A neural approach to predicting the surface form of a word given its lemmatized form

Patrice Béchard
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3T 1N8
patrice.bechard@umontreal.ca

Abstract

We study the task of predicting the surface form of a word given its lemmatized form. We try four different approaches to perform the task: a simple benchmark where the lemmatized form is returned as the surface form, a most-frequent approach, where the predicted surface form is the most frequent surface form observed for a given lemmatized word in the training set, and two different models using recurrent neural networks, namely a unidirectional and a bidirectional one. The neural models use a personalized classifier for every lemmatized word, allowing us to significantly reduce the size of the output space for each input. The models were trained on text data from Wikipedia. The neural models achieved test accuracy of 85.53% and 84.99% on the test set, respectively, which was better than the other studied models, even though they were trained on only a small subset of the training set. More training time would allow these models to achieve even better generalization performance.

1 Introduction

In natural language processing (NLP), it is common to do preprocessing on the raw data before using it in various algorithms. One of the most common procedure we do on the data is call *lemmatization*, which consists of converting a word in its most basic form in order to reduce the length of the vocabulary for the whole corpora while still keeping most of the text structure. We present some examples of the effect of lemmatization on words in table 1. We see that the procedure tends to replace capital letters with lowercase ones and changes conjugated verbs to their infinitive form. Most words and punctuation marks are unchanged by lemmatization.

Table 1: Effect of lemmatization on some words.

ORIGINAL WORD	LEMMATIZED FORM
Elizabeth	elizabeth
attended	attend
was	be
degree	degree
1994	1994
.	.

Normally, this task follows simple pre-defined rules. But what if we want to recover the original text from a sequence of lemmatized words? This is precisely the task we want to perform in this report. We will first present the different models we used for this task (§2). We will then present the setup we used to evaluate our models on the task (§3). Finally, we will present the obtained results for each model (§4). All the models are implemented in Python and are available on GitHub : `https://github.com/patricebechard/NLP_H18/tree/master/tp1`

2 Models

We explore the results obtained on three distinct models : a simple model returning the lemmatized word as the original word, which we will use as benchmark (§2.1), another simple and straightforward model using the most frequent original form of a lemmatized word (§2.2), and a model using neural networks to learn the original word forms from the lemmatized ones (§2.3).

2.1 Benchmark

We first build a simple model that we will use as benchmark for the subsequent experiments. This model, when fed the lemmatized word, returns the same word as the original word. We don't expect anything incredible for this model other than setting a lower bound for the accuracy of the other models.

2.2 Most frequent word approach

The second model we look at is a model inspired by Charniak et al. (1993). The task they worked on was Part-of-Speech tagging and they used a model which returns the most probable tag for a given word. In our case, we return the most probable surface form for every lemmatized form of a word.

To do so, we need a reference dictionary which maps a lemmatized word to every possible original word and keeps track of the number of times each original form was related to the lemmatized form. We learn this reference dictionary using a training set and evaluate the performance of the model on an unseen test set at evaluation time.

However, it may happen that the lemmatized word observed at evaluation time has never been seen by the model throughout the training set and is thus absent from the reference dictionary. In this case, we use the same technique as we did with the previous model, returning the lemmatized form as the surface form.

Intuitively, this model should perform better than the previous one. For example, if a word is usually written with capital letters, e.g. a proper noun, the benchmark method will never return the right answer. However, if this same word was seen by the model in the training set and reoccurs at evaluation time, we will be able to classify it properly this time.

2.3 Neural approach

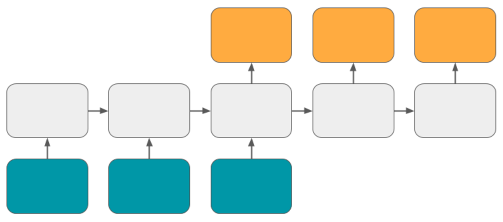
The previous models used simple procedures to predict the right surface form given the lemmatized form of a word. However, they do not learn anything about the semantics and the context of the sentences to make their prediction. Instead, they use a bag of words approach, where the order in which the words appear does not change anything to the prediction. This has some serious limitations when it comes to predicting the correct surface form of a verb, for example, since the context in which the verb is used defines almost entirely how it should be conjugated. We could use a complicated set of rules to define how to conjugate a verb given its close neighbors. This method tends to be pretty complicated to implement, since all the rules have to be explicitly implemented. Furthermore, the window size we define limits the possible long term dependencies between two words if it is too small and may result in a significant waste of memory if it is too big.

