Estimating User Location in Social Media with Stacked Denoising Auto-encoders

Ji Liu and Diana Inkpen
School of Electrical Engineering and Computer Science
University of Ottawa, Ottawa, Ontario, Canada
rexliu01@gmail.com, Diana.Inkpen@uOttawa.ca

Abstract

Only very few users disclose their physical locations, which may be valuable and useful in applications such as marketing and security monitoring; in order to automatically detect their locations, many approaches have been proposed using various types of information, including the tweets posted by the users. It is not easy to infer the original locations from textual data, because text tends to be noisy, particularly in social media. Recently, deep learning techniques have been shown to reduce the error rate of many machine learning tasks, due to their ability to learn meaningful representations of input data. We investigate the potential of building a deep-learning architecture to infer the location of Twitter users based merely on their tweets. We find that stacked denoising auto-encoders are well suited for this task, with results comparable to state-of-the-art models.

1 Introduction

Many real-world applications require the knowledge of the actual locations of users. For example, online advertisers would like to target potential buyers in particular regions. There are easy ways to obtain user locations, for example, social media service providers allow users to provide their locations, mostly through GPS locating or by manual specification. However, only a small proportion of users actually provide location information. The proportion of users who specify their locations in their profiles is reported to be 14.3% by Abrol et al. (2012); self-reported locations also tend to be unreliable because users can practically type anything they want, such as *In your backyard* or *Wonderland*. When it comes to per-tweet GPS tagging, only 1.2% of all users use this functionality (Dredze et al., 2013). In view of such extreme sparsity, researchers have developed various ways of inferring users' locations using information such as interactions between users, locations declared by users in their social media profiles, users' time zones, the text they generate, etc. The relation between geographical location and language has been studied since the 19th century as a sub-field of sociolinguistics known as dialectology (Petyt, 1980; Chambers, 1998).

In this work, our concern is how to estimate users' locations from the textual data that they generate on social media, and in particular to infer Twitter users' location using the messages they post on their Twitter accounts. For each user, we put together all the tweets written by that user, in order to predict his/her physical location. We focus on predicting users' locations with a deep learning architecture built with denoising auto-encoders proposed first by Vincent et al. (2008), since this approach was not yet applied to this task. The contribution of our work consists in designing models for solving the task and in finding the right parameter values to make the proposed models achieve good results. The first model predicts the U.S. region where the user is located and his/her U.S. state, while the second model predicts the longitude and latitude of the user's location.

2 Related Work

2.1 Location Prediction Using Twitter Data

Many methods have been proposed to predict users' locations based on social network structure (Backstrom et al., 2010), (Jurgens, 2013), (Rout et al., 2013). Here we focus on the methods that predict users' locations based on the social media texts they generate. One of the very first is by Cheng et al. (2010), who first learned the location distribution for each word, then inferred the location of users at U.S. city level according to the words in their tweets. Specifically, they estimated the posterior probability of a user being from a city c given his/her tweets t by computing:

$$P(c|t) = \prod_{w \in t} P(c|w) \times P(w) \tag{1}$$

where w is a word contained in this user's tweets. To improve the initial results, they also used several smoothing techniques such as Laplace smoothing and so-called *data-driven geographic smoothing* and *model-based smoothing*. Their best model managed to make accurate predictions (less than 100 miles away from the actual location) 51% of the time, and the average error distance is 535.564 miles. It is worth noting that the size of the dataset in their work is large, containing 4,124,960 tweets from 130,689 users.

Eisenstein et al. (2010) adopted a topic model approach. They treated tweets as documents generated by two latent variables, i.e., topic and region, and train a system they call *geographic topic model*, which could predict authors' locations based on text alone. Like Cheng et al. (2010), their model also relied on learning regional word distributions. The average distance from the model's prediction to the actual location is 900 kilometres. By comparison, their dataset is much smaller, containing 380,000 tweets from 9,500 users. This dataset is made available and has been used by a number of works.

