Learning Text Similarity with Siamese Recurrent Networks

Paul Neculoiu, Maarten Versteegh and **Mihai Rotaru**
Textkernel B.V. Amsterdam
{neculoiu,versteegh,rotaru}@textkernel.nl

Abstract

This paper presents a deep architecture for learning a similarity metric on variable-length character sequences. The model combines a stack of character-level bidirectional LSTM's with a Siamese architecture. It learns to project variable-length strings into a fixed-dimensional embedding space by using only information about the similarity between pairs of strings. This model is applied to the task of job title normalization based on a manually annotated taxonomy. A small data set is incrementally expanded and augmented with new sources of variance. The model learns a representation that is selective to differences in the input that reflect semantic differences (e.g., "Java developer" vs. "HR manager") but also invariant to non-semantic string differences (e.g., "Java developer" vs. "Java programmer").

1 Introduction

Text representation plays an important role in natural language processing (NLP). Tasks in this field rely on representations that can express the semantic similarity and dissimilarity between textual elements, be they viewed as sequences of words or characters. Such representations and their associated similarity metrics have many applications. For example, word similarity models based on dense embeddings (Mikolov et al., 2013) have recently been applied in diverse settings, such as sentiment analysis (dos Santos and Gatti, 2014) and recommender systems (Barkan and Koenigstein, 2016). Semantic textual similarity measures have been applied to tasks such as automatic summarization (Ponzanelli et al., 2015), debate analysis (Boltuzic and Šnajder, 2015) and paraphrase detection (Socher et al., 2011).

Measuring the semantic similarity between texts is also fundamental problem in Information Extraction (IE) (Martin and Jurafsky, 2000). An important step in many applications is normalization, which puts pieces of information in a standard format, so that they can be compared to other pieces of information. Normalization relies crucially on semantic similarity. An example of normalization is formatting dates and times in a standard way, so that "12pm", "noon" and "12.00h" all map to the same representation. Normalization is also important for string values. Person names, for example, may be written in different orderings or character encodings depending on their country of origin. A sophisticated search system may need to understand that the strings "李小龙", "Lee, Junfan" and "Bruce Lee" all refer to the same person and so need to be represented in a way that indicates their semantic similarity. Normalization is essential for retrieving actionable information from free, unstructured text.

In this paper, we present a system for job title normalization, a common task in information extraction for recruitment and social network analysis (Javed et al., 2014; Malherbe et al., 2014). The task is to receive an input string and map it to one of a finite set of job codes, which are predefined externally. For example, the string "software architectural technician Java/J2EE" might need to be mapped to "Java developer". This task can be approached as a highly multi-class classification problem, but in this study, the approach we take focuses on learning a representation of the strings such that synonymous job titles are close together. This approach has the advantage that it is flexible, i.e., the representation can function as the input space to a subsequent classifier, but can also be used to find closely related job titles or explore job title clusters. In addition, the architecture of the learning model allows us to learn useful representations with limited supervision.

Proceedings of the 1st Workshop on Representation Learning for NLP, pages 148–157,
Berlin, Germany, August 11th, 2016.

2 Related Work

The use of (deep) neural networks for NLP has recently received much attention, starting from the seminal papers employing convolutional networks on traditional NLP tasks (Collobert et al., 2011) and the availability of high quality semantic word representations (Mikolov et al., 2013). In the last few years, neural network models have been applied to tasks ranging from machine translation (Zou et al., 2013; Cho et al., 2014) to question answering (Weston et al., 2015). Central to these models, which are usually trained on large amounts of labeled data, is feature representation. Word embedding techniques such as word2vec (Mikolov et al., 2013) and Glove (Pennington et al., 2014) have seen much use in such models, but some go beyond the word level and represent text as a sequence of characters (Kim et al., 2015; Ling et al., 2015). In this paper we take the latter approach for the flexibility it affords us in dealing with out-of-vocabulary words.

Representation learning through neural networks has received interest since autoencoders (Hinton and Salakhutdinov, 2006) have been shown to produce features that satisfy the two desiderata of representations; that they are *invariant* to differences in the input that do not matter for that task and *selective* to differences that do (Anselmi et al., 2015).

The Siamese network (Bromley et al., 1993) is an architecture for non-linear metric learning with similarity information. The Siamese network naturally learns representations that embody the invariance and selectivity desiderata through explicit information about similarity between pairs of objects. In contrast, an autoencoder learns invariance through added noise and dimensionality reduction in the bottleneck layer and selectivity solely through the condition that the input should be reproduced by the decoding part of the network. In contrast, a Siamese network learns an invariant and selective representation directly through the use of similarity and dissimilarity information.

