Neural Dynamic Programming for Musical Self Similarity

Christian J. Walder [1 2] **Dongwoo Kim** [2 3]

Abstract

We present a neural sequence model designed specifically for symbolic music. The model is based on a learned edit distance mechanism which generalises a classic recursion from computer science, leading to a neural dynamic program. Repeated motifs are detected by learning the transformations between them. We represent the arising computational dependencies using a novel data structure, the edit tree; this perspective suggests natural approximations which afford the scaling up of our otherwise cubic time algorithm. We demonstrate our model on real and synthetic data; in all cases it out-performs a strong stacked long short-term memory benchmark.

1. Introduction

A popular approach to symbolic music modelling is to represent pieces by sequences and model them with a Recurrent Neural Network (RNN) (Eck & Schmidhuber, 2002; Fernandez & Vico, 2013; Jaques et al., 2016; Pachet et al., 2017; Briot et al., 2017; Johnson, 2017). The RNN architecture is typically based on the Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997).

Such models are useful for algorithmic composition and accompaniment; *e.g.* one may conditionally sample with human input constraints (Pachet & Roy, 2011; Walder & Kim, 2017). Without such constraints, sampling from an LSTM tends to produce music with an undesirable meandering quality that is unlike human composed music (see *e.g.* (Jaques et al., 2016) for a discussion of the issue).

Rather, human music is strongly self similar (Pareyon, 2011); a few motifs may form the building blocks for an entire piece — see *e.g.* movement one of Beethoven's *Pastoral Symphony* and its entertaining analysis by Bernstein (1973).

[1]CSIRO Data61, Black Mountain, Australia [2]The Australian National University [3]Data to Decisions CRC, Kent Town, SA, Australia. Correspondence to: Christian <christian.walder@data61.csiro.au>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

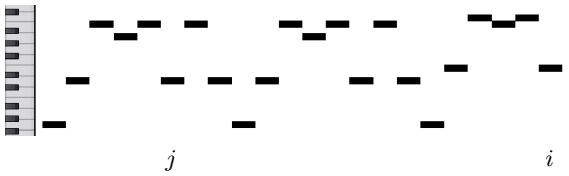


Figure 1. An excerpt from Bach's *BWV 1007* as a *piano roll* (horizontal = time; vertical = pitch as per the keyboard at left), which ends during the third repetition of a *motif*. By *analogy* with the earlier repeats, the last two notes of the excerpt will likely repeat immediately. Note that the third cycle of the motif introduces a non-trivial *diatonic* (within musical scale) shift of the upper notes. The indices i and j above are referred to in subsubsection 4.2.1.

These motific building blocks are not simply repeated verbatim throughout a piece of music; rather, as in the example of figure 1, the fragments are transformed in a musically logical way. This logic demands coherent harmony, melody and phrasing structure; this in turns requires such non-trivial motif transformations as transposition, diatonic shifting, deletion, insertion of passing tones, truncation, *etc.*

The logic of motif transformation is our focus. Like any sequence model, our model may be factorised causally as in (5) of section 4 below. The key idea is that whenever, as we are generating a sequence, we detect earlier point(s) which are motifically related, then we may expect the future to unfold analogously. This is similar in spirit to the concept of *attention* (Bahdanau et al., 2014), which has spurred lots of research interest, especially in sequence to sequence modelling for machine translation. Attentional sequence models are far less common however — the single example we are aware of is (Laird & Irvin, 2017). Indeed, naïvely modelling sequences with attention suffers from a quadratic time complexity, and is arguably less appropriate for language modelling than for machine translation.

Our model explicitly compares all sub-sequences of all lengths within the very sequence being modelled, using a dynamic program (DP) that aligns by explicitly modelling generalised edit operations; a naïve implementation has a cubic time complexity. This is a challenging example of recent research investigating structured attention and neural (or differentiable) dynamic programming; *e.g.* (Schulze et al., 2007; Lample et al., 2016; Kim et al., 2017; Cuturi & Blondel, 2017; Mensch & Blondel, 2018). The computational

structure of our algorithm may however be represented with a novel data structure which we call an *edit tree*. By only partially expanding the edit tree, we scale up sufficiently to out-perform the LSTM on synthetic and real music data.

The structure of the paper is as follows. We complete this introduction by introducing in subsection 1.1 the notation we use for both sequences and neural networks. Section 2 introduces the notion of edit distance, along with a well known DP for computing it. In section 3 we present a generic algorithm for sequence modelling, which involves comparing a sequence to itself using the edit distance. Section 4 introduces our MotifNet algorithm, which generalises all aspects of the previous section using appropriate neural network sub-modules. We introduce the edit tree in section 5, and propose an efficient MotifNet approximation based on it. Section 6 provides experiments, and section 7 summarizes our findings and contribution.

1.1. Notation

Sequences. We consider throughout ordered sequences $T = t_1 \cdot t_2 \cdots t_{|T|}$ of elements $t_i \in \Sigma$, where dots denote concatenation. The subsequence $t_i \cdots t_j, j \geq i$ of T is denoted $T(i:j)$, and has length $|T(i:j)| = j - i + 1$. Similarly $T(:j)$ is the length j prefix of T. We let our subsequence notation implicitly handle indices which are out of range by assuming that $T(i:j) = T(\max(1,i) : \min(|T|, j))$. The empty sequence is represented by ϵ. We abuse the notation by treating length one sequences like elements and vice versa, where appropriate.

