An HBPL-based Approach to the Creation of Six-word Stories

Brad Spendlove, Nathan Zabriskie, and Dan Ventura
Computer Science Department
Brigham Young University
Provo, UT 84602 USA
brad.spendlove@byu.edu, nathanzabriskie@byu.edu, ventura@cs.byu.edu

Abstract

Six-word stories are a subgenre of microfiction that presents significant challenges to authors. It is difficult to express a story with so few words, let alone an impactful or elegant one, but the best examples in the genre transcend mere storytelling to elicit an emotional response from readers. Six-word stories are an interesting and as-yet-unexplored space in the field of computational creativity. We present a system description of MICROS, a computationally creative system that generates six-word stories by making use of a hierarchical Bayesian approach. We detail how MICROS generates, evaluates, and refines stories; present stories generated by MICROS; and report a preliminary evaluation of its results.

Introduction

Six-word stories are a subgenre of microfiction (short stories composed of 1000 words or less). This restrictive word count might fool the casual observer into thinking six-word stories are simple to write, but this is not the case. With so few words to work with, the author does not have time to build up complicated worlds and characters and must instead seek to create the maximum possible impact upon the reader with every word.

"For sale: baby shoes, never worn.", a famous six-word story often attributed to Hemmingway, aptly demonstrates the potential impact of this genre of microfiction. In just a few words, the reader is given a glimpse into a larger story world before being left to themselves to imagine what else might be happening "around" the text.

Authors must display a great deal of creativity to paint these mental pictures within such a small frame. They use uncommon or unconventional grammatical structures both to fit their story into the six-word limit and to further amplify the effects of their words. As a result, the forms their stories take are often unique, both grammatically and in the semantic relationships between words.

The marriage of rigid length requirements and fluid grammatical structures makes proceduralizing the writing of six-word stories particularly difficult. Previous research has focused on computational generation of short stories and poetry but not specifically on six-word stories.

Narrative systems such as MEXICA (Pérez y Pérez and Sharples 2001), STellA (León and Gervás 2014), and Fabulist (Riedl and Young 2006) create stories by tracking characters' motivations, states of being, and locations as they perform actions to progress the plot. The story artifacts these systems output take the form of sequences of simple action phrases built from a fixed set of rules.

These systems do succeed in creating interesting stories, but the methods they use to achieve that success are often not applicable to six-word stories. For example, MEXICA will identify moments of low-interest and add new actions, such as a murder, to add tension to the narrative. Conversely, six-word stories are improved by fitting more story into the same number of words; they do not have the luxury of adding length to improve a story.

One approach to writing six-word stories would be to take a longer story, such as the output of an existing story generation system, and reduce it to six words. Such a reductive approach, however, seems unlikely to result in an interesting six-word story. Although a sequence of actions may make for an interesting story, taking a single action out of its context to fit within six words would likely result in an uninteresting or nonsensical story. This precludes a six-word story generator from working with existing story generators, such as serving as a module in a collaborative system like Slant (Montfort et al. 2013).

In fact, six-word stories rarely communicate a full narrative. Instead, they spend their limited lexical resources inviting the reader's imagination to fill in the story sketched out by the text. The best stories harness that imaginative leap to elicit emotions in the reader as well. In this way, six-word stories are more similar to poetry than narrative prose. Many previously developed poetry generators operate with a similar goal: to create poems that instill a certain feeling in the reader or deal with a specific topic.

The system described in (Colton, Goodwin, and Veale 2012) evokes a specific mood by modifying similes taken from an existing corpus to shift them towards the desired emotion. The modified phrases are then placed into user-defined templates to create a full poem. Other systems such as (Toivanen et al. 2012) swap out words in existing poems to change the topic of a poem. These systems capture the relationships between words and choose words with specific relationships to create a poem. This focus on semantics in-

stead of grammar is a common thread between generating poetry and generating six-word stories.

Computationally creative systems that deal with humor and wit face similar challenges of brevity and semantic precision (Binsted and Ritchie 1994; Oliviero and Carlo 2003).

We propose approaching the creation of a story by sampling from a distribution over the space of all possible stories. Thought of this way, story creation can be understood using the framework of Hierarchical Bayesian Program Learning (HBPL) (Lake, Salakhutdinov, and Tenenbaum 2015). This has previously been demonstrated as a viable approach to computational creativity (Bodily, Bay, and Ventura 2017), and while one may be tempted to argue that it is not suitable for producing large, complex stories, it is a very general, useful framework in which to consider many possible operationalizations for different CC tasks. In particular, here we adopt this framework to describe an approach to creating six-word stories, and present an implementation of the framework that we call MICROS.

