Learning Kernels over Strings using Gaussian Processes

Daniel Beck* **Trevor Cohn**
Computing and Information Systems
University of Melbourne, Australia
{d.beck, t.cohn}@unimelb.edu.au

Abstract

Non-contiguous word sequences are widely known to be important in modelling natural language. However they are not explicitly encoded in common text representations. In this work we propose a model for text processing using string kernels, capable of flexibly representing non-contiguous sequences. Specifically, we derive a vectorised version of the string kernel algorithm and their gradients, allowing efficient hyperparameter optimisation as part of a Gaussian Process framework. Experiments on synthetic data and text regression for emotion analysis show the promise of this technique.

1 Introduction

Text representations are a key component in any Natural Language Processing (NLP) task. A common approach for this is to average word vectors over a piece of text. For instance, a bag-of-words (BOW) model uses one-hot encoding as vectors and it is still a strong baseline for many tasks. More recently, approaches based on dense word representations (Turian et al., 2010; Mikolov et al., 2013) also showed to perform well.

However, averaging vectors discards any word order information from the original text, which can be fundamental for more involved NLP problems. Convolutional and recurrent neural networks (CNNs/RNNs) can keep word order but still treat a text fragment as a *contiguous* sequence of words, encoding a bias towards short- over long-distance relations between words. Some RNN models like the celebrated LSTMs (Hochreiter and Schmidhuber, 1997) perform better in capturing these phenomena but still have limitations. Recent work (Tai et al., 2015; Eriguchi et al., 2016) showed evidence that LSTM-based models can be enhanced by adding syntactic information, which can encode relations between non-contiguous words. This line of work requires the employment of accurate syntactic parsers, restricting their applicability to specific languages and/or text domains.

In this work we propose to revisit an approach which goes beyond contiguous word representations: string kernels (SKs). Their main power comes from the ability to represent arbitrary non-contiguous word sequences through dynamic programming algorithms. Our main contribution is a model that combines SKs with Gaussian Processes (GPs) (Rasmussen and Williams, 2006), allowing us to leverage efficient gradient-based methods to learn kernel hyperparameters. The reasoning behind our approach is that by optimising hyperparameters in a fine-grained way we can guide the kernel to learn better task-specific text representations automatically.

To enable the learning procedure we redefine the SK algorithm in a vectorised form and derive its gradients. We perform experiments using synthetic data, giving evidence that the model can capture non-trivial representations. Finally, we also show how the approach fares in a real dataset and explain how the learned hyperparameters can be interpreted as text representations.

2 String Kernels

Here we give a brief intuition[1] on string kernels, based on the formulation proposed by Cancedda

*This work was partially done while the first author was at The University of Sheffield, United Kingdom.

[1]We give a thorough explanation of the original SK equations in the Supplementary Material, as well as a detailed derivation of our vectorised version with its hyperparameter gradients.

Proceedings of the The 8th International Joint Conference on Natural Language Processing, pages 67–73,
Taipei, Taiwan, November 27 – December 1, 2017

et al. (2003). Let $|s|$ be the length of a string s, s_j the $j-$th symbol in s and $s_{:-1}$ the prefix containing the full string s except for the last symbol. Define $\mathrm{sim}(a, b)$ as a similarity measure between individual symbols a and b. Given two strings s and t and a maximum n-gram length n, the string kernel $k(s, t)$ can be obtained using the recursion

$$
\begin{aligned}
k'_0(s,t) &= 1, \text{for all } s,t, \\
&\quad \text{for all } i = 1,\ldots,n-1: \\
k'_i(sa,t) &= \lambda_g k'_i(s,t) + k''_i(sa,t), \\
k''_i(sa,tb) &= \lambda_g k''_i(sa,t) + \lambda_m^2 \mathrm{sim}(a,b) k'_{i-1}(s,t), \\
k_n(sa,t) &= k_n(s,t) + \\
&\quad \lambda_m^2 \sum_j^{|t|} \mathrm{sim}(a,t_j) k'_{n-1}(s,t_{:-1}), \\
k(s,t) &= \sum_{i=1}^{n} \mu_i k_i(s,t),
\end{aligned}
$$

where λ_g and λ_m are decay hyperparameters for symbol gaps and matches, respectively, and μ_i is the weight for the kernel of n-gram order i. The decay hyperparameters smooth the kernel values when sequences are very similar to each other while the n-gram weights help to tune the signal coming from different subsequence lengths.