Originally applied to signature verification (Bromley et al., 1993), the Siamese architecture has since been widely used in vision applications. Siamese convolutional networks were used to learn complex similarity metrics for face verification (Chopra et al., 2005) and dimensionality reduction on image features (Hadsell et al., 2006). A variant of the Siamese network, the triplet network (Hoffer and Ailon, 2015), was used to learn an image similarity measure based on ranking data (Wang et al., 2014).

In other areas, Siamese networks have been applied to such diverse tasks as unsupervised acoustic modelling (Synnaeve et al., 2014; Thiolliere et al., 2015; Kamper et al., 2016; Zeghidour et al., 2015), learning food preferences (Yang et al., 2015) and scene detection (Baraldi et al., 2015). In NLP applications, Siamese networks with convolutional layers have been applied to matching sentences (Hu et al., 2014). More recently, (Mueller and Thyagarajan, 2016) applied Siamese recurrent networks to learning semantic entailment.

The task of job title normalization is often framed as a classification task (Javed et al., 2014; Malherbe et al., 2014). Given the large number of classes (often in the thousands), multi-stage classifiers have shown good results, especially if information outside the string can be used (Javed et al., 2015). There are several disadvantage to this approach. The first is the expense of data acquisition for training. With many thousands of groups of job titles, often not too dissimilar from one another, manually classifying large amounts of job title data becomes prohibitively expensive. A second disadvantage of this approach is its lack of corrigibility. Once a classification error has been discovered or a new example has been added to a class, the only option to improve the system is to retrain the entire classifier with the new sample added to the correct class in the training set. The last disadvantage is that using a traditional classifier does not allow for transfer learning, i.e., reusing the learned model's representations for a different task.

A different approach is the use of string similarity measures to classify input strings by proximity to an element of a class (Spitters et al., 2010). The advantage of this approach is that there is no need to train the system, so that improvements can be made by adding job title strings to the data. The disadvantages are that data acquisition still needs to be performed by manually classifying strings and that the bulk of the problem is now shifted to constructing a good similarity metric.

By modeling similarity directly based on pairs of inputs, Siamese networks lend themselves well to the semantic invariance phenomena present in job title normalization: typos (e.g. "Java developeur"), near-synonymy (e.g., "developer" and

"programmer") and extra words (e.g., "experienced Java developer"). This is the approach we take in this study.

3 Siamese Recurrent Neural Network

Recurrent Neural Networks (RNN) are neural networks adapted for sequence data $(x_1, \ldots, x_T)$. At each time step $t \in \{1, \ldots, T\}$, the hidden-state vector h_t is updated by the equation $h_t = \sigma(Wx_t + Uh_{t-1})$, in which x_t is the input at time t, W is the weight matrix from inputs to the hidden-state vector and U is the weight matrix on the hidden-state vector from the previous time step h_{t-1}. In this equation and below the logistic function is denoted by $\sigma(x) = (1 + e^{-x})^{-1}$.

The Long Short-Term Memory (Hochreiter and Schmidhuber, 1997) variant of RNNs in particular has had success in tasks related to natural language processing, such as text classification (Graves, 2012) and language translation (Sutskever et al., 2014). Standard RNNs suffer from the vanishing gradient problem in which the backpropagated gradients become vanishingly small over long sequences (Pascanu et al., 2013). The LSTM model was proposed as a solution to this problem. Like the standard RNN, the LSTM sequentially updates a hidden-state representation, but it introduces a memory state c_t and three gates that control the flow of information through the time steps. An output gate o_t determines how much of c_t should be exposed to the next node. An input gate i_t controls how much the input x_t matters at this time step. A forget gate f_t determines whether the previous time step's memory should be forgotten. An LSTM is parametrized by weight matrices from the input and the previous state for each of the gates, in addition to the memory cell. We use the standard formulation of LSTMs with the logistic function (σ) on the gates and the hyperbolic tangent (`tanh`) on the activations. In the equations (1) below, $\circ$ denotes the Hadamard (elementwise) product.

$$i_t = \sigma(W_i x_t + U_i h_{t-1}) \quad (1)$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1}) \quad (2)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1}) \quad (3)$$
$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1}) \quad (4)$$
$$c_t = i_t \circ \tilde{c}_t + f_t \circ c_{t-1} \quad (5)$$
$$h_t = o_t \circ \tanh(c_t) \quad (6)$$

Bidirectional RNNs (Schuster and Paliwal, 1997) incorporate both future and past context by running the reverse of the input through a separate RNN. The output of the combined model at each time step is simply the concatenation of the outputs from the forward and backward networks. Bidirectional LSTM models in particular have recently shown good results on standard NLP tasks like Named Entity Recognition (Huang et al., 2015; Wang et al., 2015) and so we adopt this technique for this study.

