A. Proofs

A.1. Proof of Theorem 1

Note that while the ($\Leftarrow$) direction would follow from Theorem 4 below, we provide an independent proof that does not rely on the Exponential Time Hypothesis.

Proof of Theorem 1. ($\Rightarrow$) Suppose by way of contradiction that Algorithm 1 has failed. Since $L = L' = \bigcup\{v \in V_f \mid v \in P(v)\}$ but $\forall v \in V_f,\ P(v) \subsetneq L$, there must be at least two distinct $u, w \in V_f$ and two distinct plates $a \neq b \in L$ such that $a \in P(u) \setminus P(w)$ and $b \in P(w) \setminus P(u)$. Letting $v = f$ with $a, b \in P(v)$ provides the required graph minor $(\{u, v, w\},\ \{(u, v), (v, w)\},\ P)$.

($\Leftarrow$) Suppose by way of contradiction that G has such a graph minor $(\{u, v, w\},\ \{(u, v), (v, w)\},\ P)$. Algorithm 1 must eventually either fail or reach a factor f_u in leaf plate set $L_u \ni a$ where the plate a is PRODUCT-reduced. Similarly at another time, the algorithm must reach a factor f_w in plate set $L_w \ni b$ where the plate b is PRODUCT-reduced.

Consider the provenance of factors f_u and f_w. Since v contains a common factor f_v in both plates a, b, factors f_u and f_w must share f_v as a common ancestor. Note that in Algorithm 1's search for a next plate set $L' \subsetneq L$, the plate set is strictly decreasing. Thus if f_u and f_v share a common ancestor, then either $L_u \subseteq L_w$ or $L_u \supseteq L_w$. But this contradicts $a \in L_u \setminus L_w$ and $b \in L_w \setminus L_u$. □

A.2. Proof of Theorem 2

We first define plated junction trees, then prove a crucial lemma, and finally prove a stronger form of Theorem 2.

Definition 4. A *junction tree* (or *tree decomposition*) of a factor graph (V, F, E) is a tree (V_J, E_J) whose vertices are subsets of variables V such that: (*i*) each variable is in at least one junction node (i.e. $\bigcup V_J = V$); (*ii*) for each factor $f \in F$, there is a junction vertex $v_J \supseteq \{v \in V \mid (v, f) \in E\}$ containing all of that factor's variables; and (*iii*) each variable $v \in V$ appears in nodes $\{v_J \in V_J \mid v \in v_J\}$ forming a contiguous subtree of E_J.

Definition 5. A *plated junction tree* of a plated factor graph $(V, F, E, P : V \cup F \to \mathcal{P}(B))$ is a plated graph $(V_J,\ E_J,\ P_J : V_J \to \mathcal{P}(B))$ where (V_J, E_J) is a junction tree, $P_J(v_J) = \bigcup_{v \in v_J} P(v)$, and (*iv*) each variable $v \in V$ appears in some junction vertex $v_J \ni v$ with exactly the same plates $P(v_J) = P(v)$.

We extend unrolling from plated factor graphs to plated junction trees in the obvious way. Note that a plated junction tree may unroll to a junction graph that is not a tree, since unrolling may introduce cycles.

Definition 6. The *width* of a junction tree $T = (V_J, E_J)$ is $\mathrm{width}(T) = \max_{v_J \in V_J} |v_J| - 1$. The *treewidth* of a factor graph G is the width of its narrowest junction tree, $\mathrm{treewidth}(G) = \min_{T \text{ of } G} \mathrm{width}(T)$.

Lemma A.1. *Let $G = (V, F, E, P : V \cup F \to \mathcal{P}(B))$ be a plated factor graph and $W \in \mathbb{N}$. If for every plate size assignment $M : B \to \mathbb{N}$ there is a junction tree T_M of $\mathrm{unroll}(G, M, B)$ with $\mathrm{width}(T_M) \leq W$, then there is a single plated junction tree T of G such that for every size assignment $M : B \to \mathbb{N}$, $\mathrm{width}(\mathrm{unroll}(T, M, B)) \leq W$.*

Proof. By induction on $|B|$ it suffices to show that, splitting off a single plate $B = \{b\} \cup B'$, if there is a family of plated junction trees $\{T_m \mid m \in \mathbb{N}\}$ satisfying

$$\forall m \in \mathbb{N},\ \exists T_m \text{ of } \mathrm{unroll}(G, M, b),\ \forall M' : B' \to \mathbb{N},\ \mathrm{width}(\mathrm{unroll}(T_m, M', B')) \leq W \tag{1}$$

then for a single plated junction tree T,

$$\exists T \text{ of } G,\ \forall M : \{b\} \cup B' \to \mathbb{N},\ \mathrm{width}(\mathrm{unroll}(T, M, \{b\} \cup B')) \leq W \tag{2}$$

Thus let us assume the hypothesis (1) and prove (2).