Our goal is optimise the kernel hyperparameters in a fine-grained way using gradient-based methods. For this, we first redefine the kernel in a vectorised form. This not only eases gradient derivations but also allow our implementation to capitalise on recent advances in parallel processing and linear algebra libraries for better performance.[2] Given two strings s and t, the equations for our vectorised version are defined as

$$
\begin{aligned}
\mathbf{S} &= \mathbf{E}_s \mathbf{E}_t^T, \\
\mathbf{K}'_0 &= \mathbf{1}, \\
\mathbf{K}'_i &= \mathbf{D}_{|s|} \mathbf{K}''_i \mathbf{D}_{|t|}, \\
\mathbf{K}''_i &= \lambda_m^2 (\mathbf{S} \odot \mathbf{K}'_{i-1}), \\
k_i &= \lambda_m^2 \sum_{j,k} (\mathbf{S} \odot \mathbf{K}'_i)_{j,k}, \\
k(s,t) &= \boldsymbol{\mu}^T \mathbf{k},
\end{aligned}
$$

where $\mathbf{E}_s$ and $\mathbf{E}_t$ are matrices of symbol embeddings for each string, $\odot$ is the Hadamard product and $\mathbf{D}_\ell \in \mathbb{R}^\ell \times \mathbb{R}^\ell$ is the matrix

$$
\mathbf{D}_\ell = \begin{bmatrix}
0 & \lambda_g^0 & \lambda_g^1 & \ldots & \lambda_g^{|\ell|-2} \\
0 & 0 & \lambda_g^0 & \ldots & \lambda_g^{|\ell|-3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & \lambda_g^0 \\
0 & 0 & 0 & \ldots & 0
\end{bmatrix}
$$

with $\ell \in \{|s|, |t|\}$ being the corresponding string length for s or t. The purpose of D is to unroll the recursion from the original kernel equations. For the matrices $\mathbf{E}$, we focus on dense word embeddings but they could also be one-hot vectors, simulating hard matching between symbols.

Given this formulation, the hyperparameter gradients can be easily derived. From the vectorised definition, we can see that gradients with respect to $\boldsymbol{\mu}$ are simply $\mathbf{k}$, the intermediate n-gram specific kernel values. For λ_g and λ_m the gradients simply follow the kernel equations in an analogous manner. Note that the gradient calculations do not affect the time or space complexity of the main kernel, and in practice they can be obtained together using a single algorithm since they share many common terms.

Finally, we incorporate the kernel into a Gaussian Process regression model (henceforth, GP-SK). We assume the label y for an input string s is sampled from a function $f(s) \sim \mathcal{GP}(0, k(s,t))$, with t iterating over all other strings in a dataset. With this, we can define the marginal likelihood as

$$
\begin{aligned}
\log p(\mathbf{y}|\mathbf{s}, \boldsymbol{\theta}) &= \log \int_f p(\mathbf{y}|\mathbf{s}, \boldsymbol{\theta}, f) p(f), \\
&= -\frac{\mathbf{y}^T \mathbf{G}^{-1} \mathbf{y}}{2} - \frac{\log |\mathbf{G}|}{2} - \frac{n \log 2\pi}{2},
\end{aligned}
$$

where $\mathbf{G}$ is the Gram matrix with respect to the training set $\mathbf{s}$ and $\boldsymbol{\theta}$ is the set of kernel hyperparameters. By taking its derivative and plugging in the kernel gradients, we can optimise its hyperparameters using gradient-based methods.[3]

2.1 Complexity and Runtime Analysis

The original string kernel algorithm has complexity $O(n|s||s'|)$, i.e., quadratic in the size of the largest string. Our vectorised version is cubic, $O(n\ell^3)$, where $\ell = \max(|s|, |s'|)$, due to two matrix multiplications in the equations for $\mathbf{K}'_i$. Another way of reaching this result is to realise that

[2] Our open-source implementation is based on TensorFlow (Abadi et al., 2015).