Siamese networks (Chopra et al., 2005) are dual-branch networks with tied weights, i.e., they consist of the same network copied and merged with an energy function. Figure 1 shows an overview of the network architecture in this study. The training set for a Siamese network consists of triplets (x_1, x_2, y), where x_1 and x_2 are character sequences and $y \in \{0, 1\}$ indicates whether x_1 and x_2 are similar ($y = 1$) or dissimilar ($y = 0$). The aim of training is to minimize the distance in an embedding space between similar pairs and maximize the distance between dissimilar pairs.

3.1 Contrastive loss function

The proposed network contains four layers of Bidirectional LSTM nodes. The activations at each timestep of the final BLSTM layer are averaged to produce a fixed-dimensional output. This output is projected through a single densely connected feedforward layer.

Let $f_{\mathrm{W}}(x_1)$ and $f_{\mathrm{W}}(x_2)$ be the projections of x_1 and x_2 in the embedding space computed by the network function f_{W}. We define the energy of the model E_{W} to be the cosine similarity between the embeddings of x_1 and x_2:

$$E_{\mathrm{W}}(x_1, x_2) = \frac{\langle f_{\mathrm{W}}(x_1), f_{\mathrm{W}}(x_2) \rangle}{\|f_{\mathrm{W}}(x_1)\| \|f_{\mathrm{W}}(x_2)\|} \quad (7)$$

For brevity of notation, we will denote $E_{\mathrm{W}}(x_1, x_2)$ by E_{W}. The total loss function over a data set $X = \left\{ \langle x_1^{(i)}, x_2^{(i)}, y^{(i)} \rangle \right\}$ is given by:

$$\mathcal{L}_{\mathrm{W}}(X) = \sum_{i=1}^{N} L_{\mathrm{W}}^{(i)}(x_1^{(i)}, x_2^{(i)}, y^{(i)}) \quad (8)$$

The instance loss function $L_{\mathrm{W}}^{(i)}$ is a contrastive loss function, composed of terms for the similar ($y =$

