$$\arg\max_\theta \mathbb{E}_{p(x \mid y,\theta^i)} \log p(x, y \mid \theta) \qquad \text{for } i = 1, 2 \dots \quad (2)$$

until convergence. In particular, on the ith iteration, we can do

1) *E-step*: Compute the **expected complete-data sufficient statistics** (ESS), which for a HMM is
$$S^{i+1} = \frac{1}{T}\, \mathbb{E}_{\nu,\theta^i}\big[\sum_{t=0}^{T} s(X_{t-1}, X_t, Y_t) \mid Y_{0:T}\big] \quad (3)$$
where $s(\cdot)$ are the sufficient statistics from the complete-data likelihood.[1]
2) *M-step*: Update the parameter estimate to $\theta_{i+1} = \overline{\theta}(S_{i+1})$, where $\overline{\theta}$ refers to the operation of finding the maximum likelihood estimator.

B. Streaming Estimation

Classical (batch) estimation requires multiple passes through the entire data set – one pass for each iteration of the EM algorithm. In the streaming (or online) setting, the goal is to see an observation once, update model parameters, and discard the observation forever. We follow the procedure of [3], which essentially does streaming E-steps based on the data seen so far, and then a partial maximization for the M-step (i.e. model parameters are optimized prematurely, before strictly completing the E-step pass). Compared to batch learning, this technique takes less training time to get to the same level of estimation accuracy.

1) Overview: Here we compute *streaming estimates* of the expected complete-data sufficient statistics

$$S^t = \frac{1}{t}\mathbb{E}_{\nu,\theta^t}\big[\sum_{r=0}^{t} s(X_{r-1}, X_r, Y_r) \mid Y_{0:t}\big] \quad (4)$$

Streaming updates can be made to this quantity by decomposing it into two simpler functions. In addition to the typical **filter** function,

$$\phi_{t,\nu,\theta}(k) = P_{\nu,\theta}(X_t = k \mid Y_{0:t}) \quad (5)$$

[1]For example, in the case of discrete emissions, $s(X_{t-1}, X_t, Y_t) = \mathbb{1}_{(X_{t-1}=i, X_t=j, Y_t=w)}$.

we also define an **auxilliary function** as

$$\rho_{t,\nu,\theta}(i, k; \theta) = \frac{1}{t}\mathbb{E}_{\nu,\theta}\big[\sum_{r=0}^{t} s(X_{r-1}, X_r, Y_r) \mid Y_{0:t}, X_t = k\big] \quad (6)$$

Applying the *law of iterated expectation* with the filter (5) and auxiliary function (6), we can compute the *currently estimated expected complete-data sufficient statistics* as

$$S_t = \sum_x \widehat{\phi}_{t,\nu,\theta}(x)\widehat{\rho}_{t,\nu,\theta}(x) \quad (7)$$

2) Filter recursion: The filter recursion can be derived using elementary probability laws:

$$\begin{aligned}
&\phi_{t+1}(x) \propto P(X_{t+1} = x, Y_{0:t+1}) \\
&= P(Y_{t+1} \mid X_{t+1} = x) P(X_{t+1} = x, Y_{0:t}) \\
&= \sum_{x^r} P(Y_{t+1} \mid X_{t+1} = x) P(X_{t+1} = x, X_t = x^r, Y_{0:t}) \\
&= \sum_{x^r} P(Y_{t+1} \mid X_{t+1} = x) P(X_{t+1} = x, X_t = x^r) P(X_t = x^r, Y_{0:t}) \\
&\propto \sum_{x^r} \epsilon_\theta(x_{t+1}, Y_{t+1}), \tau_\theta(x^r, x) \phi_t(x^r)
\end{aligned} \quad (8)$$

3) Auxiliary function recursion: Given the filter $\phi_t(x)$, the **backwards retrospective probability** can be computed as

$$\begin{aligned}
r_t(k^r \mid k) &:= P(X_t = k^r \mid X_{t+1} = k, Y_{0:n}) \\
&\propto \phi_{t,\nu,\theta}(k^r)\tau_\theta(k^r, k)
\end{aligned} \quad (9)$$

The auxiliary variable recursion can be given in terms of that quantity:

$$\widehat{\rho}_{t+1}(k) \propto \sum_{k'} \Big(\gamma_{t+1} s(k', k, Y_{t+1}) + (1-\gamma_{t+1})\widehat{\rho}_t(k')\Big) r_t(k'|k) \quad (10)$$

where $(\gamma_t)_{t\geq 1}$ is a decreasing sequence of step-sizes satisfying $\sum_{t\geq 1}\gamma_t = \infty$ and $\sum_{t\geq 1}\gamma_t^2 < \infty$.