An approach that has gained a lot of traction recently in NLP is the use of recurrent neural networks (RNNs) to perform various tasks such as language modelling (Mikolov et al. (2010)), machine translation (Sutskever et al. (2014), Bahdanau et al. (2014)) and speech recognition (Graves et al. (2013)). RNNs tend to be particularly well suited for NLP because of their architecture, which allows the use of inputs of different lengths and takes into account the temporal correlation between the elements contained in the input. We can thus feed a sentence to the neural network without having to define a context window size like we mentioned earlier.

RNNs are known to be difficult to train, mainly because of the so-called *vanishing gradient* problem as well as the *exploding gradient* problem (Pascanu et al. (2013)). They also tend not to learn long-term dependencies between elements of an input (Bengio et al. (1994)). However, by replacing the normal basic building block of a RNN (a neuron) by a more complex computation cell like a long short-term memory (LSTM) cell (Hochreiter and Schmidhuber (1997)) or the more recent gated recurrent unit (GRU) cell (Cho et al. (2014)), it is possible to solve these problems, making the training of the RNN easier. Nonetheless, it still takes a long time training these types of models due to their sequential structure, which makes it difficult to parallelize computations on a GPU.

The input of a neural network has to be represented as a tensor (multidimensional array) of numbers, meaning that we cannot only feed the string of characters in the neural network directly. A naïve way to represent the data is to create a vector of the length of the whole vocabulary (N), where every element of it is 0 and only one element is 1 (one-hot representation). This is really constraining, because it is far from being memory efficient and it does not encode any information about the words or the relationship between them. For example, the words *car* and *boat* represented as these long vectors would be separated by the same distance in the N-dimensional space where these vectors reside as the words *car* and *hockey*, which are fairly more different in terms of meaning. Each word would also be orthogonal in this N-dimensional space, meaning that they are all independent from each other. We can argue that this is not true since both a *car* and a *boat* are transportation methods and should have a certain correlation between them. To fix this situation, we can use *word embeddings* to represent the different words as a smaller vector that encodes much more *meaning* for each word. These new representations are trained in an unsupervised way and help us fix both problems mentioned earlier (Bengio et al. (2003), Mikolov et al. (2013)).

Various types of recurrent neural networks can be used for various types of problems. In our case, we notice that the size of the input and the size of the output are always the same (each word is mapped from its lemmatized form to its surface form). It is thus intuitive to choose a *synced sequence-to-sequence model* instead of the more common encoder-decoder sequence-to-sequence architecture that is often used for tasks such as machine translation (Sutskever et al. (2014)). Figure 1 presents the basic architecture of both types of models, where each blue rectangle represent an input, each orange rectangle represent an output, and each grey rectangle represent a computational unit such as a LSTM cell or a GRU cell.


(a) Encoder-Decoder Seq2Seq architecture

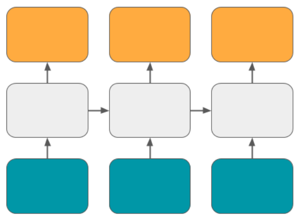

(b) Synced Seq2Seq architecture

Figure 1: Difference between a synced sequence-to-sequence model and a encoder-decoder sequence-to-sequence model.

The output of the RNN is normally mapped to a vector of probability of the size of the vocabulary of the output space. In our case, this represents all the possible original forms of words present in the dataset. However, we know that only a certain number of outputs are possible for a given input. For example, it is impossible that the lemmatized word *cat* be mapped to the word *House*. We thus want the word *cat* only to be mapped to the possible words in [*cat*, *cats*, *Cat*, *Cats*, ...].