Roller et al. (2012) used a variant of K-Nearest Neighbours (kNN); they divided the geographic surface of the Earth into *grids* and then constructed a pseudo-document for each grid; a location for a test document was chosen based on the most similar pseudo-document. Another type of model is a variant of Gaussian mixture models (GMMs) proposed by Priedhorsky et al. (2014). Their approach resembles that of Cheng et al. (2010) in constructing location-sensitive n-grams; besides tweets, they also used information such as users' self-reported locations and time zones for prediction.

2.2 Deep Neural Networks

In this section, we present the artificial neural network architectures that will appear in the subsequent sections.

2.2.1 Feedforward Artificial Neural Networks

A feedforward neural network usually has an input layer and an output layer. If the input layer is directly connected to the output layer, such a model is called a *single-layer perceptron*. A more powerful model has several layers between the input layer and the output layer; these intermediate layers are called *hidden layers*; this type of model is known as a *multi-layer perceptron* (MLP). In a perceptron, neurons are interconnected, i.e., each neuron is connected to all neurons in the subsequent layer. Neurons are also associated with activation functions, which transform the output of each neuron; the transformed outputs are the inputs of the subsequent layer. Typical choices of activation functions include the identity function, defined as $y = x$; the hyperbolic tangent, defined as $y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and the logistic sigmoid, defined as $y = \frac{1}{1+e^{-x}}$. To train a MLP, the most commonly used technique is *back-propagation* (Rumelhart et al., 1985). Specifically, the errors in the output layer are back-propagated to preceding layers and are used to update the weights of each layer.

2.2.2 Deep Neural Network Architecture

An artificial neural network (ANN) with multiple hidden layers, also called a Deep Neural Network (DNN), try to mimic the deep architecture of the brain and it is believed to perform better than shallow architectures such as logistic regression models and ANNs without hidden units. The effective training of DNNs is, however, not achieved until the work of Hinton et al. (2006) and Bengio and Lamblin (2007). In both cases, a procedure called *unsupervised pre-training* is carried out before the final supervised fine-tuning. The pre-training significantly decreases error rates of Deep Neural Networks on a number of ML tasks such as object recognition and speech recognition.

The details of DNN's are beyond the scope of this paper; interested readers can refer to the work of Hinton et al. (2006), Bengio and Lamblin (2007), Vincent et al. (2008) and the introduction by Bengio et al. (2013).

2.3 Deep Neural Networks Applied to NLP

Data representation is important for machine learning (Domingos, 2012). Many statistical NLP tasks use hand-crafted features to represent language units such as words and documents; these features are fed as the input to machine learning models. One such example is emotion or sentiment classification which uses external lexicons that contain words with emotion or sentiment prior polarities (Ghazi et al., 2014; Aman and Szpakowicz, 2008; Melville et al., 2009; Li et al., 2009). Despite the usefulness of these hand-crafted features, designing them is time-consuming and requires expertise.

A number of researchers have implemented DNNs in the NLP domain, achieving state-of-the-art performance without having to manually design any features. The most relevant to ours is the work of Glorot et al. (2011), who developed a deep learning architecture that consists of stacked denoising auto-encoders (SDA) and apply it to sentiment classification of Amazon reviews. Their stacked denoising auto-encoders can capture meaningful representations from reviews and outperform state-of-the-art methods; due to the unsupervised nature of the pre-training step, this method also performs domain adaptation well.

In the social media domain, Tang et al. (2013) extracted representations from Microblog text data with Deep Belief Networks (DBNs) and used the learned representations for emotion classification, outperforming representations based on Principal Component Analysis and on Latent Dirichlet Allocation.

Huang and Yates (2010) showed that representation learning also helps domain adaptation of part-of-speech tagging, which is challenging because POS taggers trained on one domain have a hard time dealing with unseen words in another domain. They first learned a representation for each word, then fed the learned word-level representations to the POS tagger; when applied to out-of-domain text, it can reduce the error by 29%.