An HBPL View of Story Writing

Consider the set $\mathscr{W}$ of all possible words, and let a story $S = w_1, w_2, \ldots, w_n$ be a sequence of n words w_i with $w_i \in \mathscr{W}$. Then a probabilistic approach to the problem of story creation imposes a distribution $p(S)$ over the set $\mathscr{S}$ of all possible stories S. That is, the joint distribution $p(S) = p(w_1, w_2, \ldots, w_k)$ must be computed. If this is possible, creating a story means simply sampling from $p(S)$. Of course, for stories of any length, this distribution is likely to be intractable to compute, and thus typically some simplifying assumptions are made that allow the joint distribution to be factored in some way. HBPL suggests that there are domain-specific factorizations that both simplify the computational demands of such an approach and that exhibit explanatory power as well. For example, in the case of story writing, one might consider a factorization such as

$$p(S) = p(\kappa) \prod_{i=1}^{\kappa} p(m_i|\kappa) p(C_i|i, m_i) p(R_i|C_1, ..., C_{i-1})$$

where $p(\kappa)$ models the number of chapters in S; $p(m_i|\kappa)$ models the number of paragraphs for the ith chapter for a story with κ chapters; $p(C_i|i, m_i)$ models the ith chapter with m_i paragraphs; and $p(R_i|C_1, ..., C_{i-1})$ models the relation of the ith chapter to the previous chapters.

There are several things to note about this formalism. First, it is clearly hierarchical, and the subdistributions can be further factorized until (hopefully) they become tractable to compute. Second, the formalism says nothing about how the individual distributions of the factorization should be computed. Third, the form of the factorization imposes structure on both the story generated and on the process used for its generation; e.g., in the factorization shown here, Chapter 1 cannot depend on anything in chapters that follow it, and it must be written before any following chapters can be. Factorizing the joint distribution in a different way will admit other story and process structures. If the structure imposed by the factorization is "correct", and if the individual distributions can all be modeled tractably, the result should be "good" stories.

For the case of microfiction stories, much of the hierarchy collapses, of course, and the most complete factorization comes from an application of the chain rule[1]:

$$p(S) = p(w_1, w_2, \ldots, w_n) = p(w_{i_1}) p(w_{i_2}|w_{i_1}) \ldots p(w_{i_n}|w_{i_1}, w_{i_2}, \ldots, w_{i_{n-1}})$$

where $i_j \in [1, n]$ and $i_j \neq i_k$ unless $j = k$, so that this represents a general version of the chain rule that admits any possible permutation of word order dependency.

Alternatively, we can consider a joint conditioned on some input $E = e_1, e_2, \ldots, e_m$, where $e_i \in \mathscr{W}$:

$$p(S|E) = p(w_1, w_2, \ldots, w_n|e_1, e_2, \ldots, e_m)$$

In what follows, we refer to a particular factorization of the joint as a story *format* and to the individual distributions (factors) as story *primitives*. For now, we have restricted the MICROS system to implementing a single format with six primitives, as follows:

$$\begin{aligned} p(S|E) = p(w_1, w_2, w_3, w_4, w_5, w_6|e_1) = \\ p(w_5|e_1) p(w_6|w_5, e_1) p(w_4|w_5, w_6, e_1) * \\ p(w_1|w_4, w_5, w_6, e_1) p(w_2|w_1, w_4, w_5, w_6, e_1) * \\ p(w_3|w_1, w_2, w_4, w_5, w_6, e_1) \end{aligned}$$

which, given additional independence assumptions that we've made, can be simplified to this:

$$\begin{aligned} p(w_1, w_2, w_3, w_4, w_5, w_6|e_1) = \\ p(w_5|e_1) p(w_6|w_5) p(w_4|w_5) * \\ p(w_1|w_5, w_6) p(w_2|w_1, w_5, w_6) * \\ p(w_3|w_1, w_2, w_5, w_6) \end{aligned}$$

The MICROS System

The operationalization of the format we selected for our system's stories consists of three background nouns (which we call *punchies*) that set the stage for the story and an article/subject/verb phrase that represents the action in the story. That is, w_1, w_2, w_3 are the punchies, w_4 is an article, w_5 is the subject, and w_6 is the verb. A simple example story in this format is, "Gun. Mask. Note. The teller screams."

Our system takes as input E a subject noun e_1 and returns a story $S = w_1, w_2, w_3, w_4, w_5, w_6$ in the format described above. Rewriting that format now using the word types just discussed gives

$$\begin{aligned} p(punchy_1, punchy_2, punchy_3, \\ article, subject, verb|noun) = \\ p(subject|noun) p(verb|subject) * \\ p(article|subject) p(punchy_1|subject, verb) * \\ p(punchy_2|punchy_1, subject, verb) * \\ p(punchy_3|punchy_1, punchy_2, subject, verb) \end{aligned}$$

[1] other less complete factorizations can, of course, also be considered, e.g., $p(S) = p(w_{i_1}, \ldots, w_{i_{\lfloor n/2 \rfloor}}) p(w_{i_{\lceil n/2 \rceil}}, \ldots, w_{i_n}|w_{i_1}, \ldots, w_{i_{\lfloor n/2 \rfloor}})$.