Our strategy is to progressively strengthen the hypothesis (1) by forcing symmetry properties of each T_m using Ramsey-theory arguments. Eventually we will show that for any m, there is a plated junction tree T_m satisfying hypothesis (1) that is the unrolled plated junction tree $\mathrm{unroll}(T, M, b)$ for some plated junction tree T. Choosing $m \geq C$, it follows that T satisfies (2).

First we force T_m to have nodes that are symmetric along plate b. Split variables V_1 into plated $V_p = \{v \in V_1 \mid b \in P(v)\}$ and unplated $V_u = \{v \in V_1 \mid b \notin P(v)\}$ sets. For each $S_u \subseteq V_u$ and $S_p \subseteq V_p$ color all plate indices $i \in \{1, \ldots, m\}$ by a coloring function

$$C(i) = \begin{cases} 1 & \exists v_J \in V_J,\ v_J \cap (V_u \cup V_p[i]) = S_u \cup S_p[i] \\ 0 & \text{otherwise} \end{cases}$$

where we use the shorthand $X[i] = \{x[i] \mid x \in X\}$. By choosing $m' \geq 2m - 1$, we can find $T_{m'}$ with a subset of m plate indices all of a single color. Rename these plate indices to construct a new T_m whose vertices are symmetric in this sense.

Next for each $u, v \in V_p$, color all pairs of plate indices $i < j \in \{1, \ldots, m\}$ by a coloring function

$$C(i, j) = \begin{cases} 1 & \exists v_J \in V_J,\ \{u[i], v[j]\} \subseteq v_J \\ 0 & \text{otherwise} \end{cases}$$

By Ramsey's theorem (Ramsey, 1930) we can choose a sufficiently large $m' \geq R(m, m)$ such that $T_{m'}$ has a subset of m plates such that all pairs have a single color. Indeed

by choosing $m \geq W$, we can force the color to be 0, since the contiguity property (iii) of Definition 4 would require a single junction tree node to contain at least W vertices, violating our hypothesis.

At this point we have forced V_J to be symmetric in plate indices up to duplicates, but duplicates may have been introduced in selecting out m plate indices from a larger set m' (e.g. upon removing plate index 4, $\{x\}-\{x, y[4]\}$ becomes $\{x\}-\{x\}$). We now show that these duplicates can be removed by merging them into other nodes.

Let u, w be two duplicate nodes in the plated junction tree T_m, so that $P_J(u) = P_J(w)$. Let $u-v-\cdots-w$ be the path connecting u, w (with possibly $v = w$). By the contiguity property (iii) of Definition 4, v must have at least the variables of u, w, hence must have at least as deep plate nesting, i.e. $P_J(v) \supseteq P_J(u)$. Replace the edge $u-v$ with a new edge $u-w$. No cycles can have been created, since v is more deeply plated than w. No instances of the forbidden graph minor can have been created, since the new edge (u, v) lies in a single plate. Indeed hypothesis 1 is still preserved, and symmetry of V_J is preserved. Merge u into w. Again the hypothesis and symmetry are preserved. Iterating this merging process we can force T_m to have no duplicate nodes.

Now since V_J is symmetric in plate b and P_J is defined in terms of V_J, also P_J must be symmetric in plate b. We next force E_J to be symmetric along plate b.

For each $S_u, S'_u \subseteq V_u$ and $S_p, S'_p \subseteq V_p$, color all plate indices $i \in \{1 \ldots, m\}$ by a coloring function

$$C(i) = \begin{cases} 1 & (S_u \cup S_p[i],\ S'_u \cup S'_p[i]) \in E \\ 0 & \text{otherwise} \end{cases}$$

By choosing $m' \geq 2m - 1$, we can find a $T_{m'}$ with a subset of m plate indices with symmetric within-plate-index edges.

For each $S_u, S'_u \subseteq V_u$ and $S_p, S'_p \subseteq V_p$, color all pairs of plate indices $i < j \in \{1, \ldots, m\}$ by a coloring function

$$C(i, j) = \begin{cases} 1 & (S_u \cup S_p[i],\ S'_u \cup S'_p[j]) \in E \\ 0 & \text{otherwise} \end{cases}$$

By Ramsey's theorem we can choose a sufficiently large $m' \geq R(m, m)$ such that $T_{m'}$ has a subset of m plates such that all pairs have a single color. Indeed we can force the color to be 0, since for $m \geq 3$ any complete bipartite graph would create cycles, violating the tree assumption. Hence there are no between-plate-index edges.