3 Methods

3.1 Datasets

In order to compare the performance of our system with that of other systems, we choose a publicly available dataset from Eisenstein et al. (2010) [1], which has been used by several other researchers. It includes about 380,000 tweets from 9,500 users from the contiguous United States (i.e., the U.S. excluding Hawaii, Alaska and all off-shore territories). The dataset also provides geographical coordinates of each user. A similar but much larger dataset that we use is from Roller et al. (2012) [2]; it contains 38 million tweets from 449,694 users, all from North America. We regard each user's set of tweets as a training example (labelled with location), i.e., $(x^{(i)}, y^{(i)})$ where $x^{(i)}$ represent all the tweets from the i-th user and $y^{(i)}$ is the location of the i-th user. Meta-data like user's profile and time zone will not be used in our work.

3.2 Our Models

We define our work as follows: first, a classification task puts each user into one geographical region (see Section 4 for details); next, a regression task predicts the most likely location of each user in terms of geographical coordinates, i.e., a pair of real numbers for latitude and longitude. We present one model for each task.

3.2.1 Model 1

The first model consists of three layers of denoising auto-encoders. Each code layer of denoising auto-encoders also serves as a hidden layer of a multiple-layer feedforward neural network. In addition, the top code layer works as the input layer of a logistic regression model whose output layer is a softmax layer.

Softmax Function The softmax function is defined as:

$$softmax_i(\mathbf{z}) = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{J} e^{\mathbf{z}_j}} \quad (2)$$

[1] http://www.ark.cs.cmu.edu/GeoTwitter

[2] https://github.com/utcompling/textgrounder/wiki/RollerEtAl_EMNLP2012

where the numerator z_i is the ith possible input to the softmax function and the denominator is the summation over all possible inputs. The softmax function produces a normalized probability distribution over all possible output labels. This property makes it suitable for multiclass classification tasks. Consequently, a softmax layer has the same number of neurons as the number of possible output labels; the value of each neuron can be interpreted as the probability the corresponding label given the input. Usually, the label with the highest probability is returned as the prediction made by the model.

In our model, mathematically, the probability of a label i given the input and the weights is:

$$\begin{aligned} &P(Y=i|x^N, W^{(N+1)}, b^{(N+1)}) \\ &= softmax_i(W^{(N+1)}x^N + b^{(N+1)}) \\ &= \frac{e^{W_i^{(N+1)}x^N + b_i^{(N+1)}}}{\sum_j e^{W_j^{(N+1)}x^N + b_j^{(N+1)}}} \end{aligned} \tag{3}$$

where $W^{(N+1)}$ is the weight matrix of the logistic regression layer and $b^{(N+1)}$ are its biases. N is the number of hidden layers, in our case $N = 3$. x^N is the output of the code layer of the denoising auto-encoder on top. To calculate the output of i-th hidden layer (i = 1 . . . N), we have:

$$x^i = s(W^{(i)}x^{i-1} + b^{(i)}) \tag{4}$$

where s is the activation function, $W^{(i)}$ and $b^{(i)}$ correspond to the weight matrix and biases of the i-th hidden layer. x^0 is the raw input generated from text[3], as specified in section 4. We return the label that maximizes Equation (3) as the prediction, i.e.:

$$i_{predict} = \arg\max_i P(Y=i|x^N, W^{(N+1)}, b^{(N+1)}) \tag{5}$$

We denote this model as SDA-1.

3.2.2 Model 2

In the second model, a multivariate linear regression layer replaces a logistic regression layer on top. This produces two real numbers as output, which can be interpreted as geographical coordinates. Therefore the output corresponds to locations

[3]Explained in Section 3.3

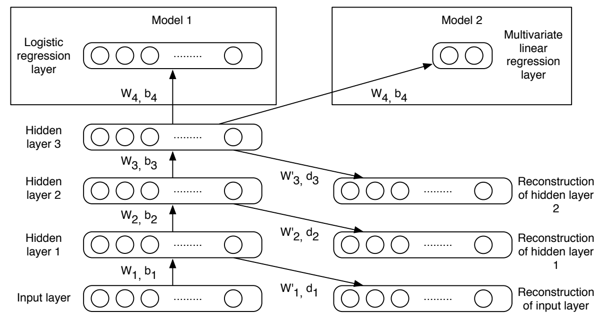
Model 1
Logistic regression layer
W4, b4
Model 2
Multivariate linear regression layer
W4, b4
Hidden layer 3
W3, b3
Hidden layer 2
W2, b2
Hidden layer 1
W1, b1
Input layer
W'3, d3
W'2, d2
W'1, d1
Reconstruction of hidden layer 2
Reconstruction of hidden layer 1
Reconstruction of input layer

