BRAINSUP: Brainstorming Support for Creative Sentence Generation

Gözde Özbal
FBK-irst
Trento, Italy
gozbalde@gmail.com

Daniele Pighin
Google Inc.
Zürich, Switzerland
daniele.pighin@gmail.com

Carlo Strapparava
FBK-irst
Trento, Italy
strappa@fbk.eu

Abstract

We present BRAINSUP, an extensible framework for the generation of creative sentences in which users are able to force several words to appear in the sentences and to control the generation process across several semantic dimensions, namely *emotions*, *colors*, *domain relatedness* and *phonetic properties*. We evaluate its performance on a creative sentence generation task, showing its capability of generating well-formed, catchy and effective sentences that have all the good qualities of slogans produced by human copywriters.

1 Introduction

A variety of real-world scenarios involve talented and knowledgable people in a time-consuming process to write creative, original sentences generated according to well-defined requisites. For instance, to advertise a new product it could be desirable to have its name appearing in a punchy sentence together with some keywords relevant for marketing, e.g. "fresh", or "thirst" for the advertisement of a drink. Besides, it could be interesting to characterize the sentence with respect to a specific color, like "blue" to convey the idea of freshness, or to a color more related to the brand of the company, e.g. "red" for a new *Ferrari*. Moreover, making the slogan evoke "joy" or "satisfaction" could make the advertisement even more catchy for customers. On the other hand, there are many examples of provocative slogans in which copywriters try to impress their readers by suscitating strong negative feelings, as in the case of antismoke campaigns (e.g., "there are cooler ways to die than smoking" or "cancer cures smoking"), or the famous beer motto "*Guinness* is not good for you". As another scenario, creative sentence generation is also a useful teaching device. For example, the *keyword* or *linkword* method used for second language learning links the translation of a foreign (*target*) word to one or more keywords in the native language which are phonologically or lexically similar to the target word (Sagarra and Alba, 2006). To illustrate, for teaching the Italian word "tenda", which means "curtain" in English, the learners are asked to imagine "rubbing a *tender* part of their leg with a *curtain*". These words should co-occur in the same sentence, but constructing such sentences by hand can be a difficult and very time-consuming process. Özbal and Strapparava (2011), who attempted to automate the process, conclude that the inability to retrieve from the web a good sentence for all cases is a major bottleneck.

Although state of the art computational models of creativity often produce remarkable results, e.g., Manurung et al. (2008), Greene et al. (2010), Guerini et al. (2011), Colton et al. (2012) just to name a few, to our best knowledge there is no attempt to develop an unified framework for the generation of creative sentences in which users can control all the variables involved in the creative process to achieve the desired effect.

In this paper, we advocate the use of syntactic information to generate creative utterances by describing a methodology that accounts for lexical and phonetic constraints and multiple semantic dimensions at the same time. We present BRAINSUP, an extensible framework for creative sentence generation in which users can control all the parameters of the creative process, thus generating sentences that can be used for practical applications. First, users can define a set of keywords which must appear in the final sentence. Second, they can slant the output towards a spe-

Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pages 1446–1455,
Sofia, Bulgaria, August 4-9 2013.

Domain	Keywords	BRAINSUP output examples
coffee	waking, cup	Between waking and doing there is a wondrous cup.
coke	drink, exhaustion	The physical exhaustion wants the dark drink.
health	day, juice, sunshine	With juice and cereal the normal day becomes a summer sunshine.
beauty	kiss, lips	Passionate kiss, perfect lips. – Lips and eyes want the kiss.
mascara	drama, lash	Lash your drama to the stage. – A mighty drama, a biting lash.
pickle	crunch, bite	Crunch your bite to the top. – Crunch of a savage byte. – A large byte may crunch a little attention.
soap	skin, love, touch	A touch of love is worth a fortune of skin. – The touch of froth is the skin of love. – A skin of water is worth a touch of love.

Table 1: A selection of sentences automatically generated by BRAINSUP for specific domains.

cific *emotion*, *color* or *domain*. At the same time, they can require a sentence to include desired phonetic properties, such as *rhymes*, *alliteration* or *plosives*. The combination of these features allows for the generation of potentially catchy and memorable sentences by establishing connections between linguistic, emotional (LaBar and Cabeza, 2006), echoic and visual (Borman et al., 2005) memory, as exemplified by the system outputs showcased in Table 1. Other creative dimensions can easily be plugged in, due to the inherently modular structure of the system.

BRAINSUP supports the creative process by greedily exploring a huge solution space to produce completely novel utterances responding to user requisites. It exploits syntactic constraints to dramatically cut the size of the search space, thus making it possible to focus on the creative aspects of sentence generation.

2 Related work

Research in creative language generation has bloomed in recent years. In this section, we provide a necessarily succint overview of a selection of the studies that most heavily inspired and influenced the development of BRAINSUP.

Humor generators are a notable class of systems exploring new venues in computational creativity (Binsted and Ritchie, 1997; McKay, 2002; Manurung et al., 2008). Valitutti et al. (2009) present an interactive system which generates humorous puns obtained through variation of familiar expressions with word substitution. The variation takes place considering the phonetic distance and semantic constraints such as semantic similarity, semantic domain opposition and affective polarity difference. Possibly closer to slogan generation, Guerini et al. (2011) slant existing textual expressions to obtain more positively or negatively valenced versions using WordNet (Miller, 1995) semantic relations and SentiWordNet (Esuli and Sebastiani, 2006) annotations. Stock and Strapparava (2006) generate acronyms based on lexical substitution via semantic field opposition, rhyme, rythm and semantic relations. The model is limited to the generation of noun phrases.