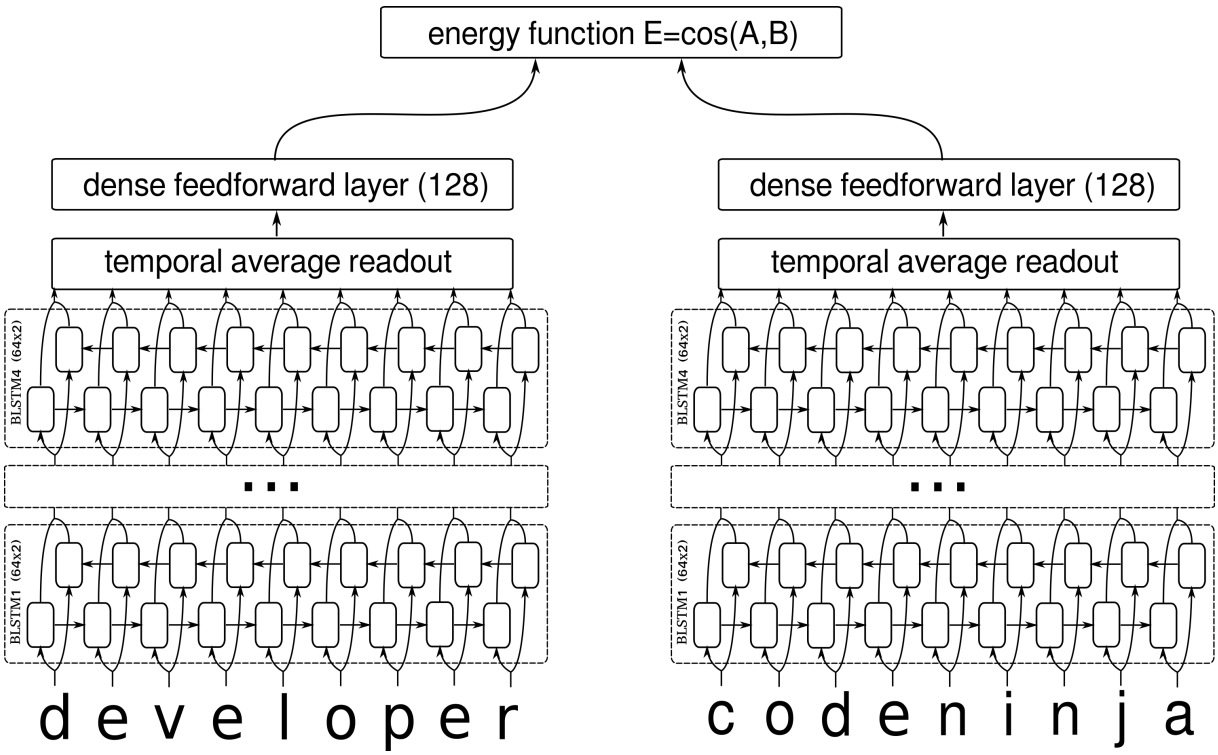

Figure 1: Overview of the Siamese Recurrent Network architecture used in this paper. The weights of all the layers are shared between the right and the left branch of the network.

1) case (L_+), and the dissimilar ($y = 0$) case (L_-):

$$L_{\mathrm{W}}^{(i)} = y^{(i)} L_+(x_1^{(i)}, x_2^{(i)}) + \tag{9}$$

$$(1 - y^{(i)}) L_-(x_1^{(i)}, x_2^{(i)}) \tag{10}$$

$$\tag{11}$$

The loss functions for the similar and dissimilar cases are given by:

$$L_+(x_1, x_2) = \frac{1}{4}(1 - E_{\mathrm{W}})^2 \tag{12}$$

$$L_-(x_1, x_2) = \begin{cases} E_{\mathrm{W}}^2 & \text{if } E_{\mathrm{W}} < m \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Figure 2 gives a geometric perspective on the loss function, showing the positive and negative components separately. Note that the positive loss is scaled down to compensate for the sampling ratios of positive and negative pairs (see below).

The network used in this study contains four BLSTM layers with 64-dimensional hidden vectors h_t and memory c_t. There are connections at each time step between the layers. The outputs of the last layer are averaged over time and this 128-dimensional vector is used as input to a dense feedforward layer. The input strings are padded to

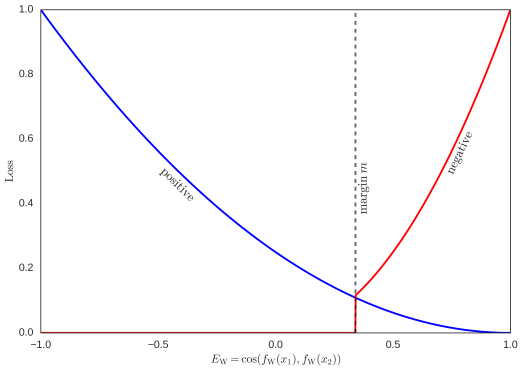

Figure 2: Positive and negative components of the loss function.

produce a sequence of 100 characters, with the input string randomly placed in this sequence. The parameters of the model are optimized using the Adam method (Kingma and Ba, 2014) and each model is trained until convergence. We use the dropout technique (Srivastava et al., 2014) on the recurrent units (with probability 0.2) and between layers (with probability 0.4) to prevent overfitting.

4 Experiments

We conduct a set of experiments to test the model's capabilities. We start from a small data set based on a hand made taxonomy of job titles. In each subsequent experiment the data set is augmented by adding new sources of variance. We test the model's behavior in a set of unit tests, reflecting desired capabilities of the model, taking our cue from (Weston et al., 2015). This section discusses the data augmentation strategies, the composition of the unit tests, and the results of the experiments.

4.1 Baseline

Below we compare the performance of our model against a baseline n-gram matcher (Daelemans et al., 2004). Given an input string, this matcher looks up the closest neighbor from the base taxonomy by maximizing a similarity scoring function. The matcher subsequently labels the input string with that neighbor's group label. The similarity scoring function is defined as follows. Let $Q = \langle q_1, \ldots, q_M \rangle$ be the query as a sequence of characters and $C = \langle c_1, \ldots, c_N \rangle$ be the candidate match from the taxonomy. The similarity function is defined as:

$$\begin{aligned}
\mathrm{sim}(Q, C) &= M - \mathrm{match}(\mathrm{Q}, \mathrm{C}) \\
\mathrm{match}(Q, C) &= |T_Q \ominus T_C| - |T_Q \cap T_C| \\
&\text{where} \\
A \ominus B &= (A \setminus B) \cup (B \setminus A) \\
T_Q &= \bigcup_{i=1}^{M-2} \{\langle q_i, q_{i+1}, q_{i+2} \rangle\} \\
T_C &= \bigcup_{i=1}^{N-2} \{\langle c_i, c_{i+1}, c_{i+2} \rangle\}
\end{aligned}$$

This (non-calibrated) similarity function has the properties that it is easy to compute, doesn't require any learning and is particularly insensitive to appending extra words in the input string, one of the desiderata listed below.

