Neural Network Models for Lyric Classification

Shaan Bijwadia and Tiwalayo Eisape
Prof. Sergio Alvarez
Boston College, CSCI3341 Artificial Intelligence
9 December 2017

Abstract

This paper presents a method for neural network genre classification of song lyrics using natural language processing techniques. This task appears to be difficult for humans relative to their ability to classify genre based on audio information. Techniques and considerations for data collection are presented along with a survey of strategies for drawing syntactic and semantic features from text. Results demonstrate that accuracies well above 80% can be achieved using these techniques, and that utilizing both syntax and semantics in chorus produces the most robust outcomes.

Introduction

The diversity of American popular music can be most readily identified by the palettes of sound used by artists of different styles to craft their songs. Music is not just instrumentation, however. The content of a song is defined by its lyrical content. This paper focuses its analysis on this latter component of songwriting, and seeks to determine whether a computational agent can correctly identify the genre of a song from its lyrics.

It is not quite obvious that the lyrics of a song should be identifiably correlated with its genre. Song subjects are not unique to genres (love songs are common in both slow ballads and pop songs), and all (English) songs use the same a common dictionary of words. However, the authors expect that the syntactic and semantic differences between genres are robust enough to produce a relatively accurate prediction. The task is also made more difficult by the fact that musical genre is inherently subjective, and the boundaries between categories are not well-defined. Songs may easily be described as two genres simultaneously, or a combination of them. Though this problem cannot be directly solved, we attempt to mitigate the effect of disagreement over classification by collecting all genre information from the same source.

Neural networks were selected as the artificial agent due to their ability to process abstract feature representations and their historically good performance in classification tasks. Such networks require large sets of data and relatively capable hardware to train the network to convergence.

This paper reviews methods from previous work on similar tasks, discusses considerations for the construction of the dataset, estimates human accuracy for the genre categorization task, and assesses the applicability and performance of different network structures. The resulting agent achieves an accuracy which is appreciably greater than random chance and human performance. We suggest that the success of the network at the prediction task is driven in part by the union of syntactic and semantic strategies for feature extraction.

Related Work

Most previous research into musical genre classification has applied the task to audio information. Tzanetakis et. al. developed a features from an audio signal to capture the rhythm and "surface" (incorporating timbre and texture) of the musical information for prediction (Tzanetakis 2001). Existing frameworks for audio processing including Marsyas (Music Analysis, Retrieval, and Synthesis for Audio Signals) can be utilized for more sophisticated processing of audio files (Haggblade, n.d.). Musical information can made more explicit by using attributes of the notes themselves (Mellon, 2014). Humans are impressively good at identifying the musical genre from audio information. A previous study showed 70% self-agreement among participants when played just three seconds of audio, and they performed better-than-chance when given just a 250ms clip (Gjerdingen, 2010). Artificial agents have yet to categorically outperform humans in this task. Below, we find suggestive evidence that genre classification from lyrical information is a much harder problem for humans.

Though the objective of lyric classification is similar to audio classification, the problem of lyric analysis more properly falls into the discipline of natural language processing (NLP). Nadkarni et. al. provide an introduction to the history of NLP. Music lyrics are a special application of NLP, and the structure of lyrics complicates traditional NLP concepts and tools, such as sentence-level grammatical analysis. Mahedero et. al. survey previous work into NLP techniques for lyrics, and suggest strategies to capture the structure and repetition of the source. The task of genre classification has not been directly investigated by previous research, but Laurier et al. note success in classifying the mood of the piece using TF -IDF weighting to process lyric representation, an approach which is adapted in this paper (Laurier 2008).

Auxiliary Data

Since the artificial agent was expected to act intelligently in its genre classification of lyrics, data was collected to estimate the accuracy of humans when put to the same task. A survey was distributed to friends and family of the authors via Google Forms. Twelve songs were selected randomly from the songs dataset, using a

clustered design to ensure each of six genres appear exactly twice. The full lyrics to each song was given and participants were asked to guess which genre the song was classified as. Because no rigorous analysis would be performed on the results, the survey design sacrificed robustness in favor of simplicity for the benefit of the participants. Only six genres were used, and participants were asked to classify each song as only one genre, despite the fact that songs in the dataset usually had more than one classification. The mean score on the 12 point test was 4.76 songs correct, with a standard deviation of 1.71. The maximum score was 9, and the minimum score was 1. A more statistically principled survey would enable the results of this paper to be compared humans by measuring the number of standard deviations from the survey mean. However, the authors feel that the design quality of the survey and the sample size were not sufficient to warrant direct inference from its results. The survey was created only for a suggestive estimate.