4) Decomposition: Assuming the standard scenario where the state variables (x_t) are discrete and θ is separable (into the state transition matrix τ and emissions distributions ϵ_k), we may decompose the auxiliary function. The **transition auxilliary function** is

$$\widehat{\rho}^{\tau}_{t,\nu,\theta}(i, j, k; \theta) = \frac{1}{t}\mathbb{E}_{\nu,\theta}\big[\sum_{r=0}^{t} \mathbb{1}\{X_{r-1} = i, X_r = j\} \mid Y_{0:t}, X_t = k\big] \quad (11)$$

and the recursion (10) simplifies to

$$\widehat{\rho}^{\tau}_{t+1}(i, j, k) = \gamma_{t+1}\mathbb{1}_{j=k} r(i \mid j) + (1 - \gamma_{t+1}) \sum_{k'=1}^{m} \widehat{\rho}^{\tau}_t(i, j, k') r_{t+1}(k' \mid k) \quad (12)$$

This quantity is a $K \times K \times K$ matrix, and the (i, j, k)th entry provides the expected number of (latent=i, latent=j) bigrams for a single "observation" of a bigram given that the final latent state is k.

Under the same scenario, the **emissions auxilliary function** is

$$\widehat{\rho}^{\epsilon}_{t,\nu,\theta}(i, k; \theta) = \frac{1}{t}\mathbb{E}_{\nu,\theta}\big[\sum_{r=0}^{t} \mathbb{1}\{X_r = i, s(Y_r)\} \mid Y_{0:t}, X_t = k\big] \quad (13)$$

and the recursion (10) simplifies to

$$\widehat{\rho}^{\epsilon}_{t+1}(i,k) = \gamma_{t+1}\mathbb{1}_{i=k}s(y_{t+1}) + (1-\gamma_{t+1})\sum_{k'=1}^{m}\widehat{\rho}^{\epsilon}_{t}(i,k')r_{t+1}(k' \mid k) \tag{14}$$

where $s(\cdot)$ are the sufficient statistics for the emissions distribution. For example, $s(Y_t) = \mathbb{1}_{Y_t=i}$ for categorical emissions and $s(Y_t) = (Y_t, Y_tY_t')$ for normal emissions. In the context of categorical emissions, we may render the emissions auxiliary variable as a $K \times W \times K$ matrix, where the (i,w,k)th entry provides the expected number of (latent=i, observed=w) bigrams for a single "observation" of a bigram given that the final latent state is k.

5) E-step: Using the decomposition of the auxiliary function given by (11) and (13), we may decompose (4) to obtain current estimates for the expected complete-data **transition statistics** as:

$$\widehat{S}^{\tau}_{t+1}(i,j) = \sum_{k=1}^{K}\widehat{\rho}^{\tau}_{t+1}(i,j,k)\widehat{\phi}_{t+1}(k) \tag{15}$$

and current estimates for the expected complete-data **emissions statistics** as

$$\widehat{S}^{\epsilon}_{t+1}(i) = \sum_{k=1}^{K}\widehat{\rho}^{\epsilon}_{t+1}(i,k)\widehat{\phi}_{t+1}(k) \tag{16}$$

6) M-step: Parameter optimization is immediate given the computation of $(\widehat{S}^{\tau}, \widehat{S}^{\epsilon})$ For instance, the update to the state transition matrix is given by $\widehat{\tau}_{ij} = \widehat{S}^{\tau}_{t}(i,j)/\sum_j \widehat{S}^{\tau}_{t}(i,j)$. The update to the kth emissions distribution in the case of an HMM on categorical observations is $\widehat{\lambda_k} \propto S^{\epsilon}_{t}(k)$.