[3] We refer the reader to Rasmussen and Williams (2006, Chap.5) for an in-depth explanation of this procedure.

$\mathbf{K}_i''$ is actually not needed anymore: all calculations can be made by updating $\mathbf{K}_i'$ only. In fact, Lodhi et al. (2002) introduced the term k'' as a way to reduce the complexity from $O(n|s||s'|^2)$ to $O(n|s||s'|)$. The complexity for the gradient calculations is also $O(n\ell^3)$.

However, even though our vectorised version has higher complexity, in practice we see large gains in runtime. We assess this empirically running the following experiment with synthetic strings:

- We employ characters as symbols with a one-hot encoding as the embedding, using all English ASCII letters, including uppercase (52 symbols in total);
- The maximum subsequence length is set to 5;
- 100 instances are generated randomly by uniformly sampling a character until reaching the desired string length.

We test our kernels with lengths ranging from 10 to 100.

Figure 1 shows wall-clock time measurements as the string lengths increase.[4] It is clear that the vectorised version is vastly faster than the original one, with up to two orders of magnitude. Comparing the CPU and GPU vectorised implementations, we see that we can reap benefits using a GPU when dealing with long sentences. GPU processing can be further enhanced by allowing portions of the Gram matrix to be calculated in batches instead of one instance at a time.

These results are intriguing because we do not expect a quadratic complexity algorithm to be outperformed by a cubic one. It is important to note that while we made efforts to optimise the code, there is no guarantee that either of our implementations is making the most of the underlying hardware. We plan to investigate these matters in more detail in the future.

3 Experiments

We assess our approach empirically with two sets of experiments using natural language sentences as inputs in a regression setting.[5] The first one uses synthetically generated response variables, providing a controlled environment to check the modelling capacities of our method. The second set uses labels from an emotion analysis dataset and serve as a proof of concept for our approach.

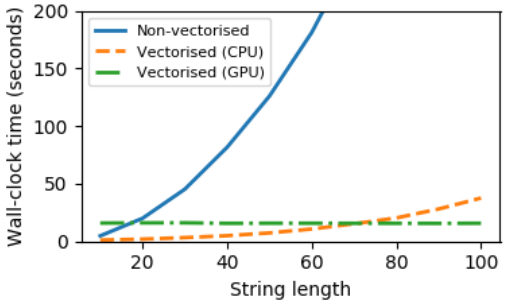

Figure 1: Wall-clock time measurements for the SK versions using different string lengths. Time is measured in seconds and correspond to the calculation of a 100 × 100 Gram matrix with random strings of a specific length.

3.1 Synthetic Labels

Consider an ideal scenario where the data is distributed according to a GP-SK. Here we aim at answering two questions: 1) whether we can retrieve the original distribution through optimisation and 2) whether a simpler model can capture the same distribution. The first gives us evidence on how feasible is to learn such a model while the second justify the choice of a SK compared to simpler alternatives.

To answer these questions we employ the following protocol. First we define a GP-SK with the following hyperparameter values:

$\lambda_g = 0.5$	$\lambda_m = 0.2$	$\sigma^2 = 0.1$
$\mu_1 = 1.0$	$\mu_2 = 0.5$	$\mu_3 = 0.25$

where σ^2 is the label GP noise. This choice of hyperparameter values is arbitrary: our goal is simply to check if we can retrieve these values through optimisation. The same procedure could be applied for different values.

After defining the GP-SK model we calculate the its corresponding Gram matrix using a set of sentences and their respective word embeddings. This matrix contains the covariances of a multivariate Gaussian distribution with mean vector $\mathbf{0}$ and we can sample from this Gaussian to create synthetic labels. As inputs we use a random sample of sentences from the Penn Treebank (Marcus

[4] Experiments were done in a machine with an Intel Xeon E5-2687W 3.10GHz as CPU and a GTX TITAN X as GPU.