This motivates the introduction of *input-wise classifiers* that takes as input the output state of one cell of the RNN and maps it to a vector of n elements, where n is the number of possible words for which the lemmatized word can be translated to. These classifiers are unique to every lemmatized word form. In our case, we implemented them only as one fully connected layer between the output of the RNN and the prediction vector. It is straightforward to add hidden layers to this classifier to make it deeper, even though it can be memory costly. These classifiers are trained using gradient descent just like the rest of the model. This approach is inspired by Andreas et al. (2016), where the architecture of the neural network depends on the input. However, in their case, this architecture was learned by the model while it is hard-coded in ours.

RNNs are normally unidirectional, as seen in figure 1. Recently, bidirectional RNNs have been introduced, which can help the model learn dependencies better in the input Graves et al. (2013). Since both the past and future information are available in our case, it is possible to use this type of architecture. We will train our model using both a unidirectional RNN and a bidirectional RNN. Moreover, Chung et al. (2014) have shown empirically that the use of LSTM and GRU cells in the RNN yield similar results in the task of sequence modelling. For that reason, we will only conduct experiments using GRU cells.

3 Experiments

In this section, we present the dataset that was used to fulfill the wanted task (§3.1) as well as some implementation details for our neural models (§3.2).

3.1 Dataset

The dataset consists of the entire Wikipedia corpus which has been lemmatized. The data files contain two columns of words, where the first column is the surface form of the word and the second column is its lemmatized form, similar to what is shown in table 1. The task is to map from the words on the second column to words on the first column.

It is clever to dig into the dataset before implementing the model. The length of the vocabulary for both the lemmatized words as well as the original words is important to know since the input-wise classifiers are unique to each lemmatized word and map them to the original words.

In our case, these values will vary depending on which dataset we use. We found the length of the vocabulary for both categories using only the training set as well as both the training and the dev set. The results are presented in table 2. We use the smaller dataset (train only) to train the neural models and the augmented one (train + dev) to train the two other ones.

Table 2: Length of vocabulary for the lemmatized words and the original words given the used training dataset.

DATASET	NUMBER OF UNIQUE WORDS (LEMMATIZED FORM)	NUMBER OF UNIQUE WORDS (SURFACE FORM)
TRAIN	3618127	4154606
TRAIN + DEV	4296129	4928238

We see that these numbers are completely unmanageable by a normal neural network, being too expensive in terms of memory. It is simply unfeasible to create 4.3M unique input-wise classifiers.

We can see that some words only have one possible original form, which can help us reduce the number of input-wise classifiers needed. Figure 2 presents the number of lemmatized word forms given their number of possible translations. We see that the number of lemmatized word possessing a certain number of translations decreases exponentially given the number of possible translations. In fact, 90% of the possible lemmatized forms only have one possible translation. Mapping them directly to their unique possible translation could help us reduce the computational burden and the memory needs of the model.

Some words may also be extremely rare in the training corpus, while others happen very frequently. It may thus be wise to focus on the words appearing often in the corpus to obtain a good score while still having a manageable model in terms of memory and training time. Figure 3 presents the number of lemmatized words given the number of times they occur in the training corpus. We see that the number of lemmatized words given their occurence approximately follows a power law (straight line in a log-log plot). This means that a large number of lemmatized words appear really sparsely in the dataset, while a small number of words occur really frequently. However, the decay of number of words given their occurence is slower than an exponential decay, which we observed in figure 2. Further analysis shows us that 90% of the training corpus is composed of approximately 8500 lemmatized words, and that for 95% of the training corpus, this number goes up to approximately 32000.

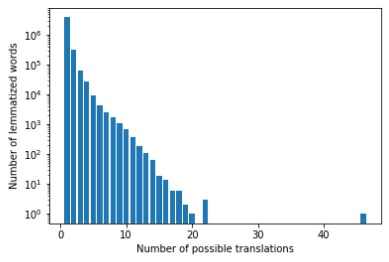


Figure 2: Number of lemmatized words given the number of possible translations. The data comes from the union of the training set and the dev set. Note that the y-axis is presented in log-scale.