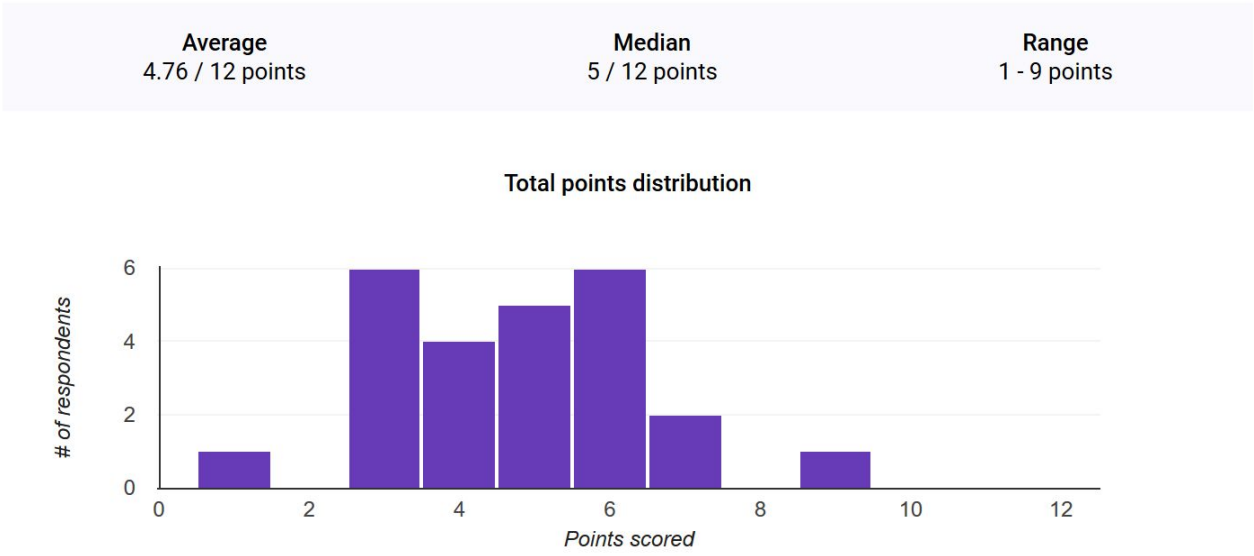


Figure 1 Histogram of scores in the 12-point quiz estimating humans' accuracy in identifying genre from lyrics. Key statistics are noted above the graph. As is discussed in the Results section, the highest-performing network in this study achieved an accuracy of 87%, implying a score of 10.3 on this scale.

Methods

Data collection

Songs were sourced from tracklists from albums sourced from a list of one thousand of the best albums of all time according to RockListMusic.com. Significant experimentation was required to identify an appropriate data source for song lists, genres, and lyrics. The RockListMusic.com dataset was chosen because of a Goldilocks-style decision calculus after exploration of several sources of data.

Quantity

A large bank of data is required to train a neural network to convergence without introducing problems of overfitting. The best and largest compilations of music easily accessible on the internet are “best of all time” lists. More songs can be added by accumulating the tracklists of albums than from rankings of specific songs (the ratio of payoff is about ten songs per album). The larger number of songs indirectly referenced by album lists is critical for the proper training of the neural net, particularly given the attrition rate of songs in their processing by web APIs (below). For this reason, our search was narrowed to lists of the best albums of all time. The most famous of such lists, Rolling Stone’s 500 best albums of all time, resulted in a relatively small number of usable songs, so RockListMusic.com’s list of 1000 albums was preferred.

Distribution

Since the result of neural network training is dependent on the distribution of the training data, care should be taken to ensure wayward distributions do not introduce bias into the network’s predictive capacity. Since our network is intended for practical use in song categorization, our data should reflect the real-world of distribution of songs it might be used to predict. Unfortunately, most “best of all time” lists include a disproportionate amount of rock songs. The data from the RockListMusic.com list was slightly more evenly distributed than Rolling Stone’s 500 best albums of all time, so it was again preferred. However, the data still exhibited considerable bias towards rock. To compensate, an ex-post algorithm was used to construct a final dataset based on a random clustered sample of the resultant songs from the RockListMusic.com’s list. Such a strategy guarantees a satisfying distribution, but reduces the maximum size of the dataset because many rock songs go unused.