In the experiments listed below, the test sets consist of pairs of strings, the first of which is the input string and the second a target group label from the base taxonomy. The network model projects the input string into the embedding space and searches for its nearest neighbor under cosine distance from the base taxonomy. The test records a hit if and only if the neighbor's group label matches the target.

4.2 Data and Data Augmentation

The starting point for our data is a hand made proprietary job title taxonomy. This taxonomy partitions a set of 19,927 job titles into 4,431 groups. Table 1 gives some examples of the groups in the taxonomy. The job titles were manually and semi-automatically collected from résumés and vacancy postings. Each was manually assigned a group, such that the job titles in a group are close together in meaning. In some cases this closeness is an expression of a (near-)synonymy relation between the job titles, as in "developer" and "developer/programmer" in the "Software Engineer" category. In other cases a job title in a group is a specialization of another, for example "general operator" and "buzz saw operator" in the "Machine Operator" category. In yet other cases two job titles differ only in their expression of seniority, as in "developer" and "senior developer" in the "Software Engineer" category. In all cases, the relation between the job titles is one of *semantic* similarity and not necessarily surface form similarity. So while, "Java developer" and "J2EE programmer" are in the same group, "Java developer" and "real estate developer" should not be.

Note that some groups are close together in meaning, like the "Production Employee" and "Machine Operator" groups. Some groups could conceivably be split into two groups, depending on the level of granularity that is desired. We make no claim to completeness or consistency of these groupings, but instead regard the wide variety of different semantic relations between and within groups as an asset that should be exploited by our model.

The groups are not equal in size; the sizes follow a broken power-law distribution. The largest group contains 130 job titles, the groups at the other end of the distribution have only one. This affects the amount of information we can give to the system with regards to the semantic similarity between job titles in a group. The long tail of the distribution may impact the model's ability to accurately learn to represent the smallest groups. Figure 3 shows the distribution of the group sizes of the original taxonomy.

We proceed from the base taxonomy of job titles in four stages. At each stage we introduce (1) an

Customer Service Agent	Production Employee	Software Engineer	Machine Operator	Software Tester
support specialist	assembler	developer	operator punch press	tester sip
service desk agent	manufacturing assistant	application programmer	machinist	test consultant
support staff	production engineer	software architect	buzz saw operator	stress tester
customer care agent III	factory employee	cloud engineer	operator turret punch press	kit tester
customer service agent	casting machine operator	lead software engineer	blueprint machine operator	agile java tester
customer interaction	helper production	senior developer	general operator	test engineer
customer care officer	production laborer	developer/programmer	operator nibbler	QTP tester

Table 1: Example job title groups from the taxonomy. The total taxonomy consists of 19,927 job titles in 4,431 groups.

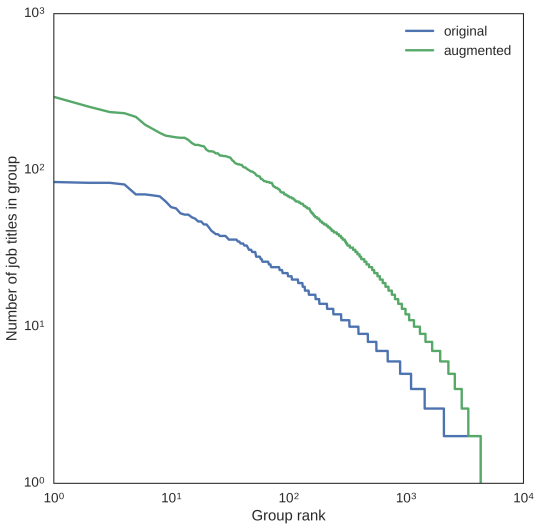

Figure 3: The distributions of group sizes in the original taxonomy (blue) and the taxonomy augmented with synonym substitutions (green) follow broken power-law distributions. Note that both axes are on a logarithmic scale. The figure shows the long tail of the distribution, in which groups contain one or only a few job titles.

augmentation of the data which focuses on a particular property and (2) a test that probes the model for behavior related to that property. Each stage builds on the next, so the augmentations from the previous stage are always included. Initially, the data set consists of pairs of strings sampled from the taxonomy in a 4:1 ratio of between-class (negative) pairs to within-class (positive) pairs. This ratio was empirically determined but other studies have found a similar optimal ratio of negative to positive pairs in Siamese networks (Synnaeve and Dupoux, 2016). In the subsequent augmentations, we keep this ratio constant.