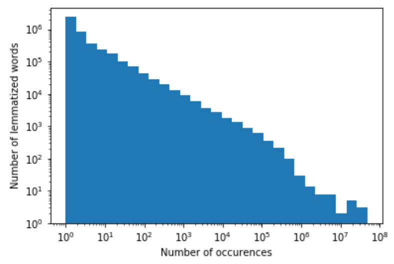
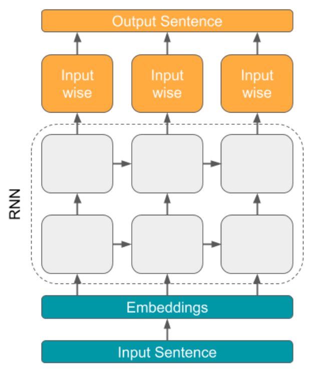


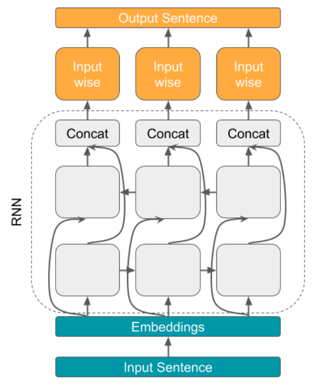
Figure 3: Number of lemmatized words given the number of times they occur in the training corpus. The data comes from the union of the training set and the dev set. Note that both axes are presented in log-scale.

3.2 Implementation details

We decided to implement two different neural models for the task. Their architecture is presented in figure 4. Both models use GRU cells as building blocks.



(a) Model with unidirectional RNN



(b) Model with bidirectional RNN

Figure 4: Neural models used for the task of mapping lemmatized words to their surface form. GRU cells are used for each RNN. The *input-wise* block represents the mapping of the RNN output to the surface form of a given lemmatized word using the procedure described in algorithm 1.

We decided to create an input-wise classifier for the 10000 most frequent words in the training set since they composed more than 90% of the dataset. This compromise helped us save precious GPU memory which we used to create a bigger RNN. We trained both a unidirectional RNN and a bidirectional RNN with GRU cells.

The first model had 2 hidden layers as shown in figure 5a. We first fed the input sentence to a pre-trained embedding layer, which was then fed to the RNN. The size of the embedding as well as the size of the hidden state of the RNN were set to 32. Each output vector of the RNN was then classified using the procedure presented in algorithm 1.

The second model had 1 hidden layer, but since it is bidirectional, the memory needed to store it is the same as the previous model. The architecture of the model is shown in figure 5b. Both the size of the embedding and the size of the hidden state of the RNN were set to 32. The output state of each direction of the RNN were concatenated at the output of the RNN and were classified using the same procedure as the first model.

Both models were trained using cross entropy loss and the Adam optimizer with learning rate of 1×10^{-4}. The sentences were fed one by one (batch size of 1). They were trained on a Nvidia

GeForce GTX 1060 6GB GPU. The first model took about 6 hours to train, while the second model took twice the time for twice the number of updates.

Algorithm 1: Prediction procedure

```
Function PredictSurfaceForm(in_sentence, out_rnn)
    predicted_out = [ ];
    for word in in_sentence do
        if word not in lem_vocabulary then
            predicted = word;
        else if only one possible mapping for word then
            predicted = only possible mapping of word;
        else if exists lemma_wise_clf[word] then
            feed associated out_rnn to lemma_wise_clf[word];
            predicted = max(output of lemma_wise_clf[word]);
        else
            predicted = most probable mapping of word;
        end
        predicted_out.append(predicted);
    end
    return predicted_out
```

4 Results

We present the obtained results for every model used in table 3.

Table 3: Accuracy (%) obtained by each model on the different datasets

MODEL	TRAINING SET	DEV SET ACCURACY	TEST SET
BASELINE	65.24	64.78	64.95
MOST-FREQUENT	80.67	80.72	80.13
NEURAL UNIDIRECTIONAL	85.07	86.95	85.53
NEURAL BIDIRECTIONAL	84.60	88.21	84.99

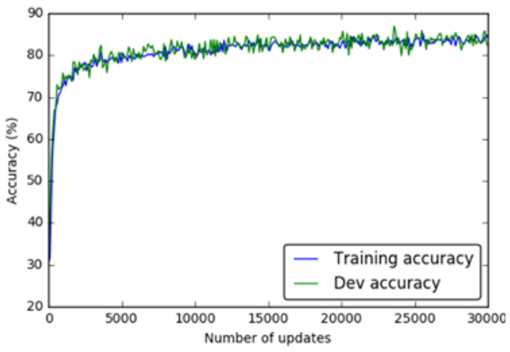
We see that approximately 65% of all words present in the corpus stay unchanged after lemmatization by the accuracy obtained by our baseline model. We also see that the most-frequent word approach yields pretty good results with approximately 80% accuracy.