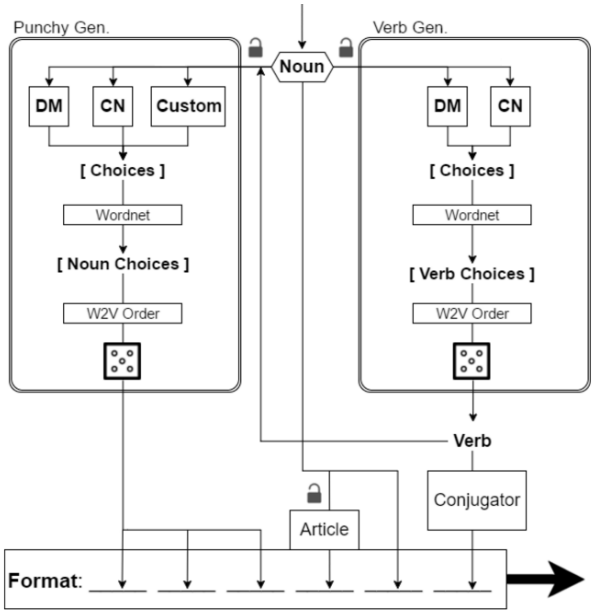

Figure 1: MICROS' six-word story generator.

These six primitive distributions are built by a generator module that then samples them to return an initial story. That story is scored with an evaluator module and passed to a refiner module that searches for higher scoring stories by resampling (some of) the primitive distributions. The refiner's output is the final six-word story artifact.

Generator

At a high level, the generator uses the input noun as the subject and then chooses a verb using the given subject. It then chooses three punchies using the subject and verb. Finally, the article is chosen, the verb is conjugated to the present tense, and the completed story is returned from the generator. Figure 1 is a diagram of this process, and the following sections detail how each primitive is constructed.

Subject and Article Modeling The subject distribution $p(subject|noun)$ is trivial to construct as all of the probability mass is placed on the input noun e_1. Sampling this distribution will therefore always return e_1 as the subject.

The article distribution $p(article|subject)$ is also simple to construct. MICROS assigns a probability of 0.5 to "the" and 0.5 to the appropriate form of "a/an" based on the chosen noun.

Verb and Punchy Modeling In order to capture the appropriate semantic relationships, the verb distribution $p(verb|subject)$ and the three punchy distributions (each conditioned on the subject, the verb, and any preceding punchies) are computed using various resources that relate words to one another. The method by which these primitive distributions are computed is similar, so we will describe them in parallel.

Both primitive generators start by collecting a set of words related to the words on which the primitive is conditioned. This is represented in Figure 1 with the inputs to the generators (the Noun for the verb generator, and the Noun and Verb for the punchy generator) being first fed into different modules that find related words. Two of these modules, labeled "DM" and "CN", query APIs for words that are related to the input words in different ways. The punchy generator has an additional related-words module called "Custom" that will be described below.

The module labeled "CN" queries ConceptNet (Speer and Havasi 2012), which stores English terms and their relationships to one another. ConceptNet defines 28 different relations that represent the ways that terms can be related. MICROS' verb generator draws on the "CapableOf", "UsedFor", and "Desires" relations while the punchy generator gathers candidate words from the "HasSubevent", "Causes", "HasPrerequisite", and "UsedFor" relations.

ConceptNet is populated from a variety of sources including semantic knowledge bases, dictionaries, and crowdsourced data. Probably due to these disparate sources, the terms and relations may be populated sparsely, are not comprehensive, may include duplicates, and sometimes reflect obscure senses of words. For example, although the word "actor" has a highly populated "CapableOf" relation, the related terms include both useful phrases like "act in a play" and "star in a movie" and oddities such as "cake on makeup" and "milk a part".

Our generator also draws from a knowledge base called Datamuse [2], represented by the "DM" module in Figure 1. Datamuse takes a word as input and returns a list of words that match certain constraints. MICROS uses the constraint called "triggers", which relates a word to other words that are statistically likely to be seen in the same literary context. For example, words that are triggered by "baby" include "boomer" and "doll".

This example also reveals a limitation with DataMuse. "Boomer" and "doll" are both words that are statistically likely to appear with "baby" in text, but they are semantically related to "baby" in radically different ways.

The final source of word relationships MICROS uses to build primitive distributions is word2vec (Mikolov et al. 2013). Word2vec is a natural language processing model that embeds words in a vector space. Using a large corpus, word2vec constructs a high-dimensional space in which each word in the corpus is represented as a vector from the origin to an associated point.

It has been demonstrated that the geometry of this vector space can represent semantic relationships between words. For example, the vector "king" minus the vector "man" plus the vector "woman" results in the vector "queen". The ability to perform vector operations on words is powerful, and the full potential of these types of approaches is still being explored.

MICROS uses word2vec to create a custom relation which we call "caused by". This relation (represented in Figure 1 by the "Custom" module) is formed by calculating the vector between example word pairs such as "eat, hunger", "drink, thirst", and "scream, fear." The vectors be-

[2] http://www.datamuse.com/api/

tween the two words in each pair are averaged together to create a relation vector. That relation vector is added to the vector of an input word, yielding a point in the word2vec space. Word vectors that are close to that point are likely to be "caused by" the input word. For example, for the input verb "help" our "caused by" custom relation returns words such as "humanitarian", "concern", and "compassion".