Poetry generation systems face similar challenges to BRAINSUP as they struggle to combine semantic, lexical and phonetic features in a unified framework. Greene et al. (2010) describe a model for poetry generation in which users can control meter and rhyme scheme. Generation is modeled as a cascade of weighted Finite State Transducers that only accept strings conforming to the desired rhyming scheme. Toivanen et al. (2012) attempt to generate novel poems by replacing words in existing poetry with morphologically compatible words that are semantically related to a target domain. Content control and the inclusion of phonetic features are left as future work and syntactic information is not taken into account. The Electronic Text Composition project[1] is a corpus based approach to poetry generation which recursively combines automatically generated linguistic constituents into grammatical sentences. Colton et al. (2012) propose another data-driven approach to poetry generation based on simile transformation. The mood and theme of the poems are influenced by daily news. Constraints about phonetic properties of the selected words or their frequencies can be enforced during retrieval. Unlike these examples, BRAINSUP makes heavy use of syntactic information to enforce well-formed sentences and to constraint the search for a solution, and provides an extensible framework in which various forms of linguistic creativity can easily be incorporated.

Several slogan generators are available on the web[2], but their capabilities are very limited as they can only replace single words or word sequences within existing slogan. This often results in syntactically incorrect outputs. Furthermore, they do not allow for other forms of user control.

[1] http://slought.org/content/11199

[2] E.g.: http://www.procato.com/slogan+generator, http://www.sloganizer.net/en/, http://www.sloganmania.com/index.htm.

3 Architecture of BRAINSUP

To effectively support the creative process with useful suggestions, we must be able to generate sentences conforming to the user needs. First of all, users can select the *target words* that need to appear in the sentence. In the context of second language learning, these might be the words that a learner must associate in order to expand her vocabulary. For slogan generation, the target words could be the key features of a product, or target-defining keywords that copywriters want to explicitly mention. On top of that, a user can characterize the generated sentences according to several dimensions, namely: 1) a specific semantic domain, e.g.: "sports" or "blankets"; 2) a specific emotion, e.g., "joy", "anger" or just "negative"; 3) a specific color, e.g., "red" or "blue"; 4) a combination of phonetic properties of the words that will appear in the sentence, i.e., rhymes, alliterations and plosives. More formally, the *user input* is a tuple: $U = \langle \mathbf{t}, \mathbf{d}, c, e, p, \mathbf{w} \rangle$, where $\mathbf{t}$ is the set of *target words*, $\mathbf{d}$ is a set of words defining the target *domain*, c and p are, respectively, the *color* and the *emotion* towards which the user wants to slant the sentence, p represents the desired phonetic features, and $\mathbf{w}$ is a set of *weights* that control the influence of each dimension on the generative process, as detailed in Section 3.3. For target and domain words, users can explicitly select one or more POSes to be considered, e.g., "drink/verb" or "drink/verb,noun".

The sentence generation process is based on morpho-syntactic *patterns* which we automatically discover from a corpus of dependency parsed sentences $\mathcal{P}$. These patterns represent very general skeletons of well-formed sentences that we employ to generate creative sentences by only focusing on the lexical aspects of the process. Candidate fillers for each empty position (*slot*) in the patterns are chosen according to the lexical and syntactic constraints enforced by the dependency relations in the patterns. These constraints are learned from relation-head-modifier co-occurrence counts estimated from a dependency treebank $\mathcal{L}$. A beam search in the space of all possible lexicalizations of a syntactic pattern promotes the words with the highest likelihood of satisfying the user specification.

Algorithm 1 provides a high-level description of the creative sentence generation process. Here, Θ is a set of meta-parameters that affect search complexity and running time of the algorithm, such as the minimum/maximum number of solutions to be generated, the maximum number of patterns to consider, or the maximum size of the generated sentences. $\mathrm{CompatiblePatterns}(\cdot)$ finds the most frequent syntactic patterns in $\mathcal{P}$ that are compatible with the user specification, as explained in Section 3.1; $\mathrm{FillInPattern}(\cdot)$ carries out the beam search, and returns the best solutions generated for each pattern p given U. The algorithm terminates when at least a minimum number of solutions have been generated, or when all the compatible patterns have been exhausted. Finally, only the best among the generated solutions are shown to the user. More details about the search in the solution space are provided in Section 3.2.

Algorithm 1 $\mathrm{SentenceGeneration}(U, \Theta, \mathcal{P}, \mathcal{L})$: U is the user specification, Θ is a set of meta-parameters; $\mathcal{P}$ and $\mathcal{L}$ are two dependency treebanks.

$\mathcal{O} \leftarrow \emptyset$
for all $p \in \mathrm{CompatiblePatterns}_{\Theta}(U, \mathcal{P})$ **do**
  while $\mathrm{NotEnoughSolutions}_{\Theta}(\mathcal{O})$ **do**
    $\mathcal{O} \leftarrow \mathcal{O} \cup \mathrm{FillInPattern}_{\Theta}(U, p, \mathcal{L})$
return $\mathrm{SelectBestSolutions}_{\Theta}(\mathcal{O})$

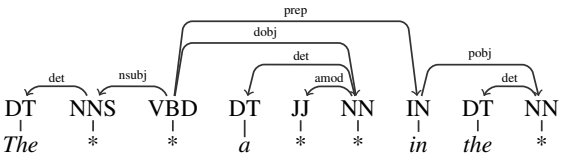

Figure 1: Example of a syntactic *pattern*. A "*" represents an empty *slot* to be filled with a *filler*.