Figure 1: Illustration of the two proposed models SDA-1 and SDA-2.

on the surface of Earth. Specifically, the output of model 2 is:

$$y_i = W_i^{(N+1)}x^N + b_i^{(N+1)} \tag{6}$$

where $i \in \{1,2\}$, $W^{(N+1)}$ is the weight matrix of the linear regression layer and $b^{(N+1)}$ are its biases, x^N is the output of the code layer of the denoising auto-encoder on top. The output of i-th hidden layer (i = 1 . . . N) is computed using Equation (4), which is the same as Model 1. The tuple (y_1, y_2) is then the pair of geographical coordinates produced by the model. We denote this model as SDA-2. Figure 1 shows the architecture of both models. They have with three hidden layers. The models differ only in the output layers. The neurons are fully interconnected. A layer and its reconstruction and the next layer together correspond to a denoising auto-encoder. For simplicity, we do not include the corrupted layers in the diagram. Note that models SDA-1 and SDA-2 are not trained simultaneously, nor do they share parameters.

3.3 Input Features

To learn better representations, a basic representation is required to start with. For text data, a reasonable starting representation is achieved with the *Bag-of-N-grams* features (Glorot et al., 2011; Bengio et al., 2013).

The input text of Twitter messages is preprocessed and transformed into a set of Bag-of-N-grams **frequency** feature vectors. We did not use binary feature vectors because we believe the frequency of n-grams is relevant to the task at hand. For example, a user who tweets *Senators* 10 times is more likely to be from Ottawa than another user who tweets it just once. (The latter is more likely to be someone from Montreal who tweets *Senators* simply because the Canadiens happen to be defeated by the Senators that time.) Due to computational limitations, we consider only the 5000 most frequent unigrams, bigrams and trigrams[4]. We tokenized the tweets using the *Twokenizer* tool from Owoputi et al. (2013).

3.4 Statistical Noises for Denoising Auto-encoders

An essential component of a DA is its statistical noise. Following Glorot et al. (2011), the statistical noise we incorporate for the first layer of DA is the masking noise, i.e., each active element has a probability to become inactive. For the remaining layers, we apply Gaussian noise to each of them, i.e., a number independently sampled from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$ is added to each element of the input vector to get the corrupted input vector. Note that the Gaussian distribution has a 0 mean. The standard deviation of the Gaussian distribution σ decides the degree of corruption; we also use the term *corruption level* to refer to σ.

3.5 Loss Functions

3.5.1 Pre-training

In terms of training criteria for unsupervised pre-training, we use the squared error loss function:

$$\ell(x,r) = ||x-r||^2 \tag{7}$$

where x is the original input, r is the reconstruction. The squared error loss function is a convex function, so we are guaranteed to find the global optimum once we find the local optimum.

The pre-training is done by layers, i.e., we first minimize the loss function for the first layer of denoising auto-encoder, then the second, then the third. We define the decoder weight matrix as the transposition of the encoder weight matrix.

3.5.2 Fine-tuning

In the fine-tuning phase, the training criteria differ for model 1 and model 2. It is a common practice to use the *negative log-likelihood* as the loss function of models that produce a probability distribution, which is the case for model 1. The equation for the negative log-likelihood function is:

$$\begin{aligned}&\ell(\theta = \{W,b\},(x,y))\\ &= -\log(P(Y=y|x,W,b))\end{aligned} \tag{8}$$

where $\theta = \{W,b\}$ are the parameters of the model, x is the input and y is the ground truth label. To minimize the loss in Equation (8), the conditional probability $P(Y=y|x,W,b)$ must be maximized, which means the model must learn to make the correct prediction with the highest confidence possible. Training a supervised classifier using the negative log-likelihood loss function can be therefore interpreted as maximizing the likelihood of the probability distribution of labels in the training set.