The outputs of these modules (DM and CN for the verb generator and DM, CN, and Custom for the punchy generator) are collected as a list of related words, which we will refer to as "choices". The choices are then filtered to include only single words of the correct part of speech, labeled in Figure 1 as "Verb Choices" and "Noun Choices", respectively. This is accomplished by querying WordNet, a large lexical database (Miller 1995).

WordNet stores words in semantically related groups called synsets that contain properties such as part of speech. MICROS also uses WordNet to discard duplicate choices that are not identical strings by taking advantage of a function called "morphy" that reduces words to their base forms, e.g. singular nouns or infinitive verbs.

Once the choices are stripped down to unique, single words of the correct part of speech, MICROS employs word2vec to score them by their similarity to words previously sampled from any primitives further up the hierarchy. Thus, the verb generator scores its choices compared to the noun, and the punchy generator scores its choices compared to the noun and the verb. Each choice word is embedded as a word2vec vector, and the cosine similarities between that vector and the vector representation of each of the preceding primitive words are summed to give a score. The lists of choices are sorted according to these scores, in ascending order, and are labeled "W2V Order" in Figure 1.

Choices that are very dissimilar from the input words are likely to be unrelated or incoherent, so the bottom fifth of the ordered list is discarded. As an example, when punchy choices are generated for the subject/verb pair "cowboy rides" the most similar words include "outlaw", "bandit", and "desperado", and the least similar words include "parks" and "symposium".

After the most dissimilar words are discarded, the generator builds the distribution $p(w_i|\dots)$ by assigning a probability to each word in the choice list, proportional to its word2vec similarity score. The generator then samples from this distribution and inserts the chosen word into the story (represented by a dice icon in Figure 1).

As seen in Figure 1, the output of the verb generator is both used as input to the punchy generator and fed into a conjugator module. Conjugation, article agreement, and later pluralization are all accomplished using the pattern.en Python library[3]. This library uses pattern matching rules and exceptions to do simple grammar tasks and is much faster than a more comprehensive dictionary-style lookup.

The punchy generator takes the verb and noun as input and computes the three punchy primitives, which differ only in that choice words that have been sampled from previous draws from a punchy primitive are assigned a probability of 0 so that punchies cannot be repeated. The punchies sampled from the three punchy primitives are slotted into the story as the first three words, followed by the article, input noun, and conjugated verb. This forms the six-word story output by the generator.

[3]https://www.clips.uantwerpen.be/pages/pattern-en

Resampling After MICROS generates $p(S|E)$ it can selectively resample primitive distributions in order to create a new story. This is done by "locking" certain words and resampling the remaining words, which mutates the original story. For example, to change the story "Humanity. Adventure. Rider. The wizard quests.", all of the words except "adventure" could be locked so only the punchy primitive $p(punchy_2|$'humanity', 'wizard', 'quest'$)$ would be resampled. The lockable primitives are represented in Figure 1 by a padlock icon.

Before sampling each primitive distribution, the generator first checks to see if that word is locked. If it is, then the locked word remains w_i rather than being (re-)sampled from the corresponding primitive distribution.

The factorization hierarchy described above affects how primitives are generated with locks. If a primitive higher in the hierarchy is unlocked and resampled, all locks for primitives lower than it will be ignored and the system will build new distributions for those primitives and sample from them. Otherwise, the system would keep punchies that are unrelated to the current verb, for example, violating the semantic relationship between the primitives. The generator's ability to mutate stories one primitive at a time will become relevant in the discussion of refinement below.

Caching Populating each primitive distribution with related words from ConceptNet and Datamuse requires network API calls that are slow to execute. In order to minimize API requests, the generator caches distributions for later reuse when the story is mutated by the refiner module.

The verb primitive distribution $p(verb|subject)$ is built once and cached. Because the subject never changes during MICROS' execution, this cached distribution can be resampled for free as many times as necessary. Punchy primitive distributions are cached on a per-verb basis and may or may not be resampled during the execution of the refiner.

Although building each distribution takes only a few seconds, this process may occur many times as verb primitives are unlocked and new punchy primitives are built. Our testing showed that MICROS' caching scheme saves an average of 100 seconds over the system's 10 minute runtime.

Exploration

The preceding section described how the generator builds a distribution $p(S|E)$ from which to sample a story S. Once the first story for a given input has been sampled, MICROS evaluates it and refines it to create progressively better stories by modifying the primitive distributions. This section details the operation of the evaluator and refiner modules used to complete this process.