At this point, we can construct a $T_m = (V_J, E_J, P_J)$ that is symmetric in plate indices and such that E_J never contains edges between nodes with b-plated variables with different b indices. Hence $T_{\max(W,3)}$ can be rolled into a plated junction tree T that always unrolls to a junction tree. Finally, since V_J never contains b-plated variables with more than one b index,

$$\text{width}(\text{unroll}(T, M, B)) \ = \ \text{width}(\text{unroll}(T, 1, B)) \ \leq \ W$$

□

We now prove a strengthening of Theorem 2.

Theorem 4. *Let $G = (V, F, E, P : V \cup F \to \mathcal{P}(B))$ be a plated factor graph with nontrivial variable domain $\forall v \in V, |\text{dom}(v)| \geq 2$, and $M : B \to \mathbb{N}$ be plate sizes. The following are equivalent:*

1. *Algorithm 1 succeeds on G.*
2. PLATEDSUMPRODUCT(G, M) *can be computed with complexity polynomial in M.*
3. *The treewidth of G's unrolled factor graph is (asymptotically) independent of plate sizes M.*
4. *There is a plated junction tree of G that unrolls to a junction tree for all plate sizes M.*
5. *There is a plated junction tree of G that has no plated graph minor $(\{u, v, w\}, \{(u, v), (v, w)\}, P)$ where $P(u) = \{a\}$, $P(v) = \{a, b\}$, $P(w) = \{b\}$, and $a \neq b$.*
6. *G has no plated graph minor $(\{u, v, w\}, \{(u, v), (v, w)\}, P)$ where $P(u) = \{a\}$, $P(v) = \{a, b\}$, $P(w) = \{b\}$, $a \neq b$, and u, w both include variables.*

Proof. (1 $\Rightarrow$ 2) Algorithm 1 has complexity polynomial in M, since both SUMPRODUCT and PRODUCT are polynomial in M and the **while** loop is bounded independently of M.

(2 $\Rightarrow$ 3) Apply (Kwisthout et al., 2010) to the unrolled factor graph.

(3 $\Rightarrow$ 4) Apply Lemma A.1.

(4 $\Rightarrow$ 5) Any plated junction tree with plated graph minor in (5) would unroll to a non-tree, since the minor would induce cycles when $M(a) \geq 2$ and $M(b) \geq 2$ (as in Example 3.2). Hence the plated junction tree of (4) must satisfy (5).

(5 $\Rightarrow$ 6) If G has such a plated graph minor, then any plated junction tree must have the corresponding plated graph minor.

(6 $\Rightarrow$ 1) Apply Theorem 1. □

Proof of Theorem 2. Apply Theorem 4 (1 $\Longleftrightarrow$ 2). □

B. Experimental details

B.1. Hidden Markov Models with Autoregressive Likelihoods

The joint probability (for a single sequence) for the HMM model is given by

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \tag{3}$$

where $\boldsymbol{x}_{1:T}$ are the discrete latent variables and $\boldsymbol{y}_{1:T}$ is the sequence of observations. For all twelve models the likelihood is given by a Bernoulli distribution factorized over the 88 distinct notes. The two Factorial HMMs have two discrete latent variables at each timestep: $\boldsymbol{x}_t$ and $\boldsymbol{w}_t$. They differ in the dependence structure of $\boldsymbol{x}_t$ and $\boldsymbol{w}_t$. In particular for the FHMM the joint probability is given by

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{w}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{w}_t|\boldsymbol{w}_{t-1}) \tag{4}$$

i.e. the dependency structure of the discrete latent variables factorizes at each timestep, while for the PFHMM (i.e. Partially Factorial HMM) the joint probability is given by

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{w}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{w}_t)p(\boldsymbol{w}_t|\boldsymbol{w}_{t-1}) \tag{5}$$

All models correspond to tractable plated factor graphs (in the sense used in the main text) and admit efficient maximum likelihood gradient-based training.