3.1 Pattern selection

We generate creative sentences starting from morpho-syntactic *patterns* which have been automatically learned from a large corpus $\mathcal{P}$. The choice of the corpus from which the patterns are extracted constitutes the first element of the creative sentence generation process, as different choices will generate sentences with different styles. For example, a corpus of slogans or punchlines can result in short, catchy and memorable sentences, whereas a corpus of simplified English would be a better choice to learn a second language or to address low reading level audiences.

A pattern is the syntactic skeleton of a class of sentences observed in $\mathcal{P}$. Within a pattern, a second element of creativity involves the selection of original combinations of words (*fillers*) that do not violate the grammaticality of the sentence. The patterns that we employ are automatic dependency trees from which all content-words have been removed, as exemplified in Figure 1. After selecting the target corpus, we parse all the sentences with the Stanford Parser (Klein and Man-

ning, 2003) and produce the patterns by stripping away all content words from the parses. Then, for each pattern we count how many times it has been observed in the corpus. Additionally, we keep track of what kind of empty *slots*, i.e., empty positions, are available in each pattern. For example, the pattern in Figure 1 can accommodate up to two singular nouns (NN), one plural noun (NNS), one adjective (JJ) and one verb in the past tense (VBD). This information is needed to select the patterns which are *compatible* with the target words $\mathbf{t}$ in the user specification U. For example, this pattern is not compatible with $\mathbf{t} = [\text{heading/VBG}, \text{edge/NN}]$ as the pattern does not have an empty slot for a gerundive verb, while it satisfies $\mathbf{t} = [\text{heading/NN}, \text{edge/NN}]$ as it can accommodate the two singular nouns. While retrieving patterns, we also need to enforce that a pattern be not completely filled just by adding the target words $\mathbf{t}$, as under these conditions there would be no room to achieve any kind of creative effect. Therefore, we also require that the patterns retrieved by $\text{CompatiblePatterns}(\cdot)$ have more empty slots than the size of $\mathbf{t}$. The minimum and maximum number of excess slots in the pattern are two other meta-parameters controlled by Θ. $\text{CompatiblePatterns}(\cdot)$ returns compatible patterns ordered by their frequency, i.e. when generating solutions the first patterns that are explored are the most frequently observed ones. In this way, we achieve the following two objectives: 1) we compensate for the unavoidable errors introduced by the automatic parser, as frequently observed parses are less likely to be the result of an erroneous interpretation of a sentence; and 2) we generate sentences that are most likely to be catchy and memorable, being based on syntactic constructs that are used more frequently. To avoid always selecting the same patterns for the same kinds of inputs, we add a small random component (also controlled by Θ) to the pattern sorting algorithm, thus allowing for sentences to be generated also from non-top ranked patterns.

3.2 Searching the solution space

With the compatible patterns selected, we can initiate a beam search in the space of all possible lexicalizations of the patterns, i.e., the space of all sentences that can be generated by respecting the syntactic constraints encoded by each pattern. The process starts with a syntactic pattern p containing only stop words, syntactic relations and morphologic constraints (i.e., part-of-speech tags) for the empty slots. The search advances towards a complete solution by selecting an empty slot to fill and trying to place candidate fillers in the selected position. Each *partially lexicalized solution* is scored by a battery of scoring functions that compete to generate creative sentences respecting the user specification U, as explained in Section 3.3. The most promising solutions are extended by filling another slot, until completely lexicalized sentences, i.e., sentences without empty slots, are generated.

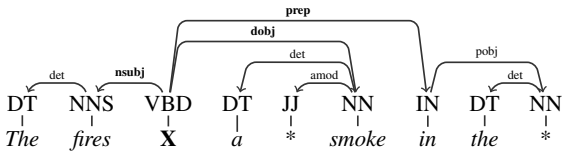

Figure 2: A partially lexicalized sentence with a highlighted empty slot marked with *X*. The relevant dependencies to fill in the slot are shown in boldface.

To limit the number of words that can occupy a given position in a sentence, we define a set of operators that return a list of candidate fillers for a slot solely based on syntactic clues. To achieve that, we analyze a large corpus of parsed sentences $\mathcal{L}$[3] and store counts of observed *head-relation-modifier* ($\langle h, r, m \rangle$) dependency relations. Let $\tau_{\text{r}}(h)$ be an operator that, when applied to a head word h in a relation r, returns the set of words in $\mathcal{L}$ which have been observed as modifiers for h in r with a specific POS. To simplify the notation, we assume that the relation r also carries along the POS of the head and modifier slots. As an example, with respect to the tree depicted in Figure 2, $\tau_{\text{amod}}(\text{smoke})$ would return all the words with POS equal to "JJ" that have been observed as adjective modifiers for the singular noun "smoke". We will refer to $\tau_{\text{r}}(\cdot)$ as the *dependency operator* for r. For every $\tau_{\text{r}}(\cdot)$, we also define an *inverse dependency operator* $\tau_{\text{r}}^{-1}(\cdot)$, which returns the list of the possible heads in r when applied to a modifier word m. For instance, with respect to Figure 2, $\tau_{\text{nsubj}}^{-1}(\text{fires})$ would return the set of verbs in the past tense of which "fires" as a plural noun can be a subject.