On the other hand, model 2 produces for every input a location $\hat{y}(\hat{lat},\hat{lon})$, which is associated with the actual location of this user, denoted by $y(lat,lon)$. Given latitudes and longitudes of two locations, their great-circle distance can be computed by first calculating an intermediate value $\Delta\sigma$ with the Haversine formula (Sinnott, 1984):

$$\Delta\sigma = \arctan\left(\frac{\sqrt{(\cos\phi_2\sin\Delta\lambda)^2 + (\cos\phi_1\sin\phi_2 - \sin\phi_1\cos\phi_2\cos\Delta\lambda)^2}}{\sin\phi_1\sin\phi_2 + \cos\phi_1\cos\phi_2\cos\Delta\lambda}\right) \tag{9}$$

Next, calculate the actual distance:

$$d((\phi_1,\lambda_1),(\phi_2,\lambda_2)) = r\Delta\sigma \tag{10}$$

where ϕ_1, λ_1 and ϕ_2, λ_2 are latitudes and longitudes of two locations, $\Delta\lambda = \lambda_1 - \lambda_2$, r is the radius of the Earth. Because d is a continuously differentiable function with respect to ϕ_1 and λ_1 (if we consider (ϕ_1, λ_1) as the predicted location, then (ϕ_2, λ_2) is the actual location), and minimizing d is exactly what model 2 is designed to do, we define the loss function of model 2 as the great-circle distance between the estimated location and the actual location:

$$\begin{aligned}&\ell(\theta = \{W,b\},(x,y))\\ &= d(Wx+b,y)\end{aligned} \tag{11}$$

where $\theta = \{W,b\}$ are the parameters of the model, x is the input and y is the actual location. [5]

[4]Not all of these 5000 n-grams are necessarily good location indicators, we don't manually distinguish them; a machine learning model after training should be able to do so.

[5]Alternatively, we also tried the loss function defined as the

Now that we have defined the loss functions for both models, we can train them with back-propagation (Rumelhart et al., 1985) and Stochastic Gradient Descent (SGD).

4 Experiment

4.1 Metrics

We train the stacked denoising auto-encoders to predict the locations of users based on the tweets they post. To evaluate SDA-1, we follow Eisenstein et al. (2010) and define a classification task where each user is classified as from one of the 48 contiguous U.S. states or Washington D.C. The process of retrieving a human-readable address including street, city, state and country from a pair of latitude and longitude is known as *reverse geocoding*. We use MapQuest API [6] to reverse geocode coordinates for each user. We also define a task with only four classes, the West, Midwest, Northeast and South regions, as per the U.S. Census Bureau.[7] The metric for comparison is the classification accuracy defined as the proportion of test examples that are correctly classified. We also implement two baseline models, namely a Naive Bayes classifier and an SVM classifier (with the RBF kernel); both of them take exactly the same input as the stacked denoising auto-encoders.

To evaluate SDA-2, the metric is simply the mean error distance in kilometres from the actual location to the predicted location. Note that this is the distance on the surface of the Earth, also known as the great-circle distance. See Equations (9)-(10) for its computation. In Section 5.2, we applied two additional metrics, which are the median error distance and the percentage of predictions less than 100 miles away from the true locations, to comply with previous work. Similarly, we implement a baseline model which is simply a multivariate linear regression layer on top of the input layer. This baseline model is equivalent to SDA-2 without hidden layers. We denote this model as baseline-MLR. After we have obtained the performance of our models, they will be compared against several existing models from previous work.

average squared error of output numbers, which is equivalent to the average Euclidean distance between the estimated location and the true location; this alternative model did not perform well.

[6] http://www.mapquest.com

[7] `http://www.census.gov/geo/maps-data/maps/pdfs/reference/us_regdiv.pdf`

4.2 Early Stopping

We define our loss functions without regularizing the weights; to prevent overfitting, we adopt the early-stopping technique (Yao et al., 2007); i.e., training stops when the model's performance on the validation set no longer improves. Specifically, we adopt the *patience* approach (Bengio, 2012), which is illustrated in pseudocode:

```
initialization
patience=20, iteration=1;
while iteration <patience do
    update parameters;
    if the performance improves then
        patience := max(patience, iteration*2);
    end
    iteration +=1
end
```

Algorithm 1: Early stopping.