Neural networks. Denote by $L(A)$ the linear transformation parametrised by $A \in \mathbb{R}^{a \times b}$ defined by $L(A) : \mathbb{R}^a \to \mathbb{R}^b; \boldsymbol{x} \mapsto L(A)(\boldsymbol{x}) = A\boldsymbol{x}$. Denote the leaky rectified linear unit by $\mathrm{ReLU}_\alpha(x) = \max(x, \alpha x)$ which we abuse the notation by allowing to operate element-wise on vectors. Denote the softmax by σ so that $\sigma(\boldsymbol{z})_i = \exp(z_i)/(\sum_j \exp(z_j))$. We may now notate *e.g.* a two layer feed-forward neural network classifier using the composition operator, *e.g.* $\sigma \circ L(A_2) \circ \mathrm{ReLU}_\alpha \circ L(A_1)$. To obtain symmetry, we employ the *pseudo-Huber* $\mathrm{PH}(x, \delta) = \delta^2 \left(\sqrt{1 + (x/\delta)^2} - 1\right)$, a smoothed absolute value, in subsection 4.1 .

2. Edit Distance

Our model features a learned generalization of the notion of edit distance between strings. The edit distance is a widely used dis-similarity measure between sequences.

Definition 1 (edit operations) *Operations $\epsilon \to a$, $a \to \epsilon$ and $a \to b$, for $a, b \in \Sigma, a \neq b$ are called **insertion**, **deletion** and **substitution** operations, respectively. Operation $a \to a$ is called the **identity** operation.*

	-	G	A	T	C	G	T	C	G	A	T	C
-	0	0	0	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	0	1	1	0	1	1	1
A	2	1	0	1	2	1	1	2	1	0	1	2
T	3	2	1	0	1	2	1	2	2	1	0	1
C	4	3	2	1	0	1	2	1	2	2	1	0

Figure 2. An example of the matching algorithm of subsection 3.1 with $T =$ "GATCGTCGATC" and $P =$ "GATC", and unit costs c. We tabulate $D(i, j)$ with the elements of P (respectively T) as the row (column) labels. The dashes are place holders for the zero index labels. Note that the bottom row gives the edit distance from P to the corresponding suffix of T which ends at the given column, so that the two zeros therein correspond to exact matches of P.

Definition 2 (trace) *A trace of A and B is any sequence $\Gamma = x_1 \to y_1 \cdot x_2 \to y_2 \cdots x_h \to y_h$ of edit operations such that $A = x_1 \cdot x_2 \cdots x_h$ and $B = y_1 \cdot y_2 \cdots y_h$. Note that ϵ allows insertion/deletion and $|A| \neq |B|$ in general.*

Definition 3 (edit distance) *Let $c(\gamma_i)$ be the non-negative cost associated with edit operation γ, and let $C(\Gamma) = \sum_{1 \leq i \leq |\Gamma|} c(\gamma_i)$ be the cost of a trace. The edit distance $d(A, B)$ is the minimum cost of a trace of A and B.*

2.1. Dynamic Program for Edit Distance

Given two sequences P and T, let $D(i, j)$ be the minimum edit distance between prefixes $P(:i)$ and $T(:j)$, for $0 \leq i \leq |P|$ and $0 \leq j \leq |T|$. We may compute the matrix D with the DP scheme

$$D(i,j) = \min \begin{cases} c(p_i \to \epsilon) + D(i-1, j) \\ c(p_i \to t_j) + D(i-1, j-1) \\ c(t_j \to \epsilon) + D(i, j-1) \end{cases} \quad (1)$$

The $\min$ is over terms with valid indices into D (*e.g.* if $i > 0$ and $j = 0$, only the top term is considered) — this notational convenience simplifies special cases $D(i, 0)$ and $D(0, j)$ as in *e.g.* (Sellers, 1980). After initializing

$$D(0, 0) = 0, \quad (2)$$

$\mathcal{O}(|P|\,|T|)$ work yields $d(P, T) = D(|P|, |T|)$.

3. A Generic Autoregressive Sequence Model

We now set up our computational architecture for the simpler analogous case of traditional edit distance, in two steps. In subsection 3.1 and subsection 3.2 we derive a DP for computing the edit distance from the suffix to all other substrings. Then in subsection 3.3 we present a general scheme for utilizing the distances computed in the previous step, for forecasting (and therefore sequence modelling).

Algorithm 1 MotifNet generalised distance.

Input: $S = s_1 \cdot s_2 \cdots s_{|S|}, f_E, f_A, f_S, f_D, D_0$
Output: $D(i,j,k)$
for $i = 1$ **to** $|S|$ **do**
  for $k = 1$ **to** i **do**
    if $k = 1$ **then**
      for $j = 1$ **to** i **do**
        $D(i,j,k) \leftarrow f_A(D_0,\, f_S(s_i, s_j))$
      end for
    else
      for $j = 1$ **to** i **do**
        $D_\downarrow \leftarrow f_A(D(i-1,j,k-1),\, f_D(s_i))$
        $D_\searrow \leftarrow f_A(D(i-1,j-1,k-1),\, f_S(s_i, s_j))$
        $D_\rightarrow \leftarrow f_A(D(i,j-1,k),\, f_D(s_j))$
        $D(i,j,k) \leftarrow \operatorname{argmax}_{D' \in \{D_\downarrow, D_\searrow, D_\rightarrow\}} f_W(D')$
      end for
    end if
  end for
end for

3.1. Dynamic Program for String Matching

Sellers' modification (Sellers, 1980) of the DP of subsection 2.1 involves initializing $D(0,j) = 0$ for all j (rather than just $j = 0$ as per (2)), and applying (1) for $i > 0$; see *e.g.* figure 2. The bottom row $D(|P|, j)$ gives the minimum edit distance from P to $T(j' : j)$ for any $j' \leq j$. That is, we match only the suffix $T(j' : j)$ of $T(: j)$.