[5] Code to replicate the experiments in this section is available at `https://github.com/beckdaniel/ijcnlp17_sk`. This also include the performance experiments in Section 2.1.

et al., 1993) and represent each word as a 100d GloVe embedding (Pennington et al., 2014).[6]

The data sampled from the procedure above is used to train another GP-SK with *randomly initialised* hyperparameters, which are then optimised. We run this procedure 20 times, using the same inputs but sampling new labels every time.

Hyperparameter stability Figure 2 shows the hyperparameter values retrieved after optimisation, for increasing training set sizes. The decay hyperparameters are the most stable ones, being retrieved with high confidence independent of the dataset size. The original noise value is also obtained but it needs more instances (1000) for that.

The n-gram coefficients are less stable compared to the other hyperparameters, although the uncertainty seems to diminish with larger datasets. A possible explanation is the presence of some level of overspecification, meaning that very different coefficient values may reach similar marginal likelihoods, which in turn corresponds to multiple plausible explanations of the data. Solutions for this include imposing bounds to the coefficients or fixing them, while freely optimising the more stable hyperparameters.

Predictive performance To evaluate if the GP-SK models can be subsumed by simpler ones, we assess the predictive performance on a disjoint test set containing 200 sentences. Test labels were sampled from the same GP-SK distribution used to generate the training labels, simulating a setting where all the data follows the same distribution.

Figure 3 shows results across different training set sizes, in Pearson's r correlation. We compare with GP baselines trained on averaged embeddings, using either a linear or a Squared Exponential (SE)[7] kernel. The SK model outperforms the baselines, showing that even a non-linear model can not capture the GP-SK distribution.

To investigate the influence of hyperparameter optimisation, we also show results in Figure 3 for a SK model with randomly initialised hyperparameter values. Clearly, optimisation helps to improve the model, even in the low data scenarios.

3.2 Emotion Analysis

As a first step towards working with real world data, we employ the proposed approach in an emotion analysis setting, where the goal is to model latent emotions in text. We use the "Affective Text" dataset from the SemEval2007 shared task (Strapparava and Mihalcea, 2007), composed of 1,250 News headlines annotated with 6 scores, one per emotion. Scores are in the $[0-100]$ range and were provided by human judges. The models, baselines and embeddings are the same used in Section 3.1. Instead of using a fixed split, we perform 10-fold

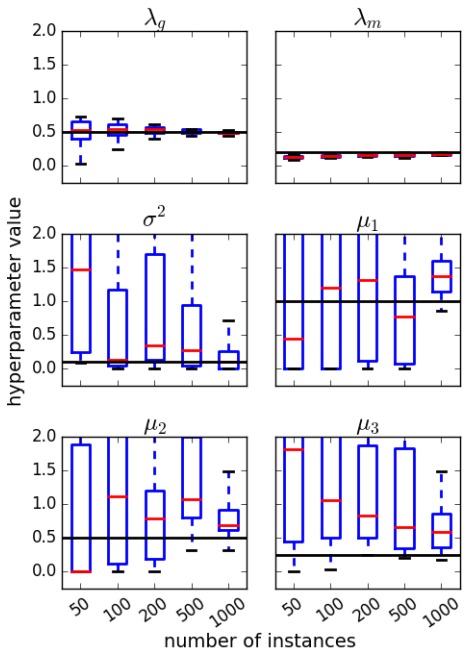

Figure 2: String kernel hyperparameter optimisation results. Original hyperparameter values are shown as black lines while each box corresponds to a specific dataset size. Red lines show the median values, while box limits correspond to the $[0.25, 0.75]$ quantiles.