4.3 Splitting the Data

To make the comparisons fair, we split the Eisenstein dataset in the same way as Eisenstein et al. (2010) did, i.e., 60% for training, 20% for validation and 20% for testing. The Roller dataset was provided split, i.e., 429,694 users for training, 10,000 users for validation and the rest 10,000 users for testing; this is the split we adopted.

4.4 Tuning Hyper-parameters

One of the drawbacks of DNNs is a large number of hyper-parameters to specify (Bengio, 2012). The activation function we adopt is the sigmoid function $y = \frac{1}{1+e^{-x}}$, which is a typical choice as the non-linear activation function. For the size (the number of neurons) of each hidden layer, usually a larger size indicates better performance but higher computational cost. Since we do not have access to extensive computational power, we set this hyper-parameter to 5000, which is equal to the size of the input layer. As for the corruption level, the masking noise probability for the first layer is 0.3; the

Gaussian noise standard deviation for other layers is 0.25. These two values are chosen because they appear to work well in our experiments based on the validation dataset. The Mini-batch size chosen for stochastic gradient descent is 32, which is a reasonable default suggested by Bengio (2012). For the learning rates, we explore different configurations in the set $\{0.00001, 0.0001, 0.001, 0.01, 0.1\}$ for both pre-learning learning rate and fine-tuning learning rate. Lastly, the pre-training stops after 25 epochs, which usually guarantees the convergence. Fine-tuning stops after 1000 epochs; because of the early stopping technique described in Section 4.2, this number is rarely reached.

4.5 Implementation

Theano (Bergstra et al., 2010) is a scientific computing library written in Python. It is mainly designed for numerical computation. A main feature of Theano is its symbolic representation of mathematical formulas, which allows it to automatically differentiate functions. We train our model with stochastic gradient descent which requires the computation of gradients, either manually or automatically. Since Theano does automatic differentiation, we no longer have to manually differentiate complex functions like Equation (9). We implemented SDA-1, SDA-2[8] and the baseline multivariate linear regression model with Theano.

Scikit-learn (Pedregosa et al., 2011) is a machine learning package written in Python. It includes most standard machine learning algorithms. The two baseline models compared against SDA-1 (Naive Bayes and SVM) are implemented using the Scikit-learn package.

5 Results

5.1 Evaluation on the Eisenstein Dataset

The SDA-1 model yields an accuracy of 61.1% and 34.8%, for region classification and state classification, respectively. The results of all models are shown in Table 1. Among all previous works that use the same dataset, only Eisenstein et al. (2010) report the classification accuracy of their models; to present a comprehensive comparison, all models from their work, not just the best one, are listed. Student's t-tests suggest that the differences between SDA-1 and the baseline models are statistically significant at a 99% level of confidence[9].

It can be seen that our SDA-1 model performs best in both classification tasks. It is surprising to find that the shallow architectures that we implemented, namely SVM and Naive Bayes, perform reasonably well. They both outperform all models in (Eisenstein et al., 2010) in terms of state-wise classification. A possible explanation is that the features we use (frequencies of n-grams with n = 1, 2, 3) are more indicative than theirs (unigram term frequencies).

	Model	Classif. Acc. (%)	
		Region (4-way)	State (49-way)
Eisenstein et al. (2010)	Geo topic model	58	24
	Mixture of unigrams	53	19
	Supervised LDA	39	4
	Text regression	41	4
	kNN	37	2
Our models	**SDA-1**	**61.1**	**34.8**
	Baseline-Naive Bayes	54.8	30.1
	Baseline-SVM	56.4	27.5

Table 1: Classification accuracy for SDA-1 and other models

Table 2 shows the mean error distance for various models trained on the same dataset. The difference between SDA-2 and the baseline model is statistically significant at a level of confidence of 99.9% [10]. Our model has the second best results and performs better than four models from previous work. In addition, the fact that SDA-2 outperforms the baseline model by a large margin shows the advantages of a deep architecture and its ability to capture meaningful and useful abstractions from input data.