The autoregressive models have the same dependency structure as in Eqn. 3-5, with the difference that the likelihood term at each timestep has an additional dependence on $\boldsymbol{y}_{t-1}$. For the four arXXX models, this dependence is explicitly parameterized with a conditional probability table, with the likelihood for each note $p(y_{t,i}|\cdot)$ depending explicitly on $y_{t-1,i}$ (but not on $y_{t-1,j}$ for $j \neq i$). For the four nnXXX models this dependence is parameterized by a neural network so that $p(y_{t,i}|\cdot)$ depends on the entire vector of notes $\boldsymbol{y}_{t-1}$. In detail the computational path of the neural network is as follows. First, a 1-dimensional convolution with a kernel size of three and N_{channels} channels is applied to $\boldsymbol{y}_{t-1}$ to produce $\boldsymbol{c}_t$. We then apply separate affine transformations to the latent variables (either $\boldsymbol{x}_t$ or $\boldsymbol{x}_t$ and $\boldsymbol{w}_t$) and $\mathbf{c}_t$, add together the resulting hidden representations, and apply a ReLU non-linearity. A final affine transformation then maps the hidden units to the 88-dimensional logits space of the Bernoulli likelihood. We vary $N_{\text{channels}} \in \{2, 4\}$ and fix the number of hidden units to 50.

We evaluate our models on three of the polyphonic music datasets considered in Boulanger-Lewandowski et al. (2012), using the same train/test splits. Each dataset contains at least 7 hours of polyphonic music; after pre-processing each dataset consists of $\mathcal{O}(100 - 1000)$ sequences, with each sequence containing $\mathcal{O}(100 - 1000)$ timesteps. For each model we do a grid search over hyperparameters and train the model to approximate convergence and report test log likelihoods on the held-out test set (normalized per timestep). For all models except for the second-order HMMs we vary the hidden dimension $D_h \in \{9, 16, 25, 36\}$. For the Factorial HMMs D_h is interpreted as the size of the entire latent space at each timestep so that the dimension of each of the two discrete latent variables $\boldsymbol{x}_t$ and $\boldsymbol{w}_t$ at each timestep is given by $\sqrt{D_h}$. For the second-order HMMs we vary the hidden dimension $D_h \in \{9, 12, 16, 20\}$ so as to limit total memory usage (which scales as $\mathcal{O}(D_h^\ell)$ for an ℓ^{th}-order HMM).

We use the Adam optimizer with an initial learning rate of 0.03 and default momentum hyperparameters (Kingma & Ba, 2014); over the course of training we decay the learning rate geometrically to a final learning rate of 3×10^{-5}. For the JSB and Piano datasets we train for up to 300 epochs, while for the Nottingham dataset we train for up to 200 epochs. We follow (noisy) gradient estimates of the log likelihood by subsampling the data into mini-batches of sequences; we use mini-batch sizes of 20, 15, and 30 for the JSB, Piano, and Nottingham datasets, respectively. We clamp all probabilities to satisfy $p \geq p_{\min} = 10^{-12}$ to avoid numerical instabilities. In order to better avoid bad local minima, for each hyperparameter setting we train with 4 (2) different random number seeds for models without (with) neural networks in the likelihood, respectively. For each dataset we then report results for the model with the best training log likelihood.

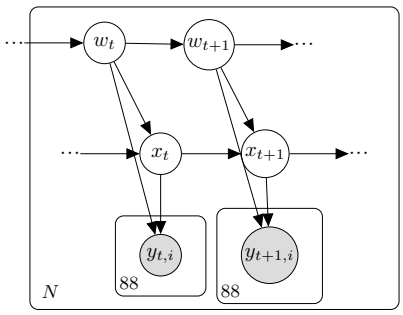

Figure 6. Plate diagram for the PFHMM in Sec. 6.1. The outermost plate encodes the independence among the N time series, while the plates at each time step encode the fact that the likelihood term $p(y_t|\cdot)$ decomposes into a product of 88 Bernoulli likelihood factors, one for each note $y_{t,i}$.

B.2. Hierarchical Mixed Effect Hidden Markov Models

B.2.1. Harbour seal tracking dataset details

We downloaded the data from `momentuHMM` ((McClintock & Michelot, 2018)), an R package for analyzing animal movement data with generalized hidden Markov models. The raw datapoints are in the form of irregularly sampled time series (datapoints separated by 5-15 minutes on average) of GPS coordinates and diving activity for each individual in the colony (10 males and 7 females) over the course of a single day recorded by lightweight tracking devices physically attached to each animal by researchers. We used the `momentuHMM` harbour seal example[20] preprocessing code (whose functionality is described in detail in section 3.7 of (McClintock & Michelot, 2018)) to independently convert the raw data for each individual into smoothed, temporally regular time series of step sizes, turn angles, and diving activity, saving the results and using them for our population-level analysis.

B.2.2. Model details

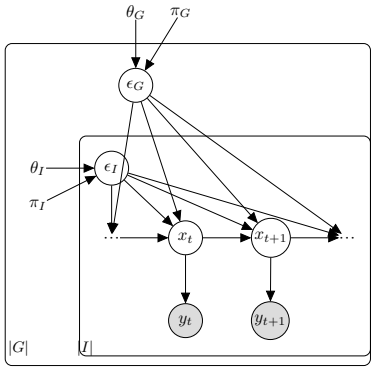

Figure 7. A single state transition in the hierarchical mixed-effect hidden Markov model used in our experiments in Section 6.2 and described below. θs and πs are learnable parameters. We omit fixed effects from the diagram as there were none in our experiments.