V. Streaming training for PROPS

Algorithm 1 provides a psuedo-code implementation of PROPS .[2] The algorithm is presented in a high-level way, merely assuming access to an EM-based streaming HMM algorithm.[3] This allows us to present the psuedo-code in a way that offloads description of things like recursion variables and step-size parameters to the streaming HMM algorithm itself.[4]

It may seem surprising that Algorithm 1 could learn the probabilistic perturbations around the fixed predictive probability distributions $\{f^k\}_{k\in\mathcal{K}_s}$ from the source models $\{\mathcal{M}_k\}_{k\in\mathcal{K}_s}$, given that the source models provide PROPS with a stochastic process of emissions parameters, $\{\lambda_k^t\}_{k\in\mathcal{K}_s, t\geq 1}$ (and therefore a stochastic process of emissions distributions, $\{\epsilon_k^t\}_{k\in\mathcal{K}_s, t\geq 1}$), rather than the constant emissions distributions of a standard HMM. On the surface, this might seem to complicate the learning of the remaining (perturbation) emission distributions, $\{\epsilon_k\}_{k\in\mathcal{K}_p}$, as well as the transition dynamics across this heterogenous system.

However, as it turns out, the streaming HMM algorithm readily wraps around the source predictive probability functions, so long as they are kept updated at each iteration of the streaming algorithm. To see this, simply consider the ingredients of the streaming recursions from the online HMM updates:

- The filter recursion (8) depends upon the recursion variable itself, the state transition matrix τ and emissions distributions $\{\epsilon_k\}_{k\in\mathcal{K}}$.
- The transition auxiliary variable recursion (12) depends upon the recursion variable itself, the step-size hyperparameter, and the transition parameter τ.
- The emissions auxiliary variable recursion (14) depends upon the recursion variable itself, the step-size hyperparameter, the transition parameter τ, and the sufficient statistics for the emissions distribution computed from an observed individual datum.

Access to these quantities are available at any timestep t. In particular:

- $\{\epsilon_k\}_{k\in\mathcal{K}}$ are available from the M-step for $k \in \mathcal{K}_p$ and from provided scoring functions $\{f^k\}_{k\in\mathcal{K}_s}$ for $k \in \mathcal{K}_s$.
- All entries of $\tau = (k,k')_{k,k'\in\mathcal{K}}$ are available due to the M-step.

Thus, so long as the black-box predictive probability distributions $\{f^k\}_{k\in\mathcal{K}_s}$ are pre-trained rather than learned, we can apply Algorithm 1 to learn the parameters θ of PROPS .

Algorithm 1 Perturbed sequence modeling using PROPS

Require: Black-box predictive probability distributions $\{f^k\}_{k\in\mathcal{K}_s}$
Require: Initialized PROPS model parameters, θ
Require: # hidden states: $K : K >= |\mathcal{K}_s|$
Require: Streaming HMM algorithm based on EM
Require: t_{min}, warm-up period before M-step updates

for $t \in \{1,2,3,...\}$ **do**
  Update black-box predictive probability distributions, $\epsilon(y_t \mid k) = f^k_{y_{1:t}}(\cdot)$ for $k \in \mathcal{K}_s$
  Do E-step for $k \in [1:K]$
  if $t \geq t_{\text{min}}$ **then**
    Do M-step on $\theta = \left(\pi, \tau, \{\lambda_k\}_{k\in\mathcal{K}_p}\right)$
  end if
end for
return PROPS model parameters, θ

VI. Experiment 1 (Simulation): Are PROPS estimators consistent?

Roughly speaking, an estimator is **consistent** if the estimator applied to the whole population equals the parameter [5]. In other words, as more samples are taken, the estimator will converge to the true value of the parameter. It can be shown that the maximum likelihood estimator for a HMM's θ parameters

[2] A pip installable python package is also available, and will be provided in final form at publication time or can be provided now in draft form upon request.

[3] In particular, one need not specifically implement the EM-based streaming HMM algorithm of [3]; one could choose others [4].