1. Typo and spelling invariance. Users of the system may supply job titles that differ in spelling from what is present in the taxonomy (e.g., "laborer" vs "labourer") or they may make a typo and insert, delete or substitute a character. To induce invariance to these we augment the base taxonomy by extending it with positive sample pairs consisting of job title strings and the same string but with 20% of characters randomly substituted and 5% randomly deleted. Of the resulting training set, 10% consists of these typo pairs. The corresponding test set (**Typos**) consists of all the 19,928 job title strings in the taxonomy with 5% of their characters randomly substituted or deleted. This corresponds to an approximate upper bound on the proportion of spelling errors (Salthouse, 1986).

2. Synonyms. Furthermore, the model must be invariant to synonym substitution. To continue on the example given above, the similarity between "Java developer" and "Java programmer" show that in the context of computer science "developer" and "programmer" are synonyms. This entails that, given the same context, "developer" can be substituted for "programmer" in any string in which it occurs without altering the meaning of that string. So "C++ developer" can be changed into "C++ programmer" and still refer to the same job. Together with the selectivity constraint, the invariance to synonym substitution constitutes a form of compositionality on the component parts of job titles. A model with this compositionality property will be able to generalize over the meanings of parts of job titles to form useful representations of unobserved inputs. We augment the data set by substituting words in job titles by synonyms from two sources. The first source is a manually constructed job title synonym set, consisting of around 1100 job titles, each with between one and ten synonyms for a total of 7000 synonyms. The second source of synonyms is by induction. As in the example above, we look through the taxonomy for groups in which two job titles share one or two words, e.g., "C++". The complements of the matching strings form a synonym candidate, e.g., "developer" and "programmer". If the can-

didate meets certain requirements (neither part occurs in isolation, the parts do not contain special characters like '&', the parts consist of at most two words), then the candidate is accepted as a synonym and is substituted throughout the group. The effect of this augmentation on the group sizes is shown in figure 3. The corresponding test set (**Composition**) consists of a held out set of 7909 pairs constructed in the same way.

3. Extra words. To be useful in real-world applications, the model must also be invariant to the presence of superfluous words. Due to parsing errors or user mistakes the input to the normalization system may contain strings like "looking for C++ developers (urgent!)", or references to technologies, certifications or locations that are not present in the taxonomy. Table 2 shows some examples of real input. We augment the data set by extracting examples of superfluous words from real world data. We construct a set by selecting those input strings for which there is a job title in the base taxonomy which is the complete and strict substring of the input and which the baseline n-gram matcher selects as the normalization. As an example, in table 2, the input string "public relations consultant business business b2c" contains the taxonomy job title "public relations consultant". Part of this set ($N = 1949$) is held out from training and forms the corresponding test set (**Extra Words**).

Input string
supervisor dedicated services share plans
part II architectural assistant or architect at
geography teacher 0.4 contract now
customer relationship management developer super user â
forgot password
public relations consultant business business b2c
teaching assistant degree holders only contract

Table 2: Example input strings to the system.

4. Feedback. Lastly, and importantly for industrial applications, we would like our model to be *corrigible*, i.e., when the model displays undesired behavior or our knowledge about the domain increases, we want the model to facilitate manual intervention. As an example, if the trained model assigns a high similarity score to the string "Java developer" and "Coffee expert (Java, Yemen)" based on the corresponding substrings, we would like to be able to signal to the model that these particular instances do not belong together. To test this behavior, we manually scored a set of 11929 predictions. This set was subsequently used for training. The corresponding test set (**Annotations**) consists of a different set of 1000 manually annotated held-out input strings.

4.3 Results

Table 3 shows the results of the experiments. It compares the baseline n-gram system and proposed neural network models on the four tests outlined above. Each of the neural network models (1)-(4) was trained on augmentations of the data set that the previous model was trained on.