It is worth mentioning that the most-frequent word approach performs also well on the test set even if it had not seen some of the words present in it before. The slight drop in accuracy ($\sim 0.5\%$) may have something to do with this.

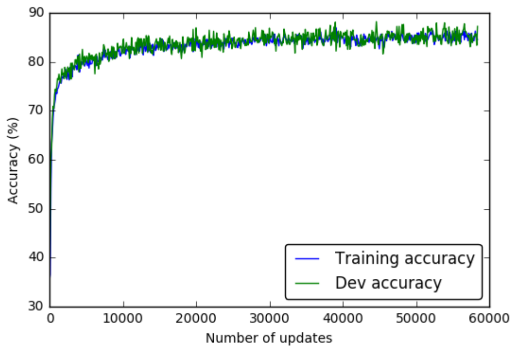
Finally, for the two different architectures of neural networks used, we see that the result are approximately 5% better than what was obtained by the other approaches on each dataset. We used early stopping at the best obtained dev set accuracy to do the final predictions on the test set. The learning curves for both neural models are presented in figure 5. Note that the x axis presents the number of updates of the model instead of the number of epochs. Since a batch size of 1 was used, that means that the models achieved the shown accuracy with only a number of sentences equivalent to the number of updates.

On both figures, we can see that the models learn very rapidly at the beginning and achieve the same accuracy as the most-frequent word approach after about 10000 updates. After that, the accuracy grows really slowly over the number of updates.

We can see that the learning curves are not exactly smooth. This is due to the fact that the accuracy is plotted at each 100 updates for the training set and is thus the mean over these values. The shown accuracy of the dev set is also only an approximation of the accuracy over 100 sentences. This explains why the best obtained scores for the dev set are two to three points higher than the training



(a) Model with unidirectional RNN



(b) Model with bidirectional RNN

Figure 5: Learning curves for both models trained on the task. Note that the learning is presented with respect to the number of updates of the network (not the training epochs).

set. However, the test accuracy was evaluated over the whole test set. This means that the expected generalization performance over unseen items is around 85% for both models.

We do not see an improvement on the performance of the model by using a bidirectional RNN instead of a unidirectional one. Furthermore, the bidirectional model tends to learn much slowly than the unidirectional one, achieving a worse test set error for twice the number of updates. This may be due to the fact that the bidirectional model used was shallower than the unidirectional one. Further experiments would be necessary to conclude on this subject.

Finally, we notice that both models do not tend to overfit the training data and that they still seem to have a small but steady accuracy improvement over the number of updates even at the end of training. In fact, the models are only trained on a tiny subset of the training set and could benefit greatly from seeing more examples by longer training time. This is mostly true for words that possess an input-wise classifier that appear less frequently than the other ones, since these classifiers don't have much chance to adjust their inner weights. However, due to time constraints and a fairly inefficient implementation of the algorithm, we had to stop the training. Ideally, we would need to parallelize the training on the final layer of the network (see algorithm 1), more specifically for the training of the input-wise classifiers. We would also benefit greatly from faster GPUs with more memory.

5 Conclusions

We compared four different models on the task of predicting the surface form of a word given its lemmatized form : a simple benchmark where we return the lemmatized form as the surface form, a most-frequent word approach, which returns the most probable surface form of a lemmatized word, and two different models using recurrent neural networks. The neural models use a custom classifier for every lemmatized word, allowing us to reduce the output space significantly. The models were trained on text data from Wikipedia. The two neural models achieved better accuracy than the other two models after training on only a tiny fraction of the dataset. These models are really costly in terms of training time, which prevented us carry on with the training until we observed overfitting on the training set. Further training would allow us to achieve better generalization error on this task.

References

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M. (1993). Equations for part-of-speech tagging. In *AAAI*, volume 93, pages 784–789.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

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

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.