While filling in a given slot X, the dependency operators can be combined to obtain a list of words which are likely to occupy that position given the syntactic constraints induced by the structure of the pattern. Let $W = \cup_i \{w_i\}$ be the set of words which are directly connected to the empty slot by

[3] Distinct from the corpus used for pattern selection, $\mathcal{P}$.

a dependency relation. Each word w_i implies a constraint that candidate fillers for X must satisfy. If w_i is the head of X, then a direct operator is used to retrieve a list of fillers that satisfy the i^{th} constraint. Conversely, if w_i is a modifier of X, an inverse operator is employed.

As an example, let us consider the partially completed sentence shown in Figure 2 having an empty slot marked with X. Here, the word "smoke" is a modifier for X, to which it is connected by a $dobj$ relation. Therefore, we can exploit $\tau^{-1}_{\text{dobj}}(\text{smoke})$ to obtain a ranked list of words that can occupy X according to this constraint. Similarly, the $\tau^{-1}_{\text{nsubj}}(\text{fires})$ operator can be used to retrieve a list of verbs in the past tense that accept "fires" as $nsubj$ modifier. Finally $\tau^{-1}_{\text{prep}}(\text{in})$ can further restrict our options to verbs that accepts complements introduced by the preposition "in". For example, the words "generated", "produced", "caused" or "formed" would be good candidates to fill in the slot considering all the previous constraints. More formally, we can define the set of candidate fillers for a slot X, $\mathcal{C}_X$, as: $\mathcal{C}_X = \tau^{-1}_{\text{r}_{\text{h}_X,\text{X}}}(h_X) \cap (\bigcap_{w_i | w_i \in M_X} \tau_{\text{r}_{\text{w}_i,\text{X}}}(w_i))$, where $r_{w_i,X}$ is the type of relation between w_i and X, M_X is the set of modifiers of X and h_X is the syntactic head of X.[4]

Concerning the order in which slots are filled, we start from those that have the highest number of dependencies (both head or modifiers) that have been already instantiated in the sentence, i.e., we start from the slots that are connected to the highest number of non-empty slots. In doing so we maximize the constraints that we can rely on when inserting a new word, and eventually generate more reliable outputs.

3.3 Filler selection and solution scoring

We have devised a set of *feature functions* that account for different aspects of the creative sentence generation process. By changing the weight $\mathbf{w}$ of the feature functions in U, users can control the extent to which each creativity component will affect the sentence generation process, and tune the output of the system to better match their needs. As explained in the remainder of this section, feature functions are responsible for ranking the candidate slot fillers to be used during sentence generation and for selecting the best solutions to be shown to the users.

[4] An empty slot does not generate constraints for X. In addition, there might be cases in which it is not possible to find a filler that satisfies all the constraints at the same time. In such cases, all the fillers that satisfy the maximum number of constraints are considered.

Algorithm 2 $\text{RankCandidates}(U, \mathbf{f}, c_1, c_2, s, X)$: c_1 and c_2 are two candidate fillers for the slot X in the sentence $\mathbf{s} = [s_0, \ldots s_n]$; $\mathbf{f}$ is the set of feature functions; U is the user specification.

```
s^{c1} ← s, s^{c2} ← s, s^{c1}[X] ← c1, s^{c2}[X] ← c2
for all f ∈ SortFeatureFunctions_Θ(U, f) do
  if f(s^{c1}, U) > f(s^{c2}, U) then return c1 ≻ c2
  else if f(s^{c1}, U) < f(s^{c2}, U) then
    return c1 ≺ c2
return c1 ≡ c2
```

Algorithm 2 details the process of ranking candidate fillers. To compare two candidates c_1 and c_2 for the slot X in the sentence s, we first generate two sentences $\mathbf{s}^{c_1}$ and $\mathbf{s}^{c_2}$ in which the empty slot X is occupied by c_1 and c_2, respectively. Then, we sort the feature functions based on their weights in descending order, and in turn we apply them to score the two sentences. As soon as we find a scorer for which one sentence is better than the other, we can take a decision about the ranking of the fillers. This approach makes it possible to establish a strict order of precedence among feature functions and to select fillers that have a highest chance of maximizing the user satisfaction.

Concerning the scoring of partial solutions and complete sentences, we adopt a simple linear combination of scoring functions. Let $\mathbf{s}$ be a (partial) sentence, $\mathbf{f} = [\text{f}_0, \ldots, \text{f}_k]$ be the vector of scoring functions and $\mathbf{w} = [w_0, \ldots, w_k]$ the associated vector of weights in U. The overall score of $\mathbf{s}$ is calculated as $\text{score}(\mathbf{s}, U) = \sum_{i=0}^{k} w_i \text{f}_i(\mathbf{s}, U)$. Solutions that do not contain all the required target words are discarded and not shown to the user.