5.2 Evaluation on the Roller Dataset

Table 3 compares the results from various models on the Roller dataset. The model by Han et al. (2014), which included extensive feature engineering, outperformed other models. In addition it achieves the

[8]Our code is available at https://github.com/rex911/usrloc

[9]We are unable to conduct t-tests on the Eisenstein models, because of the unavailability of the details of the results produced by these models.

[10]We are unable to conduct t-tests on the other models, because of the unavailability of the details of the results produced by these models.

Model	Mean Error Distance(km)
Eisenstein et al. (2011)	845
SDA-2	**855.9**
Priedhorsky et al. (2014)	870
Roller et al. (2012)	897
Eisenstein et al. (2010)	900
Wing and Baldridge (2011)	967
Baseline-MLR	1268

Table 2: Mean error distance of predictions for SDA-2 and models from previous work.

best results by utilizing about 90% of all 214,000 features; when using the top 3% (6420) features, the Accuracy was 10% [11]. The SDA-2 model, despite the computational limitation, achieved better results than that of Roller et al. (2012) using just 5,000 features.

Model	Mean error (km)	Median error (km)	Acc. %
Roller et al. (2012)	860	463	34.6
Han et al. (2014)	NA	260	45
Han et al. (2014) using top 3% features (6420)	NA	NA	10
SDA-2	733	377	24.2

Table 3: Results from SDA-2 and the best models of previous work; *NA* indicates *Not Available*

6 Conclusion and Future Work

The experimental results show that our SDA-1 model outperformed other empirical models; our SDA-2 model's performance is reasonable. We demonstrate that a DNN is capable of learning representations from raw input data that helps the inference of location of users without having to design any hand-engineered features. The results also show that deep learning models have the potential of being applied to solve real business problems that require location detection, in addition to their recent success in natural language processing tasks and to their well-established success in computer vision and speech recognition.

We should point out the comparisons in Section 5 are approximate because our models use unigram, bigram and trigrams, while some of the models with compared with use only unigram; instead, our models use a smaller number of features, especially compared to the model of Han et al. (2014).

We believe a better model can yet be built. For example, our exploration for hyper-parameters is by no means exhaustive, especially for the mini-batch size and the corruption levels, due to the very high running time required. It would be interesting to find out the optimal set of hyper-parameters. More computational capacity also allows the construction of a more powerful DNN. For example, in our models, the hidden layers have a size of 5000, which is equal to the size of input layer; however, a hidden layer larger than the input layer learns better representations (Bengio et al., 2013).

The datasets we use does not have a balanced distribution. Users are densely distributed in the West Coast and most part of the East, whereas very few are located in the middle. Such label imbalance has a negative effect on statistical classifiers, and adversely affects regression models because many target values will never be sampled.

In future work, we plan to collect a dataset uniformly distributed geographically, and the locations do not have to be limited to the contiguous United States. Alternatively, one may notice that the distribution of users is similar to that of the U.S. population, therefore it is possible to use the U.S. census data to offset such a skewed distribution of users. It could also benefit to choose the 5000 features more carefully, instead of simply selecting the most frequent ones. In addition, the input of our system consists only of tweets, because we are mostly interested in recovering users' location from the language they produce; however, real applications require a higher accuracy. To achieve this, we could also incorporate information such as users' profiles, self-declared locations, time zones and interactions with other users. Another type of stacked denoising auto-encoder is one that only does unsupervised pretraining, then the output of the code layer is regarded as input into other classifiers such as SVM (Glorot et al., 2011). It would be interesting to compare the performance of this architecture and that of an SDA with supervised fine-tuning, with respect to our task.

[11]Only this metric was reported by the author in the top 3% features configuration

References

S Abrol, L Khan, and B Thuraisingham. 2012. Tweecalization: Efficient and intelligent location mining in twitter using semi-supervised learning. In *Proceedings of the 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012)*, pages 514–523.

Saima Aman and Stan Szpakowicz. 2008. Using roget's thesaurus for fine-grained emotion recognition. In *Proceedings of IJCNLP*, pages 312–318.