Our models are special cases of a time-inhomogeneous discrete state space model whose state transition distribution is specified by a hierarchical generalized linear mixed model (GLMM). At each timestep t, for each individual trajectory $b \in I$ in each group $a \in G$ (male and female in our experiments), we have

$$\text{logit}(p(x_{ab}^{(t)} = \text{state } i \mid x_{ab}^{(t-1)} = \text{state } j)) = \left(\epsilon_{G,a} + \epsilon_{I,ab} + Z_{I,ab}^{\intercal}\beta_1 + Z_{G,a}^{\intercal}\beta_2 + Z_{T,abt}^{\intercal}\beta_3\right)_{ij}$$

where a, b correspond to plate indices, ϵs are independent random variables, Zs are covariates, and βs are parameter vectors. See Fig. 7 for the corresponding plate diagram. The models in our experiments did not include fixed effects as there were no covariates Z in the harbour seal data, but they are frequently used in the literature with similar models (see e.g. (Towner et al., 2016)) so we include them in our general definition.

The values of the independent random variable ϵ_I and ϵ_G are each sampled from a set of three possible values shared across the individual and group plates, respectively. That is, for each individual trajectory $b \in I$ in each group $a \in G$, we sample single random effect values for an entire trajectory:

$$\begin{aligned}
\iota_{G,a} &\sim \text{Categorical}(\pi_G) \\
\epsilon_{G,a} &= \theta_G[\iota_{G,a}] \\
\iota_{I,ab} &\sim \text{Categorical}(\pi_{I,a}) \\
\epsilon_{I,ab} &= \theta_{I,a}[\iota_{I,ab}]
\end{aligned}$$

Note that each ϵ is a $D_h \times D_h$ matrix, where $D_h = 3$ is the number of hidden states per timestep in the HMM.

Observations $y^{(t)}$ are represented as sequences of real-valued step lengths and turn angles, modelled by zero-inflated Gamma and von Mises likelihoods respectively. The seal models also include a third observed variable indicating the amount of diving activity between successive locations, which we model with a zero-inflated Beta distribution following (McClintock & Michelot, 2018). We grouped animals by sex and implemented versions of this model with (i) no random effects (as a baseline), and with random effects present at the (ii) group, (iii) individual, or (iv) group+individual levels.

We chose the Gamma and von Mises likelihoods because they were the ones most frequently used in other papers describing the application of similar models to animal movement data, e.g. (Towner et al., 2016); another combination, used in (McClintock & Michelot, 2018), modelled step length with a Weibull distribution and turn angle with a wrapped Cauchy distribution.

Unlike our models, the models in (McClintock et al., 2013; McClintock & Michelot, 2018) incorporate substantial additional prior knowledge in the form of hard constraints on various parameters and random variables (e.g. a maximum step length corresponding to the maximum distance a harbour seal can swim in the interval of a single timestep).

B.2.3. Training details

We used the Adam optimizer with initial learning rate 0.05 and default momentum hyperparameters (Kingma & Ba, 2014), annealing the learning rate geometrically by a factor of 0.1 when the training loss stopped decreasing. We trained the models for 300 epochs with 5 restarts from random initializations, using batch gradient descent because the number of individuals (17) was relatively small. The number of random effect parameter values was taken from

[20] `https://git.io/fjc8i`

(McClintock & Michelot, 2018) and all other hyperparameters were set by choosing the best values on the model with no random effects.

B.3. Sentiment Analysis

B.3.1. Dataset details

Sentences in Sentihood were collected from Yahoo! Answers by filtering for answers about neighbourhoods of London. Specific neighbourhood mentions were replaced with generic `location1` or `location2` tokens. We follow the previous work (Saeidi et al., 2016; Ma et al., 2018; Liu et al., 2018) and restrict training and evaluation to the 4 most common aspects: price, safety, transit-location, and general.

To give a clearer picture of the task, consider the sentence "`Location1` is more expensive but has a better quality of life than `Location2`", the labels encode that with respect to `Location1` the sentence is *negative* in aspect *price*, but *positive* in *general*. Similarly `Location2` would have the opposite sentiments in those two aspects. The remaining aspects for both locations would have the *none* sentiment.

We preprocess the text with SpaCy (Honnibal & Montani, 2017) and lowercase all words. We append the reserved symbols '$\langle bos \rangle$' and '$\langle eos \rangle$' to the start and end of all sentences.