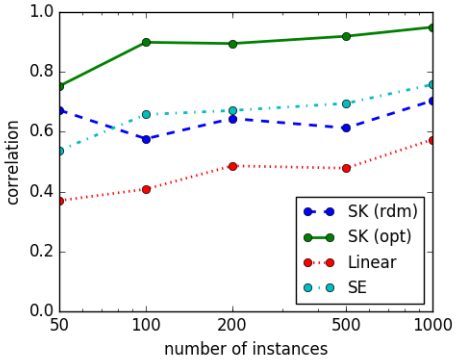

Figure 3: Prediction results, averaged over 20 runs. "SK (rdm)" corresponds to string kernel with random hyperparameter values and "SK (opt)", with optimised hyperparameters.

[6] nlp.stanford.edu/projects/glove. We use the version trained on Wikipedia and Gigaword.

[7] Also known as RBF kernel.

	NLPD ↓	MAE ↓	r ↑
SK	4.06	10.53	0.586
Linear	4.09	11.03	0.539
SE	4.03	10.09	0.611

Table 1: Emotion analysis results, averaged over all emotions and cross-validation folds.

λ_g	7.36×10^{-7}	λ_m	0.0918		
μ_1	12.37	μ_2	33.73	μ_3	154.51
μ_4	2.58	μ_5	8.54		

Table 2: SK hyperparameter values for a single model predicting the emotion *surprise*.

cross validation and average the results.

Table 1 compares the performance of GP-SK with the baselines trained on averaged embeddings. Besides Pearson's r correlation, we also compare the models in terms of Mean Absolute Error (MAE) and Negative Log Predictive Density (NLPD), a metric that takes into account the full predictive distribution into account (Quiñonero-Candela et al., 2006). The main figure is that GP-SK outperforms the linear baseline but lags behind the SE one. This shows that non-linearities present in the data can not be captured by the GP-SK model. Since the string kernel is essentially a dot product over exponentially-sized vectors, it is not surprising that it is unable to capture non-linear behaviour. This gives us evidence that developing non-linear extensions of string kernels could be a promising avenue for future work.

Inspecting hyperparameters Probing the hyperparameters can give us insights about what kind of representation the kernel is learning. On Table 2 we show the values for one of the models that predict the emotion *surprise*. We can see that λ_g has a very low value, while the $\boldsymbol{\mu}$ values show a preference for subsequences up to 3 words. This lets us conclude that the kernel learned a text representation close to *contiguous trigrams*.

4 Related Work

String kernels were originally proposed for text classification (Lodhi et al., 2002; Cancedda et al., 2003) while recent work apply them for native language identification (Ionescu et al., 2014) and sentiment analysis (Giménez-Pérez et al., 2017), with promising results. Hyperparameter optimisation in these works is done via grid search and could potentially benefit from our proposed approach.

Gaussian Processes have been recently employed in a number of NLP tasks such as emotion analysis (Beck et al., 2014), detection of temporal patterns in microblogs (Preoiuc-Pietro and Cohn, 2013), rumour propagation in social media (Lukasik et al., 2015) and translation quality estimation (Cohn and Specia, 2013; Shah et al., 2013; Beck et al., 2016). These previous works encode text inputs as fixed-size vectors instead of working directly on the text inputs.

Among other recent work that aim at learning general structured kernels, the most similar to ours is Beck et al. (2015), who use GPs to learn tree kernels. Lei et al. (2017) unroll string kernel computations and derive equivalent neural network architectures. In contrast, our work put the learning procedure inside a GP model, inheriting the advantages of Bayesian model selection procedures. Nevertheless, many of their kernel ideas could be applied to a GP setting, which we leave for future work.

5 Conclusion

In this paper we provided the first steps in combining string kernels and Gaussian Processes for NLP tasks, allowing us to learn the text representations used by the kernels by optimising its hyperparameters in a fine-grained way. Experiments showed promising results in capturing text patterns that are not modelled by simpler baselines.

For future work, we plan to extend the model to account for non-linear representations, using approaches such as Arc-cosine kernels (Cho and Saul, 2009) and also applying the ideas from Lei et al. (2017). Another important avenue to pursue is to scale the model to larger datasets using recent advances in Sparse GPs (Titsias, 2009; Hensman et al., 2013). These in turn can enable richer kernel parameterisations not only for strings but other structures as well.