The first thing to note is that both the n-gram matching system and the proposed models have near-complete invariance to simple typos. This is of course expected behavior, but this test functions as a good sanity check on the surface form mapping to the representations that the models learn.

In the performance of all tests except for the Annotations test, we see a strong effect of the associated augmentation. Model (1) shows 0.04 improvement over model (0) on the typo test. This indicates that the proposed architecture is suitable for learning invariance to typos, but that the addition of typos and spelling variants to the training input only produces marginal improvements over the already high accuracy on this test.

Model (2) shows 0.29 improvement over model (1) on the Composition test. This indicates that model (2) has successfully learned to combine the meanings of individual words in the job titles into new meanings. This is an important property for a system that aims to learn semantic similarity between text data. Compositionality is arguably the most important property of human language and it is a defining characteristic of the way we construct compound terms such as job titles. Note also that the model learned this behavior based largely on observations of *combinations* of words, while having little evidence on the individual meanings.

Model (3) shows 0.45 improvement over model (2) on the Extra Words test, jumping from 0.29 accuracy to 0.76. This indicates firstly that the proposed model can successfully learn to ignore large portions of the input sequence and secondly that the evidence of extra words around the job title is crucial for the system to do so. Being able to ignore subsequences of an input sequence is an important ability for information extraction systems.

The improvements on the Annotations test is also greatest when the extra words are added to the

	Typos ($N = 19928$)	Composition ($N = 7909$)	Extra Words ($N = 1949$)	Annotations ($N = 1000$)
n-gram	0.99	0.61	1.00*	0.83
(0) RNN base taxonomy	0.95	0.55	0.40	0.69
(1) + typos	0.99	0.54	0.36	0.77
(2) + synonyms	**1.00**	0.83	0.29	0.76
(3) + extra words	**1.00**	**0.84**	0.76	**0.87**
(4) + feedback	**1.00**	0.79	**0.82**	0.84

Table 3: Accuracy of the baseline and models on each of the four test cases. The best performing neural network in each column is indicated in **bold**. Note that the performance of the n-gram match system (*) on the Extra Words test is 1.00 by construction.

training set. Model (4) actually shows a decrease in performance with respect to model (3) on this test. The cause for this is likely the fact that the Extra Words test and the held out Annotations tests show a lot of similarity in the structure of their inputs. Real production inputs often consist of additional characters, words and phrases before or after the actual job title. It is unclear why model (4) shows an improvement on the Extra Words test while simultaneously showing a decrease in performance on the Composition and Annotations tests. This matter is left to future investigation.

5 Discussion

In this paper, we presented a model architecture for learning text similarity based on Siamese recurrent neural networks. With this architecture, we learned a series of embedding spaces, each based on a specific augmentation of the data set used to train the model. The experiments demonstrated that these embedding spaces captured important invariances of the input; the models showed themselves invariant to spelling variation, synonym replacements and superfluous words. The proposed architecture made no assumptions on the input distribution and naturally scales to a large number of classes.

The ability of the system to learn these invariances stems from the contrastive loss function combined with the stack of recurrent layers. Using separate loss functions for similar and dissimilar samples helps the model maintain selectivity while learning invariance over different sources of variability. The experiment shows that the explicit use of prior knowledge to add these sources of invariance to the system was crucial in learning. Without this knowledge extra words and synonyms will negatively affect the performance of the system.

We would like to explore several directions in future work. The possibility space around the proposed network architecture could be explored more fully, for example by incorporating convolutional layers in addition to the recurrent layers, or by investigating a triplet loss function instead of the contrastive loss used in this study.

The application used here is a good use case for the proposed system, but in future work we would also like to explore the behavior of the Siamese recurrent network on standard textual similarity and semantic entailment data sets. In addition, the baseline used in this paper is relatively weak. A comparison to a stronger baseline would serve the further development of the proposed models.

Currently negative samples are selected randomly from the data set. Given the similarity between some groups and the large differences in group sizes, a more advanced selection strategy is likely to yield good results. For example, negative samples could be chosen such that they always emphasize minimal distances between groups. In addition, new sources of variation as well as the sampling ratios between them can be explored.

Systems like the job title taxonomy used in the current study often exhibit a hierarchical structure that we did not exploit or attempt to model in the current study. Future research could attempt to learn a single embedding which would preserve the separations between groups at different levels in the hierarchy. This would enable sophisticated transfer learning based on a rich embedding space that can represent multiple levels of similarities and contrasts simultaneously.