3.2. Self Matching

Given a sequence S, let, for $j \leq i$, $D_s(i,j,k)$ be the minimum edit distance from $S(i-k+1 : i)$ (roughly, the subsequence of length k ending at i; recall subsection 1.1) to the suffix $S(j' : j)$ of $S(: j)$, for any $j' \leq j$. We interpret $D_s(i,j,k)$ as the shortest distance from the subsequence of length k ending at i to the suffix ending at any j satisfying

$$j \leq i. \tag{3}$$

To derive a recursion analogous to (1), we apply the scheme of subsection 3.1. Let $D_{X,Y}$ be the matrix obtained by applying the scheme of subsection 3.1 with $P = X$ and $T = Y$. Furthermore let $P_i = S(i-k+1 : i)$ and $T_j = S(: j)$. This notation allows the precise definition

$$D_s(i,j,k) = D_{P_i,T_j}(|P_i|, |T_j|),$$

as well as the relations

$$D_s(i-1,j,k-1) = D_{P_i,T_j}(|P_i| - 1, |T_j|)$$
$$D_s(i,j-1,k) = D_{P_i,T_{j-1}}(|P_i|, |T_{j-1}|).$$

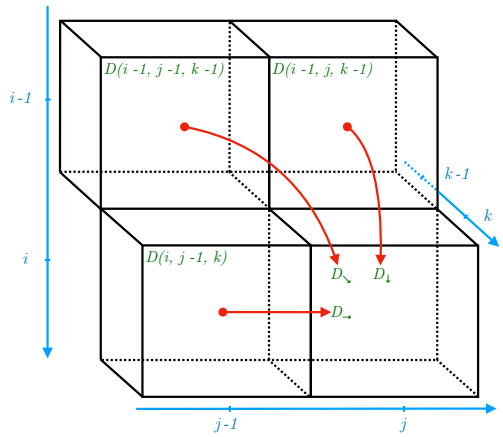


Figure 3. A diagram of the information flow (red arrows) in the generalised distance tensor $D(i,j,k)$, as per (4) and algorithm 1. The red arrows correspond to edit operations; *e.g.* deletion of s_j in the case of the lower red arrow.

Combining these relations with (1) we obtain that, for $k > 1$

$$D_s(i,j,k) = \min \begin{cases} c(s_i \to \epsilon) + D_s(i-1,j,k-1) \\ c(s_i \to s_j) + D_s(i-1,j-1,k-1) \\ c(s_j \to \epsilon) + D_s(i,j-1,k). \end{cases} \tag{4}$$

In this case the min is over terms for which both

1. the indices into D_s are non-negative (as in (1)), and
2. the second index into D_s is less than or equal to the first. This affects the first term only, and for $i = j$ only, and is due to the assumption (3). Without this condition, slices $D_s(i,:,k)$ would depend on s_{i+1}, violating temporal causality in the model which follows.

One may view $s_i \to \epsilon$ (resp. $c(s_j \to \epsilon)$) as insertion (resp. deletion), or vice versa, depending on the perspective. We model them symmetrically and refer to both as *deletion*.

3.3. Forecasting / Sequence Modelling

Since in general

$$p(S) = \prod_{i=0}^{|S|-1} p(s_{i+1} | S(: i)), \tag{5}$$

we may model a sequence S by assuming $S(: i)$ is given, and predicting the next element s_{i+1}. A natural approach is to compare suffixes $S(i-k+1 : i)$ for various k, to the previous sub-sequences $S(j' : j)$ for some $j' \leq j \leq i$. Hence $D(i,j,k)$ is useful: an exact match $D_s(i,j,k) = 0$ for some j and k, may suggest the continuation s_{j+1} to re-occur (that is, equal s_{i+1}). Generally, we may forecast

(a) functions

Notation	Role	Architecture	Mapping
f_E	embedding	lookup	$\Sigma \to \mathcal{E}$
f_D	deletion	FF	$\Sigma \to \mathcal{C}$
f_S	substitution	FF	$\Sigma \times \Sigma \to \mathcal{C}$
f_A	addition	GRU	$\mathcal{D} \times \mathcal{C} \to \mathcal{D}$
f_W	scoring*	FF	$\mathcal{D} \to \mathbb{R}$
f_G	analogy	FF	$\mathcal{D} \times \mathcal{E} \to \mathcal{O}$
f_F	forecasting	FF	$\mathcal{O} \to \mathbb{R}^{\lvert\Sigma\rvert}$

(b) sets & spaces

Notation	Interpretation of Elements
$\Sigma = \{1, 2, \ldots, \lvert\Sigma\rvert\}$	Discrete symbol
$\mathcal{E} = \mathbb{R}^{N_E}$	Embedding
$\mathcal{C} = \mathbb{R}^{N_C}$	Generalised edit cost
$\mathcal{D} = \mathbb{R}^{N_D}$	Generalised distance
$\mathcal{O} = \mathbb{R}^{N_O}$	Penultimate layer

Table 1. Our function modules and the spaces they act on. *FF* and *GRU* stand for *feed-forward* and *gated recurrent unit*, respectively. *Note that the scoring function f_W performs several roles.

using all $D(i, j, k)$ and corresponding next symbols s_{j+1}, for $1 \leq j < i$ and $1 < k < i$ by modelling

$$s_{i+1} \,|\, S(:i) \sim \mathcal{S}\left(\{(D(i,j,k), s_{j+1})\}_{0 \leq j < i,\, 0 < k < i}\right) \quad (6)$$

where $\mathcal{S}$ is a random variable parametrised by a set of (distance, symbol) pairs; *e.g.* $\mathcal{S}$ may place higher probability on symbols s_{j+1}, with smaller corresponding $D(i, j, k)$.

This scheme subsumes various sequence prediction algorithms, *e.g.* variable length Markov models, prediction suffix trees, and on-line sequence prediction algorithms (Willems et al., 1995; Saul & Jordan, 1999; Dekel et al., 2004).