Acknowledgements

Daniel Beck was supported by funding from CNPq/Brazil (No. 237999/2012-9) and from the Australian Research Council (DP160102686). Trevor Cohn is the recipient of an Australian Research Council Future Fellowship (project number FT130101105). The authors would also like to thank the anonymous reviewers for their comments.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafael Josefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

Daniel Beck, Trevor Cohn, Christian Hardmeier, and Lucia Specia. 2015. Learning Structural Kernels for Natural Language Processing. *Transactions of the Association for Computational Linguistics*, 3:461–473.

Daniel Beck, Trevor Cohn, and Lucia Specia. 2014. Joint Emotion Analysis via Multi-task Gaussian Processes. In *Proceedings of EMNLP*, pages 1798–1803.

Daniel Beck, Lucia Specia, and Trevor Cohn. 2016. Exploring Prediction Uncertainty in Machine Translation Quality Estimation. In *Proceedings of CoNLL*.

Nicola Cancedda, Eric Gaussier, Cyril Goutte, and Jean-Michel Renders. 2003. Word-Sequence Kernels. *The Journal of Machine Learning Research*, 3:1059–1082.

Youngmin Cho and Lawrence K Saul. 2009. Kernel Methods for Deep Learning. In *Proceedings of NIPS*, pages 1–9.

Trevor Cohn and Lucia Specia. 2013. Modelling Annotator Bias with Multi-task Gaussian Processes: An Application to Machine Translation Quality Estimation. In *Proceedings of ACL*, pages 32–42.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-Sequence Attentional Neural Machine Translation. In *Proceedings of ACL*.

Rosa M. Giménez-Pérez, Marc Franco-Salvador, and Paolo Rosso. 2017. Single and Cross-domain Polarity Classification using String Kernels. In *Proceedings of EACL*, pages 558–563.

James Hensman, Nicolò Fusi, and Neil D. Lawrence. 2013. Gaussian Processes for Big Data. In *Proceedings of UAI*, pages 282–290.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural Computation*, 9(8):1735–80.

Radu Tudor Ionescu, Marius Popescu, and Aoife Cahill. 2014. Can characters reveal your native language? A language-independent approach to native language identification. In *Proceedings of EMNLP*, pages 1363–1373.

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2017. Deriving Neural Architectures from Sequence and Graph Kernels. In *Proceedings of ICML*.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text Classification using String Kernels. *The Journal of Machine Learning Research*, 2:419–444.

Michal Lukasik, Trevor Cohn, and Kalina Bontcheva. 2015. Point Process Modelling of Rumour Dynamics in Social Media. In *Proceedings of ACL*, pages 518–523.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of NIPS*, pages 1–9.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of EMNLP*, pages 1532–1543.

Daniel Preoiuc-Pietro and Trevor Cohn. 2013. A temporal model of text periodicities using Gaussian Processes. In *Proceedings of EMNLP*, pages 977–988.

Joaquin Quiñonero-Candela, Carl Edward Rasmussen, Fabian Sinz, Olivier Bousquet, and Bernhard Schölkopf. 2006. Evaluating Predictive Uncertainty Challenge. *MLCW 2005, Lecture Notes in Computer Science*, 3944:1–27.

Carl Edward Rasmussen and Christopher K. I. Williams. 2006. *Gaussian processes for machine learning*, volume 1. MIT Press Cambridge.

Kashif Shah, Trevor Cohn, and Lucia Specia. 2013. An Investigation on the Effectiveness of Features for Translation Quality Estimation. In *Proceedings of MT Summit XIV*.

Carlo Strapparava and Rada Mihalcea. 2007. SemEval-2007 Task 14 : Affective Text. In *Proceedings of SemEval*, pages 70–74.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of ACL*.

Michalis K. Titsias. 2009. Variational Learning of Inducing Variables in Sparse Gaussian Processes. In *Proceedings of AISTATS*, volume 5, pages 567–574.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word Representations: A Simple and General Method for Semi-supervised Learning. In *Proceedings of ACL*, pages 384–394.