Currently, the model employs the following 12 feature functions:

Chromatic and emotional connotation. The chromatic connotation of a sentence $\mathbf{s} = [s_0, \ldots, s_n]$ is computed as $\text{f}(\mathbf{s}, U) = \sum_{s_i}(\text{sim}(s_i, c) - \sum_{c_j \neq c} \text{sim}(s_i, c_j))$, where c is the user selected target color and $\text{sim}(s_i, c_j)$ is the degree of association between the word s_i and the color c_j as calculated by Mohammad (2011). All the words in the sentence which have an association with the target color c give a positive contribution, while those that are associated with a color $c_i \neq c$ contribute negatively. Emotional connotation works exactly in the same way, but in this case word-emotion associations are taken from (Mohammad and Turney, 2010).

Domain relatedness. This feature function uses an LSA (Deerwester et al., 1990) vector space

model to measure the similarity between the words in the sentence and the target domain $\mathbf{d}$ specified by the user. It is calculated as: $\mathrm{f}(\mathbf{s}, U) = \frac{\sum_{d_i} \mathrm{v}(d_i) \cdot \sum_{s_i} \mathrm{v}(s_i)}{\| \sum_{d_i} \mathrm{v}(d_i) \| \cdot \| \sum_{s_i} \mathrm{v}(s_i) \|}$ where $\mathrm{v}(\cdot)$ returns the representation of a word in the vector space.

Semantic cohesion. This feature behaves exactly like domain relatedness, with the only difference that it measures the similarity between the words in the sentence and the target words $\mathbf{t}$.

Target-words scorer. This feature function simply counts what fraction of the target words $\mathbf{t}$ is present in a partial solution: $\mathrm{f}(\mathbf{s}, U) = (\sum_{s_i | s_i \in \mathbf{t}} 1)/|\mathbf{t}|$. The target word scorer takes care of enforcing the presence of the target words in the sentences. Letting beam search find the best placement for the target words comes at no extra cost and results in a simple and elegant model.

Phonetic features (plosives, alliteration and rhyme). All the phonetic features are based on the phonetic representation of English words of the Carnegie Mellon University pronouncing dictionary (Lenzo, 1998). The plosives feature is calculated as the ratio between the number of plosive sounds in a sentence and the overall number of phonemes. For the alliteration scorer, we store the phonetic representation of each word in $\mathbf{s}$ in a *trie* (i.e., *prefix tree*), and count how many times each node n_i of the trie (corresponding to a phoneme) is traversed. Let c_i be the value of the counts for n_i. The alliteration score is then calculated as $\mathrm{f}(\mathbf{s}, U) = (\sum_{i | c_i > 1} c_i) / \sum_i c_i$. More simply put, we count how many of the phonetic prefixes of the words in the sentence are repeated, and then we normalize this value by the total number of phonemes in $\mathbf{s}$. The rhyme feature works exactly in the same way, with the only difference that we invert the phonetic representation of each word before adding it to the TRIE. Thus, we give higher scores to sentences in which several words share the same phonetic ending.

Variety scorer. This feature function promotes sentences that contain as many different words as possible. It is calculated as the number of distinct words in the sentence over the size of the sentence.

Unusual-words scorer. To increase the ability of the model to generate sentences containing non-trivial word associations, we may want to prefer solutions in which relatively uncommon words are employed. Inversely, we may want to lower lexical complexity to generate sentences more appropriate for certain education or reading levels. We define c_i as the number of times each word $s_i \in \mathbf{s}$ is observed in a corpus $\mathcal{V}$. Accordingly, the value of this feature is calculated as: $\mathrm{f}(\mathbf{s}, U) = (1/|\mathbf{s}|)(\sum_{s_i} 1/c_i)$.

N-gram likelihood. This is simply the likelihood of a sentence estimated by an n-gram language model, to enforce the generation of well-formed word sequences. When a solution is not complete, in the computation we include only the sequences of contiguous words (i.e., not interrupted by empty slots) having length greater than or equal to the order of the n-gram model.

Dependency likelihood. This feature is related to the dependency operators introduced in Section 3.2 and it enforces sentences in which dependency chains are well formed. We estimate the probability of a modifier word m and its head h to be in the relation r as $p_r(h, m) = \mathrm{c}_r(h, m)/(\sum_{h_i} \sum_{m_i} \mathrm{c}_r(h_i, m_i))$, where $c_r(\cdot)$ is the number of times that m depends on h in the dependency treebank $\mathcal{L}$ and h_i, m_i are all the head/modifier pairs observed in $\mathcal{L}$. The dependency-likelihood of a sentence $\mathbf{s}$ can then be calculated as $\mathrm{f}(\mathbf{s}, U) = \exp(\sum_{\langle h,m,r \rangle \in r(\mathbf{s})} \log p_r(h, m))$, $\mathrm{r}(\mathbf{s})$ being the set of dependency relations in $\mathbf{s}$.

4 Evaluation

We evaluated our model on a creative sentence generation task. The objective of the evaluation is twofold: we wanted to demonstrate 1) the effectiveness of our approach for creative sentence generation, in general, and 2) the potential of BRAINSUP to support the brainstorming process behind slogan generation. To this end, the annotation template included one question asking the annotators to rate the quality of the generated sentences as slogans.

Five experienced annotators were asked to rate 432 creative sentences according to the following criteria, namely: 1) *Catchiness:* is the sentence attractive, catchy or memorable? [Yes/No] 2) *Humor:* is the sentence witty or humorous? [Yes/No]; 3) *Relatedness:* is the sentence semantically related to the target domain? [Yes/No]; 4) *Correctness*: is the sentence grammatically correct? [Ungrammatical/Slightly disfluent/Fluent]; 5) *Success*: could the sentence be a good slogan for the target domain? [As it is/With minor editing/No]. In these last two cases, the annotators

were instructed to select the middle option only in cases where the gap with a correct/successful sentence could be filled just by performing minor editing. The annotation form had no default values, and the annotators did not know how the evaluated sentences were generated, or whether they were the outcome of one or more systems.