4. Motif Networks

The previous section 3 presented a general framework for sequence prediction. We now generalise that framework to obtain our novel deep learning architecture. The components of our model are summarized in table 1.

4.1. Motif Network Distance Tensor

The neural dynamic program for $D(i, j, k)$ is given as algorithm 1, and utilises the following learned modules:

- **Embedding vectors** $\{\boldsymbol{e}_s\}_{s \in \Sigma} \subset \mathcal{E}$ of our symbols $s \in \Sigma$, so that $f_E(s) = \boldsymbol{e}_s$.
- The **cost of edit operations** is learned using feed-forward neural networks. For **deletion** $s_i \to \epsilon$ we let, in the notation of subsection 1.1, $f_D = g_D \circ f_E$ where $g_D = \mathrm{ReLU}_\alpha \circ L(A_D^{(2)}) \circ \mathrm{ReLU}_\alpha \circ L(A_D^{(1)})$. The **substitution** operation $s_i \to s_j$ is slightly more subtle; for reasons of efficiency (see subsection 5.4) and parsimony, we employ a *symmetric* function of two arguments. Symmetry is obtained via a differentiable approximation to the absolute value function, namely PH of subsection 1.1. In particular we let $f_S(s_i, s_j) = g_S(\mathrm{PH}(f_E(s_i) - f_E(s_j)), \delta))$, where g_S is of the same form as (but does not share parameters with) g_D, and $\delta = \frac{1}{2}$ throughout.
- We let the elements of the **distance tensor** $D(i, j, k)$ be vector rather than scalar, and we denote this space of generalised distances by $\mathcal{D}$. This requires generalization of the notion of **addition** on the r.h.s. of (4); we define $f_A : \mathcal{D} \times \mathcal{C} \to \mathcal{D}$, and we assume a Gated Recurrent Unit architecture (GRU) (Cho et al., 2014) where distance is the latent state, with initial value D_0. The GRU is slightly more appropriate than the LSTM cell in that we don't require an output gate.
- We generalise the $\min$ in (4) to $\arg\max_D f_W(D)$ where $f_W = L(A_W)$ is a scalar valued **scoring** function (*i.e.* $A_W \in \mathbb{R}^{N_D \times 1}$). A soft max alternative is addressed in the ablative study of subsection 6.1.

4.2. Motif Network Forecast

We generalise (6) in a manner designed to model the various self similarities inherent in symbolic music. The key concept we introduce is that of the *analogy* function $f_G(\boldsymbol{d}, \boldsymbol{e}) = g_G((\boldsymbol{d}^\top, \boldsymbol{e}^\top)^\top)$. Here g_G, which acts on the concatenation of $\boldsymbol{d}$ and $\boldsymbol{e}$, is a two layer feed-forward network, $g_G = \mathrm{ReLU}_\alpha \circ L(A_A^{(2)}) \circ \mathrm{ReLU}_\alpha \circ L(A_A^{(1)})$.

We apply the analogy function in this way to all motif positions j and lengths k, and take a weighted average, so

$$O_i = \sum_{0 \leq j < i} \sum_{0 \leq k < i} w_{i,j,k} f_G(D(i,j,k), f_E(s_{j+1})), \quad (7)$$

where the weights are obtained by applying a softmax to $f_W(D(i, j, k))$, with the same f_W as algorithm 1:

$$w_{i,j,k} = \frac{\exp(f_W(D(i,j,k)))}{\sum_{0 \leq j' < i} \sum_{0 \leq k' < i} \exp(f_W(D(i,j',k')))} \quad (8)$$

The conditional probability mass function for $s_{i+1}|S(:i)$ on the r.h.s. of (5) is then given by $f_F(O_i)$, where the forecasting function is another two layer feed-forward neural network, $f_F = \sigma \circ L(A_F^{(2)}) \circ \mathrm{ReLU}_\alpha \circ L(A_F^{(1)})$.

4.2.1. Analogy function f_G: intuition

Given *e.g.* an earlier sub-sequence $3, 7, 5, 7$, what likely follows $13, 17, 15$? Here, our generalised distance should

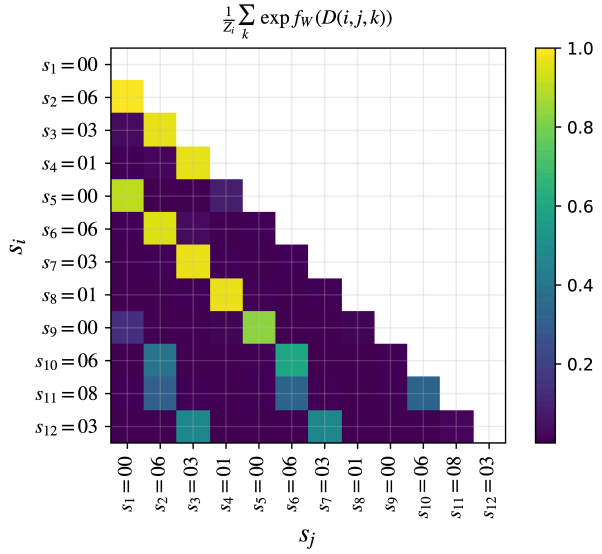


Figure 4. Visualising the scoring function f_W on a test sequence from the `markov_editloop` toy dataset. Brightness indicates alignment for all rows i and columns $j < i$, summed over length k and normalised (see the title). Values s_i and s_j are given in the axis labels. Rows $i = 5$ to $i = 8$ align the motif, while rows $i > 8$ average over two valid alignments. The insertion noise $s_{11} = 8$ is handled as evidenced by the similarity between rows $i = 10$ and $i = 11$. See subsection 6.1 for more details.

encode the transformation "add 10", which f_G should apply to 7 yielding 17. In line with (6), f_G acts on $D(i, j, k)$ and $f_E(s_{j+1})$, and returns information pertinent to predicting s_{i+1}. For another example, with i and j as marked in figure 1, and suffix length $k = 6$, $D(i, j, k)$ captures the relationship (*generalised distance* is a slight misnomer) between the first six and last six notes. Given the earlier continuation s_{j+1}, the analogy function may predict that s_{i+1} would complete the analogy (in this case we expect the second last note s_{i-1} to re-occur as s_{i+1}).