B.3.2. Model details

Our reimplemented `LSTM-Final` baseline uses a BLSTM on top of word embeddings. The hidden state of the BLSTM is initialized with an embedding of the location and aspect. A projection of the final hidden state is then used to classify the sentence-level sentiment by applying the softmax transformation. For the `CRF-LSTM-Diag` model, the potentials of the graphical model are given by:

$$\begin{aligned} G_t(z_t, l, a, \mathbf{x}) &= W[z_t]^T \text{LSTM}(\text{Emb}(\mathbf{x}), \text{Emb}(l, a))[t]) \\ F_t(y, z_t, l, a, \mathbf{x}) &= \text{diag}(\theta_{\text{none}}, \theta_{\text{pos}}, \theta_{\text{neg}})[y, z_t] \end{aligned} \tag{6}$$

where $W \in \mathbb{R}^{3 \times D}$ contains a D dimensional vector for each sentiment, D is the dimensionality of the LSTM, and the output of the LSTM is of dimension $T \times D$. The LSTM function takes as input a sequence of word embeddings as well as an initial hidden state given by embedding the location and aspect. The Emb function projects the words or location and aspect into a low-dimensional representation. The potentials of the `CRF-LSTM-LSTM` model are given by:

$$\begin{aligned} G_t(z_t, a, l, \mathbf{x}) &= W_{a,l}[z_t]^T \text{LSTM}(\text{Emb}(\mathbf{x}), \text{Emb}(l, a))[t]) \\ F_t(y, z_t, a, l, \mathbf{x}) &= \begin{pmatrix} \theta_{\text{none}} & 0 & 0 \\ 0 & & \\ 0 & & M^t \end{pmatrix} [y, z_t] \end{aligned} \tag{7}$$

where $M^t = W_M^T \text{LSTM}(\text{Emb}(\mathbf{x}), \text{Emb}(a, l))[t]$ is a matrix that dictates the interaction between positive/negative word sentiment and positive/negative sentence sentiment. The projection $W_M \in \mathbb{R}^{D \times 2^2}$ where D is the dimensionality of the LSTM's output. M^t is then reshaped into an 2×2 matrix. The potentials of the `CRF-Emb-LSTM` model, are similar:

$$\begin{aligned} G_t(z_t, a, l, \mathbf{x}) &= W_{a,l}[z_t]^T \text{Emb}(\mathbf{x})[t]) \\ F_t(y, z_t, a, l, \mathbf{x}) &= \begin{pmatrix} \theta_{\text{none}} & 0 & 0 \\ 0 & & \\ 0 & & M^t \end{pmatrix} [y, z_t] \end{aligned} \tag{8}$$

where M^t is defined above.

B.3.3. Training details

Since we treat each tuple $(\boldsymbol{x}, a, l, y)$ as a separate example, we create a class imbalance problem as most sentiments are *none*. Thus during training we subsample to ensure that every batch has an equal number of none, positive, and negative sentiments. In each epoch we iterate over all examples from the smallest class, in this case the *negative* class, and subsample the rest. In all experiments we use a batch size of 33 during training. We utilize the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.01 and default momentum parameters. We do not decay the learning rate during training and select the final model based on the validation sentiment accuracy at each epoch. We terminate training after 1000 epochs.

B.3.4. Model parameters

For all models we utilize the GloVe 840B 300D embeddings (Pennington et al., 2014) to initialize the word embeddings and do not update the word embeddings during training. Since the dataset is extremely small, we found that the word embeddings had a very large impact on performance. We also use a 2-layer BLSTM with 50 hidden dimensions. Dropout is used with probability 0.2 after the embeddings and in the BLSTM.

B.3.5. Evaluation

The primary evaluation metrics we use for Sentihood are the sentiment accuracy and the aspect macro-F1 score. We calculate accuracy over examples with non-'none' gold sentiment labels. Similarly, we follow Ma et. al. (Ma et al., 2018) in their calculation of the macro F1 score by pre-

dicting the sentiment of all location and aspect pairs for a given sentence and using the number of correctly predicted sentiments among the non-none gold sentiments for the precision and recall. We ignore sentences with no non-none gold sentiments.