We started by collecting slogans from an online repository of slogans[5]. Then, we randomly selected a subset of these slogans and for each of them we generated an input specification U for the system. We used the commercial domain of the advertised product as the target domain $\mathbf{d}$. Two or three content words appearing in each slogan were randomly selected as the target words $\mathbf{t}$. We did so to simulate the brainstorming phase behind the slogan generation process, where copywriters start with a set of relevant keywords to come up with a catchy slogan. In all cases, we set the target emotion to "positive" as we could not establish a generally valid criteria to associate a specific emotion to a product. Concerning chromatic slanting, for target domains having a strong chromatic correlation we allowed the system to slant the generated sentences accordingly. In the other cases, a random color association was selected. In this manner, we produced 10 tuples $\langle \mathbf{t}, \mathbf{d}, c, e, p \rangle$. Then, from each tuple we produced 5 complete user specifications by enabling or disabling different feature function combinations[6]. The four combinations of features are: **base**: Target-word scorer + N-gram likelihood + Dependency likelihood + Variety scorer + Unusual-words scorer + Semantic cohesion; **base+D**: all the scorers in *base* + Domain relatedness; **base+D+C**: all the scorers in *base+D* + Chromatic connotation; **base+D+E**: all the scorers in *base+D* + Emotional connotation; **base+D+P**: all the scorers in *base+D* + Phonetic features. For each of the resulting 50 input configurations, we generated up to 10 creative sentences. As the system could not generate exactly 10 solutions in all the cases, we ended up with a set of 432 items to annotate. The weights of the feature functions were set heuristically, due to the lack of an annotated dataset suitable to learn an optimal weight configuration. We started by assigning the highest weight to the *Target Word* scorer (i.e., 1.0), followed by the *Variety* and *Unusual Word* scorers (0.99), the *Phonetic Features*, *Chromatic/Emotional Connotation* and *Semantic Cohesion* scorers (0.98) and finally the *Domain*, N-*gram* and *Dependency Likelihood* scorers (0.97). These settings allow us to enforce an order of precedence among the scorers during slot-filling, while giving them virtually equal relevance for solution ranking.

MC	Cat.	Hum.	Corr.	Rel.	Succ.	RND_2	RND_3
2	-	-	16.67	-	22.22	-	37.04
3	47.45	39.58	43.52	13.66	44.21	62.50	49.38
4	33.10	37.73	32.18	21.99	22.22	31.25	12.35
5	19.44	22.69	07.64	64.35	11.34	06.25	01.23

Table 2: Majority classes (%) for the five dimensions of the annotation.

As discussed in Section 3 we use two different treebanks to learn the syntactic patterns ($\mathcal{P}$) and the dependency operators ($\mathcal{L}$). For these experiments, patterns were learned from a corpus of 16,000 proverbs (Mihalcea and Strapparava, 2006), which offers a good selection of short sentences with a good potential to be used for slogan generation. This choice seemed to be a good compromise as, to our best knowledge, there is no published slogan dataset with an adequate size. Besides, using existing slogans might have legal implications that we might not be aware of. Dependency operators were learned by dependency parsing the British National Corpus[7]. To reduce the amount of noise introduced by the automatic parses, we only considered sentences having less than 20 words. Furthermore, we only considered sentences in which all the content words are listed in WordNet (Miller, 1995) with the observed part of speech.[8] The LSA space used for the semantic feature functions was also learned on BNC data, but in this case no filtering was applied.

4.1 Results

To measure the agreement among the annotators, similarly to Mohammad (2011) and Ozbal and Strapparava (2012) we calculated the majority class for each dimension of the annotation task. A

[5] http://www.tvacres.com/advertising_slogans.htm

[6] An alternative strategy to keep the annotation effort under control would have been to generate fewer sentences from a larger number of inputs. We adopted the former setting since we regarded it as more similar to a brainstorming session, where the system proposes different alternatives to inspire human operators. Forcing BRAINSUP to only output one or two sentences would have limited its ability to explore and suggest potentially valuable outputs.

[7] http://www.natcorp.ox.ac.uk/

[8] Since the CMU pronouncing dictionary used by the phonetic scorers is based on the American pronunciation of words, we actually pre-processed the whole BNC by replacing all British-English words with their American-English counterparts. To this end, we used the mapping available at http://wordlist.sourceforge.net/.

	Cat.	Rel.	Hum.	Succ.	Corr.
Yes	67.59	93.98	12.73	32.41	64.35
Partly	-	-	-	23.15	31.71
No	32.41	06.02	87.27	44.44	03.94

Table 3: Majority decisions (%) for each annotation dimension.

majority class greater than or equal to 3 means that the absolute majority of the 5 annotators agreed on the same decision[9]. Table 2 shows the observed agreement for each dimension. The column labeled RND_2 (RND_3) shows the random agreement for a given number of annotators and a binary (ternary) decision. For example, all five annotators (MC=5) agreed on the annotation of the catchiness of the slogans in 19.44% of the cases. The random chance of agreement for 5 annotators on the binary decision problem is 6.25%. The figures for MC $\geq$ 4 are generally high, confirming a good agreement among the annotators. The agreement on the relatedness of the slogans is especially high, with all 5 annotators taking the same decision in almost two cases out of three, i.e., 64.35%.