4.3. Combining with a Recurrent Neural Network

We may combine the Motif Network with a traditional recurrent neural network such as that of the Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997). To this end we apply the usual LSTM recurrence with the embeddings $f_E(s_i)$ as the input for the i-th time step. We then concatenate the LSTM output with the Motif Network output O_i of (7) before applying f_F.

5. Efficient Approximation with Trees

From *e.g.* (7) we see that the cost of forecasting s_{i+1} given $S(: i)$ is quadratic in i, leading to a cubic time complexity $O(|S|^3)$ for the entire sequence. These calculations involve redundancies which we can exploit. In contrast to existing literature focusing on suffix trees (Ukkonen, 1993; 1995; Willems et al., 1995; Dekel et al., 2004), the natural data structure in our case is the *edit* tree. This is a natural consequence of modelling transformations between arbitrary sub-sequences of a sequence, rather than the relationship between immediately subsequent sub-sequences.

5.1. Edit Trees

From algorithm 1 it is evident that each $D(i, j, k) \in \mathcal{D}$ is a function of a sequence of edit operations which correspond to an alignment of sub-strings of S. We track these edit operations using a rooted tree $G = (V, E)$ whose edges $E = \Sigma \times (\Sigma \cup \epsilon)$ correspond either to deletion $s \to \epsilon$; $s \in \Sigma$ or substitution $s \to s'$; $s, s' \in \Sigma$. Let $\nu : V \to \mathcal{D}$ map nodes to distances, and (loosely speaking) let $f_{SD}(e)$ take on the value $f_S(e)$ (respectively $f_D(e)$) for e corresponding to substitution (respectively deletion). For all vertices v, v' connected by e and satisfying $\mathrm{parent}(v') = v$, ν satisfies $v(\mathrm{root}(G)) = D_0$ as well as the recursion $\nu(v') = f_A(\nu(v), f_{SD}(e))$; $v' \neq \mathrm{root}(G)$.

5.2. Priority Queue

We assume that the number of useful alignments is small compared to the quadratic total number. These fruitless alignments are benign in that they may obtain insignificant weights $w_{i,j,k}$, as learned by the function f_W, and thereby have little effect on the forecast due to (8).

This suggests the following approximation. Rather than computing the entire tree, we expand only the most promising branches using f_W as our heuristic. The mechanism we propose is to only expand children from those nodes which have a heuristic score which is among the n_{priority} best among its siblings. More precisely, when algorithm 1 would lead to a new vertex v being created in G as a child of some vertex v' (so that $v' = \mathrm{parent}(v)$), we compute the descending rank order statistic of the associated heuristic $f_W(\nu(v))$, among that of the extant siblings of the parent, $\{f_W(v'')\}_{v'' \in \mathrm{siblings}(v')}$. If the rank exceeds some parameter n_{priority}, then v is omitted from the tree, and the associated elements of $D(i, j, k)$ (those satisfying $D(i, j, k) = \nu(v)$) are omitted from (7) and (8). This is the third role assumed by f_W (after (8), and the $\arg\min$ in algorithm 1).

5.3. Bounded Tree Depth and Suffix Length

We further assume that the length of alignment need not exceed a threshold, $d_{\max}$. Vertices beyond this tree depth are omitted, along with associated terms in (7) and (8). We further bound by $d_{\max}$ the length of suffix used for prediction, thereby considering only $D(i, j, k)$ for $k \leq d_{\max}$.

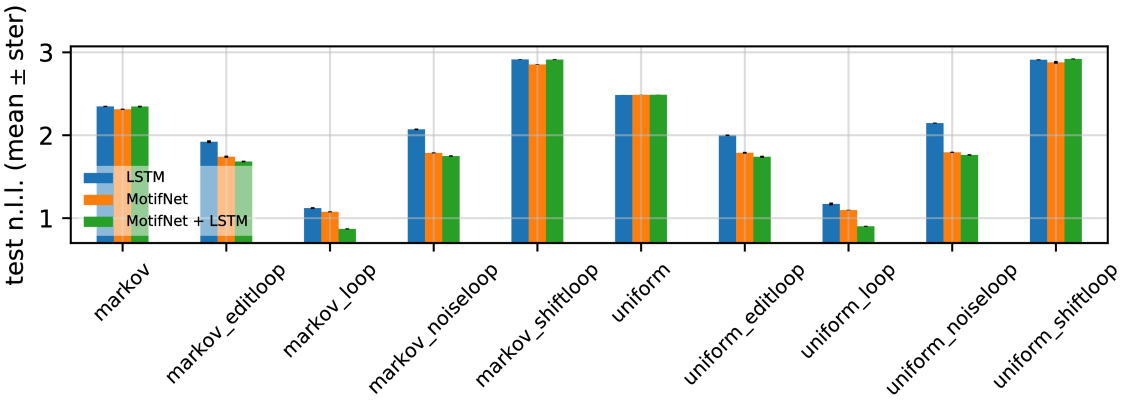


Figure 5. Average test set negative log likelihood (with barely visible ± 1 *standard error* bars) for a stacked LSTM, MotifNet, and their combination (see subsection 4.3), on a suite of toy problems. See subsection 6.1 for more details.