[4] Compared to other classes of streaming algorithms for estimating HMM parameters, the EM-based streaming algorithms provide more accurate state estimates and are easier to implement [4]. However, other techniques (e.g., gradient-based and minimum prediction error) may be more appropriate when K is large. We leave the application of PROPS to those classes of streaming HMM algorithms the reader.

is strongly consistent (i.e. converges *a.s.*) under a minimal set up assumptions [6].[5] The estimator obtained from the streaming training algorithm of [3] is not as well-understood analytically, but its consistency is well-supported by a heuristic analytical argument and with empirical simulations [3]. But is it reasonable to expect the same behavior with PROPS ?

We conjecture that the answer is yes, because the proof of consistency for the maximum likelihood estimator of HMM parameters (and the heuristic proof for the corresponding streaming estimators of [3]) would carry over to PROPS , given the arguments of Section V. In other words, both the algorithmic recursions and the proof should be "blind" to the fact that some emissions are pinned to an external stochastic process.

Here we empirically evaluate whether the estimators of Algorithm 1 are consistent for the PROPS parameters θ.

A. Methodology

To evaluate consistency (which depends on convergence) we need to have a notion of distance between models. Luckily, [7] provides a notion of divergence between HMM model:

$$D(\theta_0, \theta) = \lim_{t\to\infty} D_t = \lim_{t\to\infty} \frac{1}{t}\Big[\log P(Y_t \mid \theta_0) - \log P(Y_t \mid \theta)\Big]$$

In practice, one recursively generates samples $Y_{1:t} \sim \text{HMM}(\theta_0)$ for increasing $t \geq 1$ and computes D_t until $\{D_s\}_{s=1}^t$ meets a convergence threshold. We then approximate $D(\theta_0, \theta) \approx D_t$. We can co-opt D as a distance[6] between PROPS models by applying it to the PROPS parameters instead.

To check if Algorithm 1 provides consistent estimates of the PROPS model, we sample $Y_{1:t} \sim \text{PROPS}\ (\theta_0)$ for some t, and then we apply Algorithm 1 to form estimator $\widehat{\theta}_t$. If the PROPS estimator is consistent, then we should have that $D(\theta_0, \widehat{\theta}_t) \to 0$ as t increases.

We evaluate consistency for a PROPS model with $K = 3, K_s = 1, f_t \overset{i.i.d}{\sim} \text{Dirichlet}(\mathbb{1}_W)$, categorical emissions, and various W.[7] Note that computing $D(\theta_0, \widehat{\theta}_t)$ between a true and an estimated model requires a *separate* hold-out sample, $Y'_{1:t} \sim \text{PROPS}\ (\theta_0)$. Each time a sample is drawn from PROPS (θ_0), we perturb f_t slightly by drawing $f'_t \overset{i.i.d}{\sim} \text{Dirichlet}(\alpha\, f_t)$ for concentration parameter α=100.

B. Results and Discussion

Figure 2 shows that, as desired, $D(\theta_0, \widehat{\theta}_t)$ decreases as sample size t grows. As expected, the rate of convergence is slower for models that are more complex.

[5]Although it seems that, for batch EM training, statistical consistency can be achieved only by using more batch EM iterations as the sample size grows[3].

[6]Technically, this isn't a mathematical distance function, or metric, due to the function's non-symmetry. However, this function is easily symmetrizable but taking the mean of $D(\theta_0, \theta)$ and $D(\theta, \theta_0)$.

[7]In particular, the choice of f_t reflects the fact that a black-box feedforward model is sufficiently complex that its sequence of predictive probability distributions may look like independent draws from a Dirichlet.