Table 3 lists the distribution of answers for each dimension in the cases where a decision can be taken by majority vote. The generated slogans are found to be catchy in more than 2/3 of the cases, (i.e., 67.59%), completely successful in 1/3 of the cases (32.41%) and completely correct in 2/3 of the cases (64.35%). These figures demonstrate that BRAINSUP is very effective in generating grammatical utterances that have all the appealing properties of a successful slogan. As for humor, the sentences are found to have this property in only 12.73% of cases. Even though the figure is not very high, we should also consider that BRAINSUP is not explicitly trying to generate amusing utterances. Concerning success, we should point out that in 23.15% of the cases the annotators have found that the generated slogans have the potential to be turned into successful ones only with minor editing. This is a very important piece of result, as it corroborates our claim that BRAINSUP can indeed be a valuable tool for copywriting, even when it does not manage to output a perfectly good sentence. Similar conclusions can be drawn concerning the correctness of the output, as in almost one third of the cases the slogans are only affected by minor disfluencies.

The relatedness figure is especially high, as in almost 94% of the cases the majority of annotators found the slogans to be pertinent to the target domain. This result is not surprising, as all the slogans are generated by considering keywords that already exist in real slogans for the same domain. Anyhow, this is exactly the kind of setting in which we expect BRAINSUP to be employed, i.e., to support creative sentence generation starting from a good set of relevant keywords. Nonetheless, it is very encouraging to observe that the generation process does not deteriorate the positive impact of the input keywords.

We would also like to mention that in 63 cases (14.58%) the majority of the annotators have labeled the slogans favorably across all 5 dimensions. The examples listed in Table 1 are selected from this set. It is interesting to observe how the word associations established by BRAINSUP can result in pertinent yet unintentional rhetorical devices such as metaphors ("a summer sunshine"), puns ("lash your drama") and personifications ("lips and eyes want"). Some examples show the effect of the phonetic features, e.g. plosives in "*p*assiona*t*e *k*iss, *p*erfe*ct* li*p*s", alliteration in "*th*e *d*ark *d*rink" and rhyming in "lip*s* and eye*s* want the ki*ss*". In some cases, the output of BRAINSUP seems to be governed by mysterious philosophical reasoning, as in the delicate examples generated for "soap".

For comparison, Table 4 lists a selection of the examples that have been labeled as unsuccessful by the majority of raters. In some cases, BRAINSUP is improperly selecting attributes that highlight undesirable properties in the target domain, e.g., "A pleasant tasting, a *heady* wine". To avoid similar errors, it would be necessary to reason about the valence of an attribute for a specific domain. In other cases, the N-gram and the Dependency Likelihood features may introduce phrases which are very cohesive but unrelated to the rest of the sentence, e.g., "Unscrupulous doctors smoke *armored units*". Many of these errors could be solved by increasing the weight of the Semantic Cohesion and Domain Relatedness scorers. In other cases, such as "A sixth calorie may taste *an own good*" or "*A same sunshine* is fewer than a juice of day", more sophisticated reasoning about syntactic and semantic relations in the output might be necessary in order to enforce the generation of sound and grammatical sentences.

We could not find a significant correlation be-

[9]For the binary decisions (i.e., catchiness, relatedness and humor), at least 3 annotators out of 5 must necessarily agree on the same option.

Domain	Keywords	BRAINSUP output examples
pleasure	wine, tasting	A pleasant tasting, a heady wine. – A fruity tasting may drink a sparkling wine.
healthy	day, juice, sunshine	Drink juice of your sunshine, and your weight will choose day of you. – A same sunshine is fewer than a juice of day.
cigarette	doctors, smoke	Unscrupulous doctors smoke armored units. – Doctors smoke no arrow.
mascara	drama, lash	The such drama is the lash.
soap	skin, love, touch	The touch of skin is the love of cacophony. – You love an own skin for a first touch.
coke	calorie, taste, good	A sixth calorie may taste an own good.
coffee	waking, cup	You cannot cup hands without waking some fats.

Table 4: Unsuccessful BRAINSUP outputs.

tween the input variables (e.g., presence or absence of phonetic features or chromatic slanting) and the outcome of the annotation, i.e. the system by and large produces correct, catchy, related and (at least potentially) successful outputs regardless of the specific input configurations. In this respect, it should be noted that we did not carry out any kind of optimization of the feature weights, which might be needed to obtain more heavily characterized sentences. Furthermore, to better appreciate the contribution of the individual features, comparative experiments in which the users evaluate the system before and after triggering a feature function might be necessary. Concerning the correlation among output dimensions, we only observed relatively high Spearman correlation between correctness and relatedness (0.65), and catchiness and success (0.68).

5 Conclusion

We have presented BRAINSUP, a novel system for creative sentence generation that allows users to control many aspects of the creativity process, from the presence of specific target words in the output, to the selection of a target domain, and to the injection of phonetic and semantic properties in the generated sentences. BRAINSUP makes heavy use of dependency parsed data and statistics collected from dependency treebanks to ensure the grammaticality of the generated sentences, and to trim the search space while seeking the sentences that maximize the user satisfaction.