5.4. Computational Complexity

The maximum fan-out of the edit tree is $O(|\Sigma|^2)$ due to there being $|\Sigma|\,(|\Sigma|-1)/2$ unique substitutions (recall that f_S is symmetric), and $|\Sigma|$ deletions. By limiting the tree depth the required number of tree nodes (and associated unique generalised distance tensors) is therefore $O(|\Sigma|^{2d_{\max}})$. The forecast (7) combines these distance tensors with the other argument $f_E(s_{j+1})$ (of which there are $|\Sigma|$ unique values) to the analogy function f_G. This is done per time-step, leading to an overall time complexity of $O(|S|\,|\Sigma|^{2d_{\max}+1})$.

In the worst case, the priority queue does not reduce the effective fan-out of the tree (as the children may be added in ascending order of the value of the heuristic f_W). In the best case, the number of required nodes is $O((n_{\text{priority}})^{d_{\max}})$. Including a factor of $|\Sigma|$ due to (7), we get an overall best case complexity of $O(|S|\,|\Sigma|\,(n_{\text{priority}})^{d_{\max}})$. The real runtime is therefore heavily affected by the actual data distribution — see subsection 6.2 for an empirical investigation.

6. Experiments

Methodology. We used a train/validation/test scheme based on log likelihoods. We train full epochs up to three *strikes* (increases in validation set log likelihood after a training epoch). Training and validation was performed for a range of hyper-parameters. We let the spaces $\mathcal{E}, \mathcal{C}, \mathcal{D}$ have the same dimension, which we varied as $2, 4, 8, \ldots, 2048$. Three algorithms are compared throughout: the **LSTM** (Hochreiter & Schmidhuber, 1997), our **MotifNet**, and the combination **MotifNet+LSTM** of subsection 4.3.

Implementation. Our implementation relied heavily on the dynamic graph of the *PyTorch* software; nonetheless we found the tree based implementation of MotifNet to be rather more involved than, say, the LSTM. We trained with stochastic gradient descent using *Adam* (Kingma & Ba, 2014). GPUs yielded only modest speed ups, so we worked with a CPU cluster. Proper parallelization of MotifNet is the subject of ongoing research, and our results suggest that the full power of MotifNet remains to be revealed (see figure 7).

Key parameters. We allowed the LSTM variants an advantage by letting the LSTM cell be *stacked* (Schmidhuber, 1992; El Hihi & Bengio, 1995) with number of layers ranging 1, 2, 3, 4, whereas for the MotifNet we fixed f_A to be a GRU with one layer. For the toy problems of subsection 6.1 we used an exact MotifNet; for the music data of subsection 6.2 we used an approximate MotifNet with the crude setting of $d_{\max} = 4$, and n_{priority} ranging $2, 4, 8, \ldots, 256$. The hyper-parameters with best validation likelihood (per data replicate in the case of the toy data) were applied to the test set. We further ensured that the best models occurred well within the range considered for each parameter above, with the exception of the n_{priority} parameter (for which MotifNet may clearly perform even better, with larger but more computationally expensive settings — see figure 7).

6.1. Toy Problems

Data. We considered two toy processes. **`uniform`** generates each symbol s_i independently from the uniform distribution on the base alphabet $\Sigma_0 = (0, 1, \ldots, 11)$. While `uniform` has no structure, we generate structured sequences by repeating motifs generated by it. **`markov`**, is a Markov chain with initial and transition probabilities drawn uniformly at random. For each process and the five generation schemes below, we generate 300 sequences each for training, validation and testing. The process is repeated for each of 16 replicates (each of which with different `markov` parameters). The five generation schemes were:

- **(no label)**: draw a sequence of length $|S| = 12$.
- **`loop`**: a motif of length 4 is drawn from the pro-

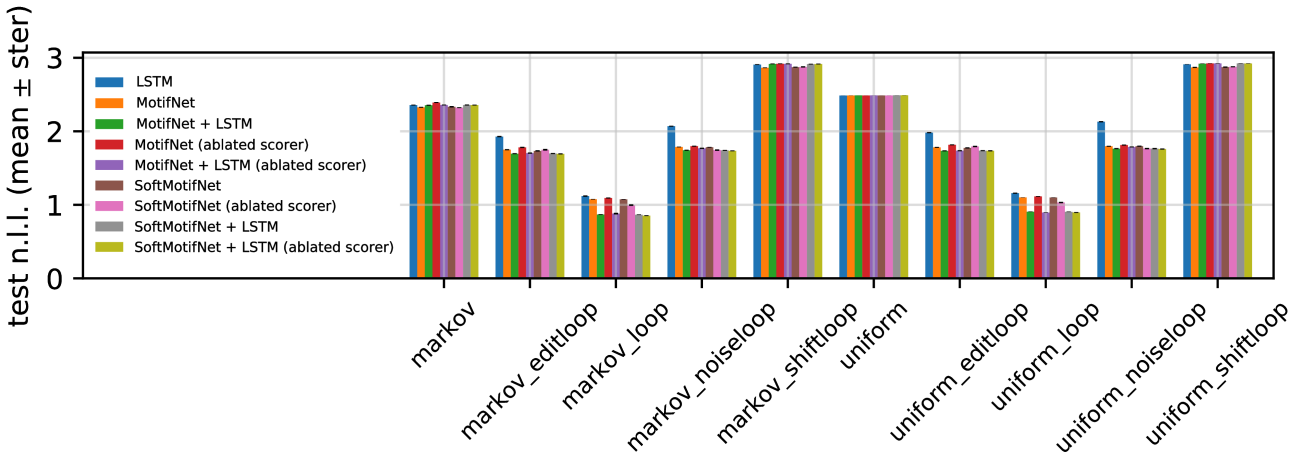


Figure 6. An expanded figure 5 which includes variations of our MotifNet. See subsection 6.1 for more details.

cess and repeated three times. For example $S = (0, 1, 5, 10, 0, 1, 5, 10, 0, 1, 5, 10)$.