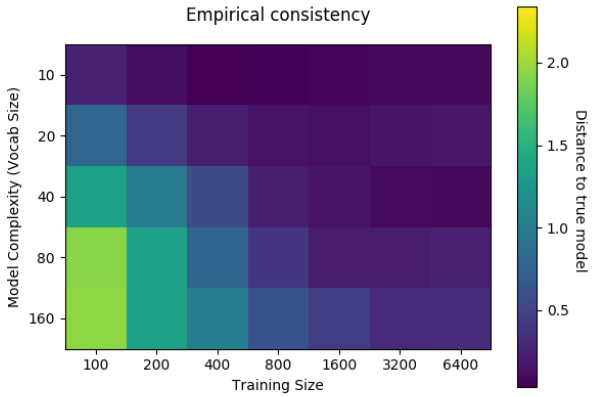

Fig. 2: *Empirical investigation on the consistency of the* PROPS *estimator provided by Algorithm 1.*

VII. Experiment 2: Probabilistically Perturbing a Wikipedia language model into a model of President Donald J. Trump's tweeting

Now we investigate the utility of the PROPS model in the context of personalized language modeling.

A. Methodology

The source model was created on a g2.8xlarge EC2 instance (which is backed by four NVIDIA GPUs) using these steps:

1) The entire Wikipedia corpus (2,581,793 articles, approximately 15GB on disk) was obtained using the `gensim` python module. There were approximately 2,568 words per article. We removed punctuation and capitalization from the articles.
2) A word2vec model was built on this corpus using the python module `henson` with a random subcorpus of 5,000 Wikipedia articles. In particular, a continuous bag of words (CBOW) model with a context window of 5 was used. The embedding was chosen to have dimensionality 100. The word had to appear at least 12 times in the subcorpus in order to survive. This led to a vocabulary of $W = 13,711$ words. All other words were rendered as 'OOV' (out of vocabulary).
3) A stacked Long Short Term Memory (LSTM) network was then trained on the full Wikipedia corpus. The structure of the LSTM was one embedding layer (with embedding weights from the pretrained word2vec model) and two LSTM layers stacked together. The recurrent activation (for the input/forget/output gates) used was hard-sigmoid, whereas the activation used for the cell state and the hidden state was 'tanh'. Each LSTM cell has a output, and then the output from the whole LSTM layer is a list. Each member of this list is fed into a dense layer, where the activation is the 'tanh.' That output was fed into another dense layer, where the activation is softmax. The RMSprop optimizer was used for training.

The loss was the categorical cross-entropy loss between the input sequence and the output sequence, where the output sequence is a shift left by one word of the input sequence. We treat 50 words as a sequence, and use a minibatch size of 5000 sequences. We trained for 2 epochs (i.e. went through the corpus two times). The LSTM training took > 1 day.

Our target corpus for transfer learning was a corpus of tweets posted by President Donald J. Trump from 07/15/2014 until 12/21/2016.[8] We removed urls, phrases with leading hashtags, and punctuation.

We used PROPS to perturb the baseline, black-box Wikipedia model into a probabilistic, personalized language model of Trump's tweeting. The PROPS model was fit with $K = 3, |\mathcal{K}_s| = 1, W = 13,711$. The model was very lightly trained – on the first 2,000 words in the tweet corpus, fed to the model in chronological order.

For comparison, we also fit a standard streaming HMM model to the tweets alone. The HMM model was fit with $K = 3, W = 13,711$.

B. Results and Discussion

In Figure 3, we show the log likelihood ratio of **predictive probability scores** (i.e. $P(y_t \mid y_{1:t-1}) = \sum_k P(y_t \mid x_t = k)P(x_t = k \mid y_{1:t-1})$) for the PROPS model versus a standard HMM model on a representative subtweet. We first note that the mean is positive, meaning that the PROPS model outperforms a standard HMM model in predicting the next word. Indeed, the mean predictive probability is 0.01 for the PROPS model, compared to 0.004 for the standard HMM model (and $7.2 \cdot 10^{-5}$ for random guessing). Further insight can be obtained by investigating the blue stretches of text – i.e. the particular locations where the PROPS model outperforms the standard HMM. The black-box predictive model is able to contribute knowledge about collocations that inhere in the English language generally ("poll states that", "a record number", "have lost all"). A standard HMM, trained from scratch on Donald J. Trump's tweets, would have virtually zero knowledge of such general collocations, particularly after a short training time.