C. Walking through Algorithm 1

C.1. Walking through the intractable Example 3.2

Consider the intractable model Example 3.2, which is the minimal plated factor graph for which Algorithm 1 fails:

$$\begin{aligned}V &= \{x, y\}\\ F &= \{f_{xy}\}\\ E &= \{(x, f_{xy}), (y, f_{xy})\}\\ P &= \{(x, \{a\}), (y, \{b\}), (f_{xy}, \{a, b\})\}\end{aligned}$$

On the first pass through the **while** loop, there is a single choice of leaf and a single connected component

$$\begin{aligned}L &\leftarrow \{a, b\}\\ V_L &\leftarrow \{\} && = V_c\\ F_L &\leftarrow \{f\} && = F_c\\ E_L &\leftarrow \{\} && = E_c\end{aligned}$$

At this point no variable can be eliminated since $V_c = \emptyset$:

$$\begin{aligned}f &\leftarrow \textsc{SumProduct}(\{f_{xy}\}, \emptyset) && = f_{xy}\\ V_f &\leftarrow \{x, y\}\end{aligned}$$

Since V_f is not empty, we search for a next plate set but find

$$L' \leftarrow P(x) \cup P(y) \quad = \{a\} \cup \{b\} \quad = L$$

Now since $L' = L$ the algorithm cannot progress and results in **error**.

C.2. Walking through the experimental model 6.4

Consider the experimental model of 6.4 with

$$\begin{aligned}V &= \{v, w, x, y, z\}\\ F &= \{f_{vw}, f_{wx}, f_x, f_{xy}, f_{yz}\}\\ E &= \{(v, f_{vw}), (w, f_{vw}), (w, f_{wx}), (x, f_{wx}), (x, f_x),\\ &\qquad (x, f_{xy}), (y, f_{xy}), (y, f_{yz}), (z, f_{yz})\}\\ P &= \{(v, \{a, b\}), (w, \{a\}), (x, \{\}), (y, \{b\}), (z, \{a, b\}),\\ &\qquad (f_{vw}, \{a, b\}), (f_{wx}, \{a\}), (f_x, \{\}),\\ &\qquad (f_{xy}, \{b\}), (f_{yz}, \{a, b\})\}\end{aligned}$$

On the first pass through the **while** loop, there is a single choice of leaf:

$$\begin{aligned}L &= \{a, b\}\\ V_L &= \{v, z\}\\ F_L &= \{f_{vw}, f_{yz}\}\\ E_L &= \{(v, f_{vw}), (z, f_{yz})\}\end{aligned}$$

There will be two connected components, one with v and one with z. We process them in an arbitrary order.

1. Connected component $V_c = \{v\}$, $F_c = \{f_{vw}\}$:
   We first eliminate the variable v via a sum-reduction and record that variable w remains to be eliminated:

   $$\begin{aligned}f &\leftarrow \textsc{SumProduct}(\{f_{vw}\}, \{v\})\\ V_f &\leftarrow \{w\}\end{aligned}$$

   In searching for the next plate set, we find $L' \leftarrow \{a\}$ and product-reduce plate b:

   $$f' \leftarrow \textsc{Product}(f, \{b\}, M)$$

   After adding the new factor $f' =: \hat{f}_w$ and updating data structures, we have

   $$\begin{aligned}V &= \{w, x, y, z\}\\ F &= \{\hat{f}_w, f_{wx}, f_x, f_{xy}, f_{yz}\}\\ E &= \{(w, \hat{f}_w), (w, f_{vw}), (w, f_{wx}), (x, f_{wx}), (x, f_x),\\ &\qquad (x, f_{xy}), (y, f_{xy}), (y, f_{yz}), (z, f_{yz})\}\\ P &= \{(w, \{a\}), (x, \{\}), (y, \{b\}), (z, \{a, b\}),\\ &\qquad (\hat{f}_w, \{a, b\}), (f_{wx}, \{a\}), (f_x, \{\}),\\ &\qquad (f_{xy}, \{b\}), (f_{yz}, \{a, b\})\}\end{aligned}$$

2. Connected component $V_c = \{z\}$, $F_c = \{f_{yz}\}$:
   Operating similarly with z we have

   $$\begin{aligned}V &= \{w, x, y\}\\ F &= \{\hat{f}_w, f_{wx}, f_x, f_{xy}, \hat{f}_y\}\\ E &= \{(w, \hat{f}_w), (w, f_{vw}), (w, f_{wx}), (x, f_{wx}), (x, f_x),\\ &\qquad (x, f_{xy}), (y, f_{xy}), (y, \hat{f}_y)\}\\ P &= \{(w, \{a\}), (x, \{\}), (y, \{b\})),\\ &\qquad (\hat{f}_w, \{a, b\}), (f_{wx}, \{a\}), (f_x, \{\}),\\ &\qquad (f_{xy}, \{b\}), (\hat{f}_y, \{b\})\}\end{aligned}$$