References

Fabio Anselmi, Lorenzo Rosasco, and Tomaso Poggio. 2015. On invariance and selectivity in representation learning. *arXiv preprint arXiv:1503.05938*.

Lorenzo Baraldi, Costantino Grana, and Rita Cucchiara. 2015. A deep siamese network for scene detection in broadcast videos. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 1199–1202. ACM.

Oren Barkan and Noam Koenigstein. 2016. Item2vec: Neural item embedding for collaborative filtering. *arXiv preprint arXiv:1603.04259*.

Filip Boltuzic and Jan Šnajder. 2015. Identifying prominent arguments in online debates using semantic textual similarity. In *Proceedings of the 2nd Workshop on Argumentation Mining*, pages 110–115.

Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2004. Timbl: Tilburg memory-based learner. *Tilburg University*.

Cícero Nogueira dos Santos and Maira Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78.

Alex Graves. 2012. *Supervised sequence labelling*. Springer.

Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition*, pages 84–92. Springer.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*, pages 2042–2050.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Faizan Javed, Matt McNair, Ferosh Jacob, and Meng Zhao. 2014. Towards a job title classification system. In *WSCBD 2014: Webscale Classification: Classifying Big Data from the Web, WSDM Workshop*.

Faizan Javed, Qinlong Luo, Matt McNair, Ferosh Jacob, Meng Zhao, and Tae Seung Kang. 2015. Carotene: A job title classification system for the online recruitment domain. In *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*, pages 286–293. IEEE.

H. Kamper, W. Wang, and K. Livescu. 2016. Deep convolutional acoustic word embeddings using word-pair side information. In *Proceedings ICASSP*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

Emmanuel Malherbe, Mamadou Diaby, Mario Cataldi, Emmanuel Viennet, and Marie-Aude Aufaure. 2014. Field selection for job categorization and recommendation to social network users. In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 588–595. IEEE.

James H. Martin and Daniel Jurafsky. 2000. Speech and language processing. *International Edition*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *Thirtieth AAAI Conference on Artificial Intelligence*.

R. Pascanu, T. Mikolov, and Y. Bengio. 2013. On the difficulty of training recurrent neural networks. *Journal of Machine Learning Research*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Luca Ponzanelli, Andrea Mocci, and Michele Lanza. 2015. Summarizing complex development artifacts by mining heterogeneous data. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 401–405. IEEE Press.

Timothy A. Salthouse. 1986. Perceptual, cognitive, and motoric aspects of transcription typing. *Psychological bulletin*, 99(3):303.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681.

Richard Socher, Eric H. Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y. Ng. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809.

Martijn Spitters, Remko Bonnema, Mihai Rotaru, and Jakub Zavrel. 2010. Bootstrapping information extraction mappings by similarity-based reuse of taxonomies. In *CEUR Workshop Proceedings*, volume 673. Citeseer.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Gabriel Synnaeve and Emmanuel Dupoux. 2016. A temporal coherence loss function for learning unsupervised acoustic embeddings. *Procedia Computer Science*, 81:95–100.

Gabriel Synnaeve, Thomas Schatz, and Emmanuel Dupoux. 2014. Phonetics embedding learning with side information. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 106–111. IEEE.

Roland Thiolliere, Ewan Dunbar, Gabriel Synnaeve, Maarten Versteegh, and Emmanuel Dupoux. 2015. A hybrid dynamic time warping-deep neural network architecture for unsupervised acoustic modeling. In *Proc. Interspeech*.

Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. 2014. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393.

Peilu Wang, Yao Qian, Frank K Soong, Lei He, and Hai Zhao. 2015. A unified tagging solution: Bidirectional lstm recurrent neural network with word embedding. *arXiv preprint arXiv:1511.00215*.

Jason Weston, Antoninipe Bordes, Sumit Chopra, and Tomas Mikolov. 2015. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

Longqi Yang, Yin Cui, Fan Zhang, John P Pollak, Serge Belongie, and Deborah Estrin. 2015. Plateclick: Bootstrapping food preferences through an adaptive visual interface. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 183–192. ACM.

Neil Zeghidour, Gabriel Synnaeve, Maarten Versteegh, and Emmanuel Dupoux. 2015. A deep scattering spectrum - deep siamese network pipeline for unsupervised acoustic modeling. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Will Y Zou, Richard Socher, Daniel M Cer, and Christopher D Manning. 2013. Bilingual word embeddings for phrase-based machine translation. In *EMNLP*, pages 1393–1398.