In Figure 4, we show the log likelihood ratio for whether the latent state has membership in a local perturbation mode. In other words, if $p_t := P(x_t \in \mathcal{K}_p \mid y_{1:t})$, then $1 - p_t := P(x_t \in \mathcal{K}_s \mid y_{1:t})$, and we plot the log of $\frac{p_t}{1-p_t}$. What we see here is that the PROPS model has successfully customized (or personalized) the more generic language model to handle the particular content of President Trump's tweeting. The final word in the phrases "a record number of...Americans", "have lost all...faith", and "in president...Obama" are all substantially more likely to be observed in language from President Trump than in a broader corpus of English language (i.e. Wikipedia). The local perturbation modes, $\mathcal{K}_p$, of the PROPS model provide this capacity for personalization.

[8]Corpus obtained from https://www.kaggle.com/austinvernsonger/donaldtrumptweets/#data.csv

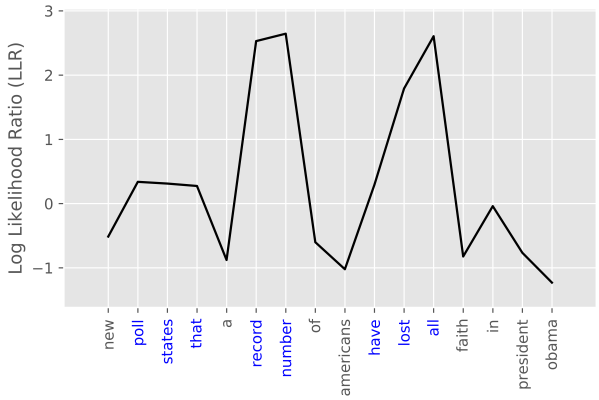

Fig. 3: *Log likelihood ratio of predictive probability scores for the* PROPS *model (trained on Trump tweets) versus a standard HMM model (trained on Trump tweets).* Words with large positive values are colored in blue.

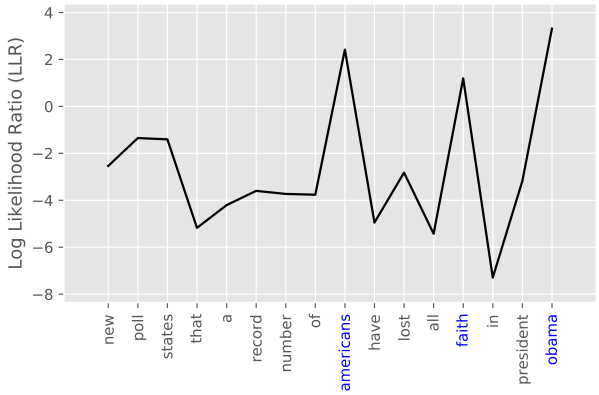

Fig. 4: *Log likelihood ratio for the* PROPS *model that the latent state underlying the observed language belongs to a local perturbation (Trump) mode rather than a baseline (Wikipedia) mode.* Words with large positive values are colored in blue.

Of course, the cost of personalization is a poorer fit to the language of others. In Figure 5, we show the log likelihood ratio of predictive probability scores for the PROPS model versus the baseline RNN model on a tweet from Yann LeCun on Oct. 8, 2018.[9] Compared to the more generic RNN model, the PROPS model, which was personalized to Trump, was much more surprised by the ending of phrases like "to...theoretical", "and....experimental physics", and "applying deep...learning."

We emphasize the lightweight nature of the transfer.

[9]"So many papers applying deep learning to theoretical and experimental physics! Fascinating."

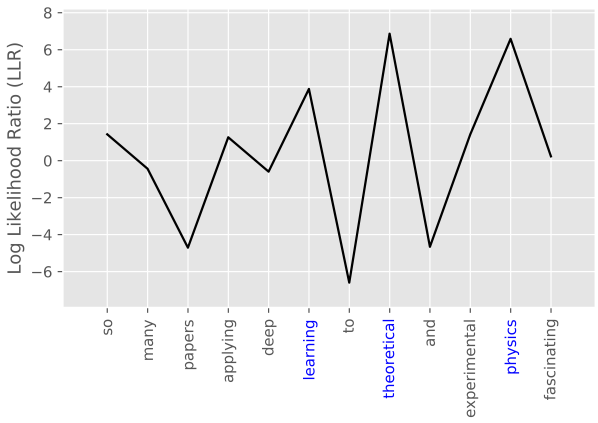

Fig. 5: *Log likelihood ratio of predictive probability scores for the generic English language model versus a corresponding* PROPS *model that has been customized to President Donald J. Trump's tweets.* The test data here is a tweet from machine learning researcher Yann Lecun. Words with large positive values are colored in blue.