On the second pass through the **while** loop, there are two possible leaves, $L = \{a\}$ or $L = \{b\}$. Arbitrarily choosing leaf a, we set

$$\begin{aligned}L &= \{a\}\\ V_L &= \{w\}\\ F_L &= \{\hat{f}_w, f_{wx}\}\\ E_L &= \{(w, \hat{f}_w), (w, f_{wx})\}\end{aligned}$$

There is a single connected component with $V_c = V_L$, $F_c = F_L$. We eliminate variable w via a vector-matrix multiply and record that variable x remains to be eliminated:

$$\begin{aligned}f &\leftarrow \textsc{SumProduct}(\{\hat{f}_w, f_{wx}\}, \{w\})\\ V_f &\leftarrow \{x\}\end{aligned}$$

The next plate set is $L' \leftarrow \emptyset$, so we product-reduce plate a:

$$f' \leftarrow \text{PRODUCT}(f, \{a\}, M)$$

After adding the new factor $f' =: \hat{f}_x$ and updating data structures, we have

$$\begin{aligned}
V &= \{x, y\} \\
F &= \{\hat{f}_x, f_x, f_{xy}, \hat{f}_y\} \\
E &= \{(x, \hat{f}_x), (x, f_x), (x, f_{xy}), (y, f_{xy}), (y, \hat{f}_y)\} \\
P &= \{(x, \{\}), (y, \{b\}), \\
&\qquad (\hat{f}_x, \{\}), (f_x, \{\}), (f_{xy}, \{b\}), (\hat{f}_y, \{b\})\}
\end{aligned}$$

On the third pass through the **while** loop we choose $L \leftarrow \{b\}$ and similarly eliminate y, resulting in

$$\begin{aligned}
V &= \{x\} \\
F &= \{\hat{f}_x, f_x, \hat{f}'_x\} \\
E &= \{(x, \hat{f}_x), (x, f_x), (x, \hat{f}'_x)\} \\
P &= \{(x, \{\}), (\hat{f}_x, \{\}), (f_x, \{\}), (\hat{f}'_x, \{\})\}
\end{aligned}$$

On the final pass through the **while** loop, we choose $L \leftarrow \emptyset$. We eliminate x via a three-way dot product and record that no more variables remain to eliminate:

$$\begin{aligned}
f &\leftarrow \text{SUMPRODUCT}(\{\hat{f}_w, f_x, \hat{f}'_x\}, \{x\}) \\
V_f &\leftarrow \emptyset
\end{aligned}$$

Since V_f is empty we add the new factor $\text{PRODUCT}(f, \emptyset, M) =: \hat{f}$ to scalars S. Note that in this example the final PRODUCT is a no-op, however it would have been nontrivial if some plate had contained all variables, as e.g. happens when training on a minibatch of data.

Finally we **return** $\text{SUMPRODUCT}(\{f\}, \emptyset)$; again this final operation is a no-op since there was only one connected component in the input plated factor graph.

D. Plates in Pyro

The Pyro probabilistic programming language provides a Python context manager `pyro.plate` to declare that a portion of a probabilistic model is replicated over a tensor dimension and is statistically independent over that dimension. An example of an independent dimension is the index over data in a minibatch: each datum should be independent of all others.

To declare a component of a model as plated, a user writes sample statements in a context, e.g.

```
with pyro.plate("my_plate", 100, dim=-1):
    x = pyro.sample("x", Bernoulli(0.5))
    y = pyro.sample("y", Bernoulli(0.1 +
        0.8 * x))
```

In the above example, the distribution at sample site `"x"` is expanded from 1 to 100 conditinally independent samples, and `x` will have shape `(100,)`. The Bernoulli distribution over `x` is already batched because its parameters depend on `x`, hence it does not need to be expanded.

Plates can be nested to account for multiple dimensions

```
with pyro.plate("x_axis", 320, dim=-1):
    # within this context, batch dimension
        -1 is independent
    with pyro.plate("y_axis", 200, dim=-2):
        # within this context, batch
            dimensions -2 and -1 are
            independent
```

Note that dimensions use negative indices to follow the NumPy convention of counting from the right of a tensor shape; this allows indices to be compatible with tensor broadcasting.

To create overlapping plates with non-strictly-nested relationship, users can create the context managers beforehand and enter the appropriate contexts at each sample statement. For example in a model where some noise depends on an x position, some noise depends only on y position, and some noise depends on both, users can write:

```
x_axis = pyro.plate("x_axis", 320, dim=-2)
y_axis = pyro.plate("y_axis", 200, dim=-3)
with x_axis:
    # within this context, batch dimension
        -2 is independent
with y_axis:
    # within this context, batch dimension
        -3 is independent
with x_axis, y_axis:
    # within this context, batch dimensions
        -3 and -2 are independent
```