The system has been designed as a supporting tool for a variety of real-world applications, from advertisement to entertainment and education, where at the very least it can be a valuable support for time-consuming and knowledge-intensive sentence generation needs. To demonstrate this point, we carried out an evaluation on a creative sentence generation benchmark showing that BRAINSUP can effectively produce catchy, memorable and successful sentences that have the potential to inspire the work of copywriters.

To our best knowledge, this is the first systematic attempt to build an extensible framework that allows for multi-dimensional creativity while at the same time relying on syntactic constraints to enforce grammaticality. In this regard, our approach is dual with respect to previous work based on lexical substitution, which suffers from limited expressivity and creativity latitude. In addition, by acquiring the lexicon and the sentence structure from two distinct corpora, we can guarantee that the sentences that we generate have never been observed. We believe that our contribution constitutes a valid starting point for other researchers to deal with unexplored dimensions of creativity.

As future work, we plan to use machine learning techniques to estimate optimal weights for the feature functions in different use cases. We would also like to consider syntactic clues while reasoning about semantic properties of the sentence, e.g., color and emotion associations, instead on relying solely on lexical semantics. Concerning the extension of the capabilities of BRAINSUP, we want to include common-sense knowledge and reasoning to profit from more sophisticated semantic relations and to inject humor on demand. Further tuning of BRAINSUP to build a dedicated system for slogan generation is also part of our future plans. After these improvements, we would like to conduct a more focused evaluation on slogan generation involving human copywriters and domain experts in an interactive setting.

We would like to conclude this paper with a pearl of BRAINSUP's wisdom:

It is wiser to believe in science
than in everlasting love.

Acknowledgments

Gözde Özbal and Carlo Strapparava were partially supported by the PerTe project (Trento RISE).

References

Kim Binsted and Graeme Ritchie. 1997. Computational rules for generating punning riddles. *Humor - International Journal of Humor Research*, 10(1):25–76, January.

Andy Borman, Rada Mihalcea, and Paul Tarau. 2005. Pic-net: Pictorial representations for illustrated semantic networks. In *Proceedings of the AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors*.

Simon Colton, Jacob Goodwin, and Tony Veale. 2012. Full-FACE Poetry Generation. In *Proceedings of the 3rd International Conference on Computational Creativity*, pages 95–102.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal Of The American Society for Information Science*, 41(6):391–407.

Andrea Esuli and Fabrizio Sebastiani. 2006. Sentiwordnet: A publicly available lexical resource for opinion mining. In *In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC'06)*, pages 417–422.

Erica Greene, Tugba Bodrumlu, and Kevin Knight. 2010. Automatic analysis of rhythmic poetry with applications to generation and translation. In *EMNLP*, pages 524–533.

Marco Guerini, Carlo Strapparava, and Oliviero Stock. 2011. Slanting existing text with valentino. In *Proceedings of the 16th international conference on Intelligent user interfaces*, IUI '11, pages 439–440, New York, NY, USA. ACM.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kevin S. LaBar and Roberto Cabeza. 2006. Cognitive neuroscience of emotional memory. *Nature reviews. Neuroscience*, 7(1):54–64, January.

Kevin Lenzo. 1998. The cmu pronouncing dictionary. http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

Ruli Manurung, Graeme Ritchie, Helen Pain, Annalu Waller, Dave O'Mara, and Rolf Black. 2008. The Construction of a Pun Generator for Language Skills Development. *Applied Artificial Intelligence*, 22(9):841–869, October.

J McKay. 2002. Generation of idiom-based witticisms to aid second language learning. In *Twente Workshop on Language Technology 20*, pages 70–74.

R. Mihalcea and C. Strapparava. 2006. Learning to laugh (automatically): Computational models for humor recognition. *Journal of Computational Intelligence*, 22(2):126–142, May.

George A. Miller. 1995. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41.

Saif M. Mohammad and Peter D. Turney. 2010. Emotions evoked by common words and phrases: using mechanical turk to create an emotion lexicon. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, CAAGET '10, pages 26–34, Stroudsburg, PA, USA. Association for Computational Linguistics.

Saif Mohammad. 2011. Even the abstract have color: Consensus in word-colour associations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 368–373, Portland, Oregon, USA, June. Association for Computational Linguistics.

Gözde Özbal and Carlo Strapparava. 2011. Automatized Memory Techniques for Vocabulary Acquisition in a Second Language. In Alexander Verbraeck, Markus Helfert, José Cordeiro, and Boris Shishkov, editors, *CSEDU*, pages 79–87. SciTePress.

Gozde Ozbal and Carlo Strapparava. 2012. A computational approach to the automation of creative naming. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 703–711, Jeju Island, Korea, July. Association for Computational Linguistics.

N. Sagarra and M. Alba. 2006. The key is in the keyword: L2 vocabulary learning methods with beginning learners of spanish. *The Modern Language Journal*, 90(2):228–243.

Oliviero Stock and Carlo Strapparava. 2006. Laughing with hahacronym, a computational humor system. In *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, pages 1675–1678. AAAI Press.

J. M. Toivanen, H. Toivonen, A. Valitutti, and O. Gross. 2012. Corpus-based Generation of Content and Form in Poetry. In *International Conference on Computational Creativity*, pages 175–179.

A. Valitutti, C. Strapparava, , and O. Stock. 2009. Graphlaugh: a tool for the interactive generation of humorous puns. In *Proceedings of ACII-2009, Third Conference on Affective Computing and Intelligent Interaction, Demo track*.