Evaluation MICROS' evaluator is based on skip-thought vectors (Kiros et al. 2015), which are mappings of natural language sentences to a high-dimensional vector space, with

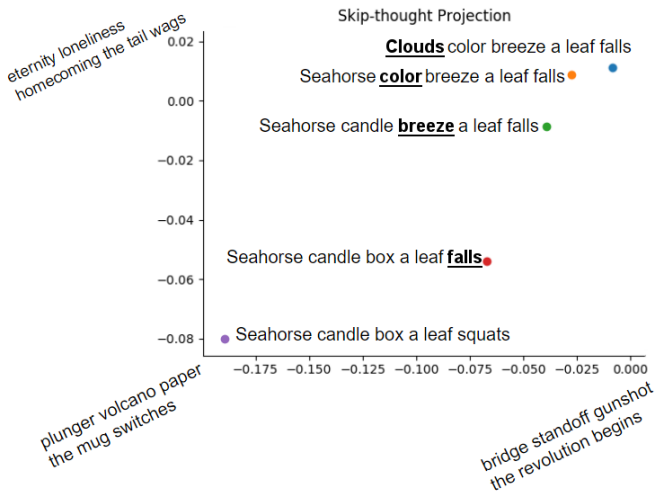

Figure 2: Example of skip-thought scores for progressively more coherent six-word stories. The underline indicates which word changed as the story mutates one word at a time from "Seahorse candle box a leaf squats" to "Clouds color breeze a leaf falls".

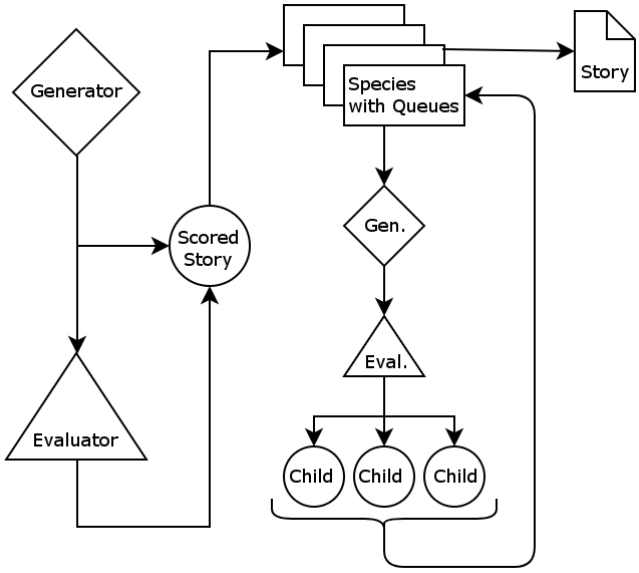

Figure 3: MICROS' refinement process.

semantically similar sentences mapping to similar vectors. In this way, skip-thoughts can be thought of as a sentence-level version of word2vec: while vectors in word2vec represent words, skip-thought vectors represent sentences.

Each time the evaluator receives a story, it first encodes it as a vector using a skip-thought encoder that was pre-trained on a large corpus of novels. It then projects this vector onto a two-dimensional plane defined by two axis vectors.

These axis vectors are calculated by subtracting the vector representation of a "bad" six-word story from the vectors of two "good" stories, all of which are hardcoded and in the same format as the generated stories. The bad vector, which forms the origin of the two-dimensional plane, is a nonsensical collection of six words while the good vectors are human-written, cohesive stories. In our system, the bad story is "Plunger. Volcano. Paper. The mug switches." and the two good stories are "Bridge. Standoff. Gunshot. The revolution begins." and "Eternity. Loneliness. Homecoming. The tail wags."

By defining the axes in this way, a story that projects to a positive coordinate will be more coherent than a story that projects to a negative coordinate. Our initial experiments showed that these axes give scores that reflect coherence reasonably well. Figure 2 shows the results of an experiment in which we took a randomly generated story and changed one word at a time to make it increasingly coherent. As can be seen, after each word change (increasing coherence), the resulting story maps to a vector with increasingly positive coordinates on the plane.

MICROS' evaluator scores each story using Manhattan distance from the origin point on the plane, yielding more positive scores for more coherent stories.

Refinement Using the generator and evaluator described above, our system refines stories through a branch-and-bound style search. Figure 3 shows how this process works.

At its most basic level, the refiner takes generated stories and places them in a priority queue according to their score. The queue is initially populated with the first generated story for the given input. At each iteration, the system dequeues the story with highest priority, saves it if it has the highest score seen so far, and mutates it several times by locking all but one of the story's words and resampling $p(S|E)$.

The resampled "children" stories are then scored and placed back into the queue to prepare for the next search iteration. By repeating this process for many iterations, the system searches for stories with higher scores. This cycle is represented in Figure 3 by the loop from the queue, through the generator and evaluator, and back into the queue.

This refinement method can unfairly favor some stories over others. Because a format's primitives exist in a hierarchy, changing the verb primitive of a story causes all of the punchy primitives to be regenerated because they are lower in the hierarchy. If MICROS mutates the verb of a story, the resulting child will differ from its parent by its three punchies and its verb. Because the child has changed so dramatically, it is likely to be less coherent and thus score lower than its parent, which may have already gone through many refinement loops with the same verb. If stories with newly-generated verbs are forced to compete against stories with already-refined verbs, they will likely continually be given low priority and never reach the front of the queue.

To mitigate this unfairness, the system speciates stories based on their verb, maintaining several species during the refinement process. A species maintains its own priority queue and tracks the highest scoring story it has seen so far. At each iteration, every species dequeues the story at

the front of its queue and mutates it to create a new group of children. MICROS then scores each child story and inserts it into the species queue that matches its verb, creating a new species if the verb has not yet been seen.