Whereas the source RNN model was trained on $\sim$ 6.6 billion words from the Wikipedia corpus, the PROPS customization was performed on mere 2,000 additional words from Trump tweets.

VIII. Related Work

The (RNN-based) language model personalization scheme of [8] provides many of the desiderata outlined in the introduction. By forming a source RNN on a large dataset and then retraining only the last layer on the endpoint, they obtain a lightweight, fast transfer learning scheme for sequential data that respects user privacy. The primary drawback of [8] relative to PROPS is the loss of interpretability in the personalized model. Because the hidden variables of that model are not stochastic, one loses any insight into the respective contributions of the original source model vs. the personalization modes towards predictions about upcoming behavior. A subsidiary drawback is the loss of reliability as the more expressive model will also have higher variance.

A *stochastic recurrent neural network* (SRNN) [9] addresses a shortcoming of RNN's relative to state-space models such as HMM's by allowing for stochastic hidden variables. We surmise that the the SRNN framework will eventually generate a state-of-the-art transfer learning mechanism for sequential data that satisfies the interpretability desideratum from the introduction. However, to our knowledge, such a mechanism has not yet been developed. Moreover, the training of a SRNN is substantially more complex than training a standard RNN, let alone a HMM, and one would expect that computational complexity to spill over into the transference algorithm. If so, PROPS would provide a lightweight alternative.

IX. Conclusion

The PROPS model is a lightweight transfer learning mechanism for sequence learning which learns probabilistic perturbations around the predictions of one or more arbitrarily complex, pre-trained black box models. In the results section, we saw that the PROPS model has an advantage over a standard HMM, because it has a "head start" – for example, it knows about collocations in the English language before ever seeing a single Trump tweet. On the other hand, the PROPS model has an advantage over a generic black-box RNN model, because it is customized to the target audience. While there are other schemes for transfer learning with RNN's, PROPS is a fully probabilistic model. In particular, one can perform inference (filtering, smoothing, and MAP estimates) on the hidden states. With this inference, one can make probability statements about when an agent is behaving like the general corpus, when they are behaving in ways that are more characteristic of themselves, and when they are acting unusually more generally.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] M. Panzner and P. Cimiano, "Comparing hidden markov models and long short term memory neural networks for learning action representations," in *International Workshop on Machine Learning, Optimization and Big Data*, pp. 94–105, Springer, 2016.

[3] O. Cappé, "Online em algorithm for hidden markov models," *Journal of Computational and Graphical Statistics*, vol. 20, no. 3, pp. 728–749, 2011.

[4] W. Khreich, E. Granger, A. Miri, and R. Sabourin, "A survey of techniques for incremental learning of hmm parameters," *Information Sciences*, vol. 197, pp. 105–130, 2012.

[5] E. S. Martín and F. Quintana, "Consistency and identifiability revisited," *Brazilian Journal of Probability and Statistics*, pp. 99–106, 2002.

[6] R. Douc, E. Moulines, J. Olsson, R. Van Handel, *et al.*, "Consistency of the maximum likelihood estimator for general hidden markov models," *the Annals of Statistics*, vol. 39, no. 1, pp. 474–513, 2011.

[7] B.-H. Juang and L. R. Rabiner, "A probabilistic distance measure for hidden markov models," *AT&T technical journal*, vol. 64, no. 2, pp. 391–408, 1985.

[8] S. Yoon, H. Yun, Y. Kim, G.-t. Park, and K. Jung, "Efficient transfer learning schemes for personalized language modeling using recurrent neural network," *arXiv preprint arXiv:1701.03578*, 2017.

[9] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther, "Sequential neural models with stochastic layers," in *Advances in neural information processing systems*, pp. 2199–2207, 2016.