Lars Backstrom, Eric Sun, and Cameron Marlow. 2010. Find me if you can: Improving geographical prediction with social and spatial proximity. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 61–70, New York, NY, USA. ACM.

Yoshua Bengio and Pascal Lamblin. 2007. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19(153).

Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence*, 35(8):1798 – 1828.

Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade*, 7700:437–478.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, volume 4, page 3.

John Kenneth Chambers. 1998. *Dialectology*. Cambridge University Press.

Zhiyuan Cheng, James Caverlee, and Kyumin Lee. 2010. You are where you tweet: a content-based approach to geo-locating Twitter users. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*, volume October 26, pages 759–768, Toronto.

Pedro Domingos. 2012. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Mark Dredze, Michael J Paul, Shane Bergsma, and Hieu Tran. 2013. Carmen: A twitter geolocation system with applications to public health. In *Proceedings of the AAAI Workshop on Expanding the Boundaries of Health Informatics Using AI (HIAI)*.

Jacob Eisenstein, Brendan O'Connor, Noah A Smith, and Eric P Xing. 2010. A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1277–1287, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jacob Eisenstein, Amr Ahmed, and Eric P Xing. 2011. Sparse additive generative models of text. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pages 1041–1048.

Diman Ghazi, Diana Inkpen, and Stan Szpakowicz. 2014. Prior and contextual emotion of words in sentential context. *Computer Speech & Language*, 28(1):76–92, January.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pages 513–520.

Bo Han, Paul Cook, and Timothy Baldwin. 2014. Text-based twitter user geolocation prediction. *Journal of Artifficial Intelligence Research (JAIR)*, 49:451–500.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–54, July.

Fei Huang and Alexander Yates. 2010. Exploring representation-learning approaches to domain adaptation. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 23–30.

David Jurgens. 2013. That's what friends are for: Inferring location in online social media platforms based on social relationships. In *Proceedings of the Seventh International Conference on Weblogs and Social Media, ICWSM 2013, Cambridge, Massachusetts, USA, July 8-11, 2013*.

Tao Li, Yi Zhang, and Vikas Sindhwani. 2009. A non-negative matrix tri-factorization approach to sentiment classification with lexical prior knowledge. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 244–252. Association for Computational Linguistics, August.

Prem Melville, Wojciech Gryc, and Richard D. Lawrence. 2009. Sentiment analysis of blogs by combining lexical knowledge with text classification. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*, page 1275, New York, New York, USA, June. ACM Press.

Olutobi Owoputi, Brendan OConnor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL-HLT*, pages 380–390.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer,

R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

K Malcolm Petyt. 1980. *The study of dialect: An introduction to dialectology*. Andre Deutsch.

Reid Priedhorsky, Aron Culotta, and Sara Y. Del Valle. 2014. Inferring the origin locations of tweets with quantitative confidence. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '14)*, pages 1523–1536, New York, USA, February. ACM Press.

Stephen Roller, Michael Speriosu, Sarat Rallapalli, Benjamin Wing, and Jason Baldridge. 2012. Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1500–1510. Association for Computational Linguistics, July.

Dominic Paul Rout, Kalina Bontcheva, Daniel Preotiuc-Pietro, and Trevor Cohn. 2013. Where's @wally?: a classification approach to geolocating users based on their social ties. In *24th ACM Conference on Hypertext and Social Media (part of ECRC), HT '13, Paris, France - May 02 - 04, 2013*, pages 11–20.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. Learning internal representations by error propagation. Technical report, DTIC Document.

Roger W Sinnott. 1984. Virtues of the haversine. *Sky and Telescope*, 68:158.

Duyu Tang, Bing Qin, Ting Liu, and Zhenghua Li. 2013. Learning Sentence Representation for Emotion Classification on Microblogs. *Natural Language Processing and Chinese Computing*, 400:212–223.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*, pages 1096–1103.

Benjamin P. Wing and Jason Baldridge. 2011. Simple supervised document geolocation with geodesic grids. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL HLT '11)*, pages 955–964. Association for Computational Linguistics, June.

Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. 2007. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315.