Because transitive verbs do not fit grammatically into our format, if a story is generated with a transitive verb its species is marked as "transitive" and is never dequeued or iterated upon. To determine a verb's transitivity, we query the Oxford Dictionary API[4]. This API allows only a limited number of requests per month, but because our species each represent a unique verb, we only need to query the API once per species. Although they will never be iterated upon, preserving transitive verb species allows MICROS to avoid redundant API calls.

At the end of each iteration, if a species has not improved its maximum score for a preset number of iterations, it is considered stagnant and will no longer be dequeued to produce children. Once all species are stagnant, MICROS performs a weighted choice of the top scoring stories of every intransitive species and returns it as the output of the refinement process.

Our system performs this final randomization rather than automatically returning the highest scoring story overall because the skip-thought evaluator can favor some noun/verb combinations, consistently assigning them high scores run after run. Despite this favoritism, the stories in the top-scoring species are often of comparable quality. By performing a weighted random choice, our system gives less-favored combinations a chance to be chosen as output.

The selected story is prepared for final output by pluralizing the punchies. To accomplish this, MICROS uses the Oxford Dictionary API to check whether the punchies are mass or proper nouns. If they are not, they are pluralized with pattern.en. Once that is complete, the story is output as the final artifact.

Results & Analysis

Six-word stories, like all art, can only be judged subjectively. Each reader will come to their own conclusion about the quality of a given story, and that estimation may change over time. Thus, our discussion of our system's results will naturally be biased based on our own tastes. Such bias cannot be removed, so instead we will briefly explain what we think makes a good six-word story.

Our first criteria is *coherence*—does the story make logical sense? Next, we consider *impact*—does the story elicit an emotional response? Finally, we consider a story's *subtlety*. The best six-word stories tell their stories without explicitly stating the story's topic, mood, or even its central action. The words in a subtle six-word story all semantically "point" to a story but either do not tell it explicitly or tell only a portion of it. This allows the reader's mind to fill in the gaps, resulting in a deeper and more interesting story.

With these criteria in mind, we turn to an analysis of our system's results and performance. All examples given in the following subsections were created by MICROS. See the Appendix for more results.

[4]https://developer.oxforddictionaries.com/

Successes

Our system generally succeeds at coherence; the majority of its artifacts fulfill this criteria and evoke at least a vague narrative. The examples "Schools. Institutes. Honors. A bachelor graduates." and "Redemption. Crimes. Chaos. A policeman escapes." typify the coherence of our system's artifacts; most of the words are at least loosely associated and paint a hazy picture in the mind of the reader. A story's skip-thought score tends to increase as its coherence improves. As a result, the evaluator favors and selects coherent stories.

Our system's artifacts only rarely have an impact on the reader. In the majority of cases, their coherence does not serve to tell an interesting story. Occasionally, an artifact's elements will combine to portray a fairly interesting narrative, but even those stories are still vague or slightly nonsensical. "Injustice. Ambitions. Minnesota. A farmer emigrates." and "Cowardice. Injustice. Motive. A hero stands." tell slightly interesting stories, and "Depravity. Misfortune. Hysteria. A clown laughs." succeeds in eliciting fear or dark humor in the reader. MICROS' stories are impactful when they are highly coherent and, by chance, the generated primitives are emotionally charged words.

Shortcomings

Although the majority of MICROS' artifacts are coherent, some are not due to the independence assumptions made by the format (factorization). Although this can yield nouns and verbs that don't make sense together, it occasionally results in comical or punny combinations such as "A wizard potters." or "A mechanic brakes". However, these punny stories often lack coherence because the generator can't choose punchies to match untraditional subject/verb pairs, such as "Immortality. Berserkers. Adventures. A wizard potters."

Our reliance on WordNet to identify verbs also causes problems because it contains every conceivable sense of a word, even archaic or obscure ones. For example, one of its synsets for the word "harlequin" lists it as a verb (which is defined as "[to] variegate with spots or marks"). Thus, if "harlequin" was returned from a relation, WordNet would identify it as a verb, even though using it as one in a story would likely confuse the reader.

WordNet is fast but clearly not the best way to determine the parts of speech in a phrase. However, even if a sophisticated parser—such as the Stanford Parser (De Marneffe, MacCartney, and Manning 2006)—was used to identify the relationship's parts of speech more accurately, the words' meanings would still be unknown. The example "Seclusion. Pregnancy. Love. A baby sleeps." demonstrates this. Even though the three punchies are all individually related to either "baby" or "sleep", the story makes no sense because the system has no way of knowing how they are related or how they should be used.

ConceptNet fails to provide such deep semantic relationships because its relations for each word are not separated by sense. A richer semantic database could represent relationships between specific word senses, instead of conflating all those senses into single terms. This would allow deeper understanding of the relationships between words while retaining ConceptNet's easy-to-search structure of relations.

The most difficult criteria for a six-word story to fulfill is subtlety. Even human writers struggle to write good stories that are just subtle enough to be interesting without being vague or illogical. Our system cannot compete at this level. Its most impactful stories still consist of words that are all directly related to the noun or verb.