- **`shiftloop`**: similar to `loop` but each repetition of the base motif is shifted by an integer drawn uniformly from (0,1,...,11). For example $S = (0, 1, 2, 10, 0 + 2, 1 + 2, 2 + 2, 10 + 2, 0 + 1, \textit{etc.})$.

- **`noiseloop`**: Similar to `loop`, but each element of the sequence is randomly (with probability 0.15) replaced with a uniform sample from Σ_0.

- **`editloop`**: Similar to `loop`, but for each element s_i we randomly (with probability 0.15) either (with probability 0.5) delete s_i or (with probability 0.5) insert a new element (drawn uniformly from Σ_0) after s_i.

Visualisation. In figure 4 we demonstrate that the learned f_W aligns the sequence with itself as intended.

Numerical results. we compare test scores in figure 5. As intended MotifNet better captures the regularity of loops in all cases. The combination MotifNet+LSTM adds a further small improvement, with the exception of `shift_loop`, where the out-performance of MotifNet is already very slight (disappointingly; this requires investigation).

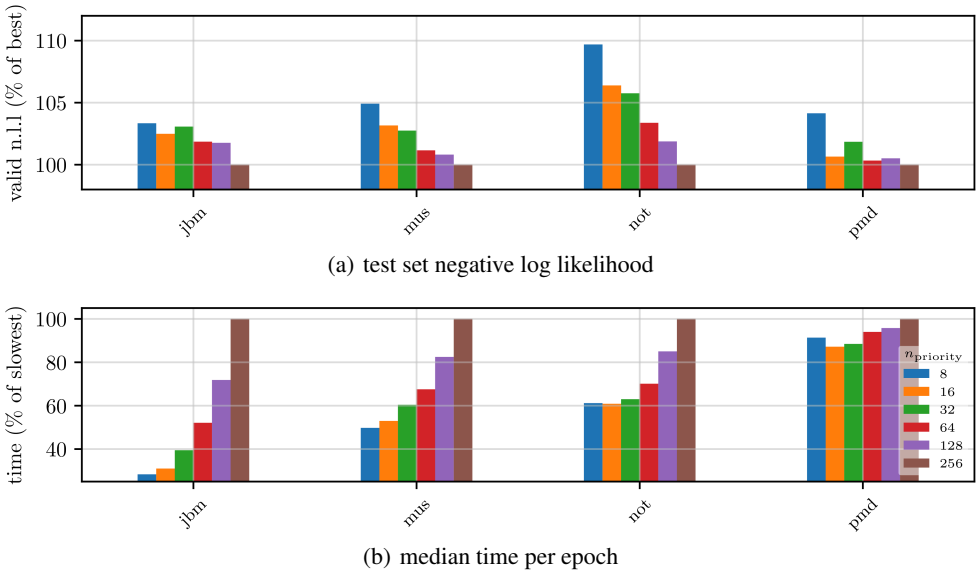
Ablative Study. In figure 6 we consider the following MotifNet variations. The first, *ablated scorer*, learns a separate f_W for the recursion (1) and forecasting (7–8). The second, *Soft*, replaces the hard max of (1) with a softmax based on f_W (similarly to (7–8), for example). The results demonstrate the point of coupling f_W across these two distinct roles in MotifNet (recall there is even a third role of f_W, namely tree pruning as per section 5). Indeed, we find that decoupling breaks the algorithm (intuitively, the hard max (4) does not permit gradient flow). While introducing the softmax heals this breakage, it also leads to non sparse gradients and, fatally, makes the tree approximation of section 5 impossible — the reason for this being that under the softmax, elements $D(i, j, k)$ would depend on all of the possible transitions (red arrows in figure 3). Importantly, the softmax with ablative scorer does not outperform (interestingly, it slightly under-performs) the basic (and computationally tractable) scheme — *e.g.* compare *MotifNet* and *SoftMotifNet (ablated scorer)* in figure 6.

Table 2. Average test set negative log likelihood for a stacked LSTM, MotifNet, and their combination (see subsection 4.3) on real symbolic music problems. See subsection 6.2 for more details.

	JBM	MUS	NOT	PMD
LSTM	1.82	2.03	1.03	2.67
MotifNet	1.77	1.88	0.81	1.90
MotifNet+LSTM	1.79	1.83	0.73	1.85

6.2. Symbolic Music Data

Data. We used the same four sets of midi files as (Boulanger-Lewandowski et al., 2012), but rather than deriving simplified piano rolls, we derived simplified note onset sequences. The Bach chorale midis of (Boulanger-Lewandowski et al., 2012) lack valid channel data, so we downloaded the analogous files from (MuseData) for that dataset. The four datasets are **JBM** (J.S. Bach chorales from (MuseData)), **MUS** (the MuseData set of Boulanger-Lewandowski et al. (2012)), **NOT** (Nottingham chord data of Shlien converted to midi by Boulanger-Lewandowski et al. (2012)) and **PMD** (piano midis provided by Krueger).



(a) test set negative log likelihood

(b) median time per epoch

Figure 7. Trading MotifNet (embedding dimensionality 2048) accuracy (a) for speed (b) with n_{priority} of section 5. See subsection 6.2.

Preprocessing. We split the files by midi channel (or instrument). We ordered the midi *onset* events (discarding other event types) by increasing time and then pitch, and retained only the midi number in our final test sequence. A more sophisticated application of our MotifNet to music generation is the subject of non-trivial ongoing work.