MICROS' method of constructing primitives precludes it from achieving subtlety; each primitive is generated by selecting words that are directly related to parent words. If instead primitives were generated such that they all related to a separate concept that was not itself a primitive, the resulting story could perhaps approach that latent concept subtly.

Finally, MICROS makes occasional pluralization or conjugation mistakes such as in the story "Destructivenesses. Aristotles. Questions. A philosopher thinks." These mistakes are not unexpected and are due to the limitations of the pattern.en Python library. Although fast, pattern.en is limited to pattern matching and does not explicitly use the rules of English grammar. However, it is a good example of how such rules can be simplified to work in most cases. "Good enough" solutions like pattern.en are useful to creative computer systems where generation speed is often more important than correctness.

Community Evaluation

Evaluation by a community is the ultimate metric of artifacts' value, and we are pleased to report that MICROS performed better than some human writers in a real world environment. In order to test the results of our system in the wild, we submitted one MICROS-generated story a day to the /r/sixwordstories community on Reddit for a week. Each submitted story was freshly generated by MICROS on the day of submission and posted without curation.

Reddit is a social media platform that allows users to "upvote" content they like and "downvote" content they don't. The /r/sixwordstories subreddit has over 29,000 subscribers and one-to-two dozen submissions daily. Importantly, the stories posted on the subreddit are written by average users, who may be amateur writers at best. The typical best post on any given day will have 40–100 points and posts with less than 6 points are common. (A post's points are basically its upvotes minus its downvotes, to a minimum of 0.)

Of the seven stories we posted, two have 0 points, three received no votes, and two have positive scores: 5 and 7, respectively. Although those numbers may seem low, they do outscore other posts (presumably) written by humans. The story "Companionship. Youths. Fulfillment. The teacher cares." scored 5 points, which was higher than 5 of the 10 other stories posted that day. "Poverty. Retribution. Heroes. A villain acts." scored 7 points, outscoring 5 of the 12 other posts that day. These results are encouraging and show that MICROS can compete in the six-word story community, at least among amateur writers.

Conclusion & Future Work

MICROS represents an initial foray into the creation of microfiction using a novel HBPL-based approach. Although its results are often bland and never rise to the level of truly great writing, MICROS' approach to creative computation could serve as both an example and jumping off point for future research in computational creativity.

Our HBPL-based approach to formally defining the format of a creative artifact provides a convenient way to describe stories and poetry. It is more descriptive, for example, to define MICROS' six-word story format as being composed of three punchy primitives followed by an article primitive, subject primitive, and verb primitive than to simply list the parts of speech.

This approach of intentionally choosing factorizations of $p(S)$ to give a desired structure to and relationship between the different parts of a creative artifact could easily be applied in other story or poetry generation contexts to provide the system with information about how each word or phrase of the piece should relate to the others.

Conversely, MICROS could incorporate a module that programatically extracts factorizations of $p(S)$ from human-written text. A system that can identify primitives and learn the semantic relationships between them would allow a creative system to generate more varied and novel artifacts while retaining the semantic richness of the underlying structure. For example, MICROS could incorporate elements of the system described by (Toivanen et al. 2012) to operate on a corpus of existing stories, analyze the semantic relationships between the words that comprise them, and sample new stories from the learned format instead of replacing words into stories directly. This would allow the replacement words to not only relate to a topic but also to the other words in the story as dictated by the format.

More sophisticated and nuanced primitives, whether designed by researchers or extracted from existing text, could allow MICROS to generate six-word stories with high emotional impact. The words in the famous "baby shoes" story all relate to one another in a deep semantic way, and encapsulating those semantics into a factorization of $p(S)$ would enable the system to generate equally compelling stories.

Although MICROS currently only operates in the domain of writing, our approach is powerful because it is domain agnostic. The input E in $p(S|E)$ does not necessarily need to match the domain of S. Similar to how a human mind can be inspired to write a poem by a beautiful view, future work could allow MICROS to create six-word stories by sampling from a distribution conditioned on music or images. Designing factors of $p(S|E)$ that connect disparate domains could be an interesting avenue for future research.

By building systems that harness the rich semantic connections between inspirational and artistic domains as well as the equally-rich connections between the individual elements that comprise a creative artifact, we can further approximate human creativity. The MICROS system we have presented in this paper represents the first steps toward that goal in the domain of six-word stories, and the descriptive formalism we have adopted provides guidance for the designers of future creative systems by framing computational creativity in a standard formal structure.

Appendix

This appendix contains various six-word story artifacts created with MICROS. These stories were generated sequentially and are presented without curation.