Numerical results. MotifNet out-performs the LSTM benchmark (see table 2). Recall that in addition to tuning the embedding dimensionality, for LSTM we tune the number of layers, whereas for MotifNet we merely tune n_{priority} of section 5. This is significant; from figure 7 see that n_{priority} does indeed merely trade computation time for accuracy. Moreover while the performance has not yet plateaued up to a value of 256, we already out-perform the LSTM. We also simply fixed the d_{max} hyper-parameter of section 5. The results we present for MotifNet are at the limit of what is computationally convenient. Further scaling up in terms of dataset size and computational effort (as parametrised by *e.g.* n_{priority}) requires algorithmic and implementation advances which are the subject of ongoing research.

Discussion. Recall that $d_{\text{max}} = 5$ limits the motif alignment length to 5 edit operations. It is interesting that this rather small value is sufficient to beat the LSTM on all music datasets. We conjecture that the reason the performance gap is smallest on the JBM set is that this dataset mainly features rather short sequences with little self similarity. As such, it is interesting that MotifNet can even match the LSTM on the JBM set, and seems to suggest that in the absence of strong motif alignments exploit, some global structure (key, harmonic sequence, *etc.*) is partially captured.

7. Summary

MotifNet combines a generalised edit distance recursion (between a sequence and itself) with an *analogy* based forecasting rule. This captures regularities in the relationships between non-adjacent sub-sequences within the same sequence. The model is qualitatively different to traditional recurrent neural networks which, notwithstanding the power of the LSTM architecture, tend to focus on the relationship between sub-sequences and their immediate continuation.

While naïvely cubic in time complexity, the computational dependencies of the MotifNet lend themselves to representation and approximation by an *edit tree*. This is a novel data structure which has edit operations (insertion, deletion, *etc.*) as edges. By partially expanding this tree using a learned heuristic function, MotifNet is able to effectively model real music sequence data.

An important novelty is the reuse of the scoring function f_W, the hardest working function in our model. Learning of f_W is permitted by the gradients which back propagate from the loss function via (7). The learned f_W then further serves by dictating the alignments found by the dynamic program of algorithm 1). The function also affords computational tractability through the termination of unpromising alignments as per subsection 5.2.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *ICLR*, 2014.

Bernstein, L. The unanswered question lecture 3: Musical semantics. Harvard, 1973. URL https://www.youtube.com/watch?v=8IxJbc_aMTg.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*, 2012.

Briot, J., Hadjeres, G., and Pachet, F. Deep learning techniques for music generation - A survey. *CoRR*, 2017.

Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, 2014.

Cuturi, M. and Blondel, M. Soft-dtw: a differentiable loss function for time-series. In *ICML*, 2017.

Dekel, O., Shalev-Shwartz, S., and Singer, Y. The power of selective memory: Self-bounded learning of prediction suffix trees. In *NIPS*, 2004.

Eck, D. and Schmidhuber, J. A first look at music composition using LSTM recurrent neural networks. *Istituto Dalle Molle Di Sull Intelligenza Artificiale*, 2002.

El Hihi, S. and Bengio, Y. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, 1995.

Fernandez, J. D. and Vico, F. J. AI methods in algorithmic composition. *JAIR*, 2013.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

James Kent, W. Blat - the blast-like alignment tool. 12: 656–64, 05 2002.

Jaques, N., Gu, S., Turner, R. E., and Eck, D. Tuning recurrent neural networks with reinforcement learning. *CoRR*, 2016.

Johnson, D. D. Generating polyphonic music using tied parallel networks. In *CIMSAD*, 2017.

Kim, Y., Denton, C., Hoang, L., and Rush, A. M. Structured attention networks. In *ICLR*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, 2014.

Krueger, B. URL http://www.piano-midi.de.

Laird, D. and Irvin, J. Autoregressive attention for parallel sequence modeling. Technical report, Stanford University, 2017.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. Neural architectures for named entity recognition. In *NAACL*, 2016.

Largeron-Leténo, C. Prediction suffix trees for supervised classification of sequences. *PRL*, 2003.

Mensch, A. and Blondel, M. Differentiable dynamic programming for structured prediction and attention. In *ICML*, 2018.

MuseData. URL http://www.musedata.org.

Pachet, F. and Roy, P. Markov constraints: steerable generation of markov sequences. *Constraints*, 16(2):148–172, March 2011.

Pachet, F., Papadopoulos, A., and Roy, P. Sampling variations of sequences for structured music generation. In *ISMIR*, 2017.

Pareyon, G. *On Musical Self-Similarity : Intersemiosis as Synecdoche and Analogy*. 2011.

Saul, L. K. and Jordan, M. I. Mixed memory Markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, 1999.

Schmidhuber, J. Learning complex, extended sequences using the principle of history compression. *Neural Comput.*, 1992.

Schulze, U., Hepp, B., Ong, C., and Rätsch, G. Palma: mrna to genome alignments using large margin algorithms. *Bioinformatics*, 2007.

Sellers, P. The theory and computation of evolutionary distances: Pattern recognition. *Algorithms*, 1, 1980.

Shlien, S. Nottingham dataset. URL http://ifdo.ca/~seymour/nottingham/nottingham.html.

Ukkonen, E. Approximate string matching over suffix trees. In *Combinatorial Pattern Matching*, 1993.

Ukkonen, E. On-line construction of suffix trees. *Algorithmica*, 1995.

Walder, C. and Kim, D. Computer assisted composition with recurrent neural networks. *ACML*, 2017.

Willems, F. M. J., Shtarkov, Y. M., and Tjalkens, T. J. The context tree weighting method: Basic properties. *Transactions on Information Theory*, 1995.