Mirages. Injustice. Adventures. A ninja revenges.
Immortality. Berserkers. Adventures. A wizard potters.
Devotion. Honors. Alphas. An undergraduate majors.
Vocations. Thoughts. Schools. A philosopher teaches.
Vengeance. Hordes. Injustice. A soldier battles.
Necessities. Wellbeings. Achievements. A student begins.
Jealousy. Children. Immortality. A baby rattles.
Parties. Retribution. Parliaments. A manager resigns.
Injustice. Foes. Despair. A farmer rises.
Humanity. Devotion. Retribution. A hero stands.
Guardians. Followups. Indifference. A reporter replies.
Injustice. Acting. Regimes. A woman falls.
Devotion. Charlottes. Seconds. A queen plays.
Prisons. Courts. Justices. The lawyer comments.
Bloodlust. Diseases. Fulfillments. A surgeon parts.
Forgiveness. Weekends. Retaliation. A reporter replies.
Terrors. Actresses. Love. An actor shows.
Journeys. Terrors. Torment. A horse travels.
Personalities. Meltdowns. Desires. A plumber leaks.
Cavalries. Hardship. Regiments. A soldier drives.
Loneliness. Troupes. Champions. A dancer wells.
Families. Companionship. Fasts. A dog eats.
Terrors. Aces. Adventures. A monster cards.
Families. Dyings. Reigns. The king peoples.
Consciousnesses. Reigns. Sweden. A king groups.
Tournaments. Substitutes. Feuds. A wrestler matches.
Parties. Votes. Mates. A senator resigns.
Failure. Motives. Generations. A computer powers.
Talents. Partners. Concerns. A celebrity letters.
Destinies. Afterlives. Hysteria. A monster appears.
Loneliness. Affection. Misfortune. A student lunches.
Loyalty. Demoralizations. Souls. A band disbands.
Epics. Folks. Impunity. A poet rhymes.
Affection. Grandsons. Perfection. A politician plays.
Soccer. Appearances. Feuds. A wrestler competes.
Killings. Duties. Crimes. A policeman soldiers.
Elation. Wingers. Detriment. A writer rights.
Companionship. Injustice. Chases. A woman hunts.
Sights. Evil. Samurais. The ninja eyes.
Humanity. Redemption. Depredations. A hunter preys.
Rebirth. Loneliness. Adventures. A hobo bums.
Abhorrence. Wraiths. Revenge. A pirate bilges.
Academies. Graduation. Guys. A professor smarts.
Humanity. Causes. Necessities. A hero stands.
Contentment. Empresses. Sorrow. A queen consorts.
Weights. Collisions. Failure. A mechanic brakes.
Riders. Mankind. Desires. A horse races.
Hinds. Desires. Destinies. A hunter tails.
Adventures. Wells. Mafias. A detective gangs.
Humanity. Devotion. Templars. A knight duels.
Companionship. Bravery. Strikes. A coach plays.
Poverty. Everlastings. Retribution. A woman acts.
Fates. Tragedies. Adventures. A ninja revenges.
Soccers. Leagues. Childishnesses. A wrestler teams.

References

Binsted, K., and Ritchie, G. 1994. A symbolic description of punning riddles and its computer implementation. *arXiv preprint cmp-lg/9406021*.

Bodily, P.; Bay, B.; and Ventura, D. 2017. Computational creativity via human-level concept learning. In *Proceedings of the 8th International Conference on Computational Creativity*, 57–64.

Colton, S.; Goodwin, J.; and Veale, T. 2012. Full-FACE Poetry Generation. In *Proceedings of the 3rd International Conference on Computational Creativity*, 95–102.

De Marneffe, M.-C.; MacCartney, B.; and Manning, C. D. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the International Conference on Language Resources and Evaluation*, volume 6, 449–454.

Kiros, R.; Zhu, Y.; Salakhutdinov, R. R.; Zemel, R.; Urtasun, R.; Torralba, A.; and Fidler, S. 2015. Skip-thought vectors. In *Advances in Neural Information Processing Systems*, 3294–3302.

Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science* 350(6266):1332–1338.

León, C., and Gervás, P. 2014. Creativity in story generation from the ground up: Non-deterministic simulation driven by narrative. In *Proceedings of the 5th International Conference on Computational Creativity*, 201–210.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv* abs/1301.3781.

Miller, G. A. 1995. WordNet: A lexical database for English. *Communications of the Association for Computing Machinery* 38(11):39–41.

Montfort, N.; Pérez y Pérez, R.; Harrell, D. F.; and Campana, A. 2013. Slant: A blackboard system to generate plot, figuration, and narrative discourse aspects of stories. In *Proceedings of the 4th International Conference on Computational Creativity*, 168–175.

Oliviero, S., and Carlo, S. 2003. Hahacronym: Humorous agents for humorous acronyms.

Pérez y Pérez, R., and Sharples, M. 2001. MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence* 13(2):119–139.

Riedl, M. O., and Young, R. M. 2006. Story planning as exploratory creativity: Techniques for expanding the narrative search space. *New Generation Computing* 24(3):303–323.

Speer, R., and Havasi, C. 2012. Representing general relational knowledge in ConceptNet 5. In *Proceedings of the International Conference on Language Resources and Evaluation*, 3679–3686.

Toivanen, J.; Toivonen, H.; Valitutti, A.; Gross, O.; et al. 2012. Corpus-based generation of content and form in poetry. In *Proceedings of the 3rd International Conference on Computational Creativity*, 175–179.