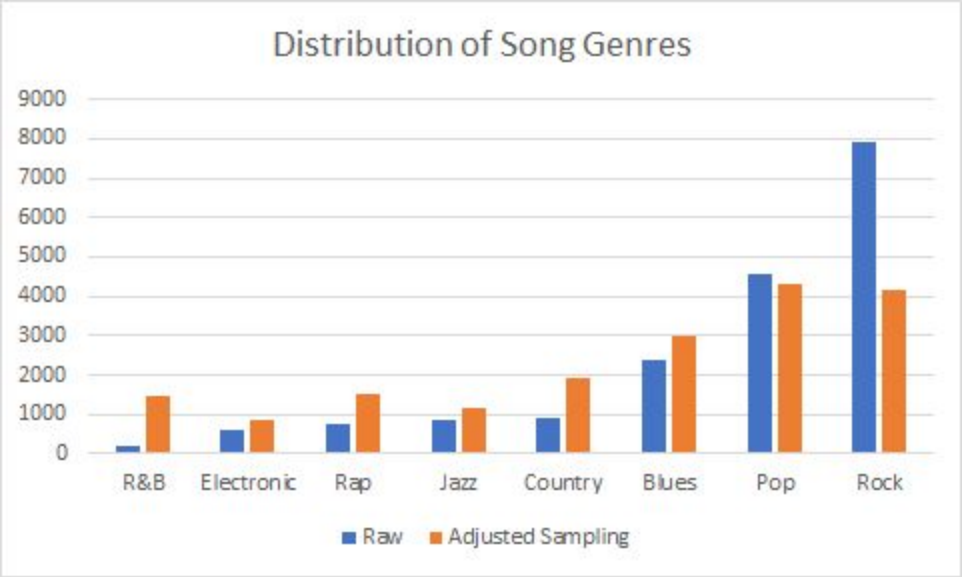


Figure 2 Bar chart of the frequency of each genre. Blue bars represent the data before clustered sampling, and orange bars represent the results of such sampling. The adjusted data is not totally uniform because each song can have more than one genre. This means if a relatively rare genre is sampled, chances are that another, more popular genre also coexists in the information for that song. This method of compensating for genre skewness was devised after collecting our initial results which are included here. The authors suggest additional investigation into the success of networks trained using the above nine genres using clustered sampling.

Content

Our research question implies each song should contain lyrics, and the feature set used in our network requires each song be in English. Other common audio file types, such as podcasts or audiobooks, are similarly disqualified. Though Columbia University's Million Song dataset was large and relatively relatively evenly distributed across genres, the inclusion of foreign-language songs and non-musical entries (poems, plays, etc) rendered it unusable.

Once the set of songs was identified, lyrics and genre information was sourced. This was accomplished using the Genius and Spotify Web APIs, respectively. Song information was passed to each API over the Internet, and the appropriate information was retrieved from the response. Using two APIs required that all songs exist in both databases. Absence of information from either of these sources was the primary source of attrition as the data was prepared for feature extraction.

Genius

Lyrics were sourced from the Genius API, which maintained by Genius, a popular music lyrics aggregation site. Since the metadata available directly through the API does not include the lyrics, a combination approach was used. Song information was passed to the API to find the URL for the song's lyrics page, then a web-scraping algorithm extracted the lyrics from the page's HTML source.

Spotify

Genres were drawn from the Spotify API using the Spotipy python package interface. Because Spotify classifies artists by genre but not individual songs or albums, each song was assigned the genres of the artist that wrote it. This requires the assumption that the genres of a song matches the genres of its primary artist[1].

Spotify uses thousands of genres for classification, and each artist can be tagged with multiple genres. For categorization purposes, a list of genres was adapted from the "parent" genres in an online catalog of music genealogy, in order to capture all relevant sub-genres within the umbrella of our chosen genres. The final genre list consists of blues, country, electronic, pop, jazz, reggae, rock, r&b, and rap. Though ideally each genre should be mutually exclusive of the others, large amounts of overlap exist between the categories in the artists' lists of genres, meaning it is unrealistic to enforce a one genre limit on songs. As a result, songs have multiple genres, leading to the use of a normalized vector for genre encoding (below).

Features

Data representations were chosen in order to extract key syntactic structure as well as semantic tone. Two main groups of features were selected: word frequency and part of speech (POS) N-grams. These features were then modified by applying term frequency-inverse document frequency (TF-IDF) vector representation to both of them.

TF -IDF (shown below) is a weighting system for text analysis that weighs terms by their relative frequencies in the sample corpus, weighting rare terms higher than common ones. Capturing rarity in addition to syntactic occurrence was critical because unique words were fairly few and far between, due to shared topics in genres and repetition within songs

---

[1] This assumption can be avoided by altering the research question to be "Can a neural network agent predict from the lyrics of a song the primary genres of the artist that wrote it?" For example, for a given song, a question might be asked of the network, "Was this written by a rock artist?"

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

Figure 3 The formula for computing term frequency - inverse document frequency for a word in a corpus calculated by dividing a term's frequency in the document by the frequency of the word that is most frequent in the document, to regularize for document size, then multiplying that ratio by a logarithmic transformation of the number of documents divided by the number of documents in the corpus that contained the term.

Grammatical structure was captured by N-grams of POS, which are all possible permutations of POS for each number less than or equal to N. For example, for a corpus where only articles and nouns were used, the POS 2-gram set of this corpus would be [A, N, AN, NA]. This tensor was than feed through the same TF -IDF weighting function to detract weighting from extremely common syntactic combinations, such as article - noun etc.

Both of these features have their strengths and weaknesses. Word count identifies which terms are more frequent in each document, thereby granting some semantic interpretation, one downside is that our models, when using this feature vector, will only be equipped to handle words it has seen before. By structuring feature vectors with a one to one relation between indices and unique words in the corpus, if our models come across new words they will have no space in the feature array for them and will need to drop them, losing some useful data. Furthermore, word count cannot account for metaphor or double entendre there for our models may be be easily confused when artists use these literary devices. POS N-grams, though they do give us an idea of the POS combinations used in each song, capture only a small representation of grammar, syntax in general is much more complicated than this simplistic representation.

Though these are both fairly basic language processing techniques, they capture the essence of the relevant features we identified and yield powerful results when combined.

Models

Neural networks are useful in the interpretation of meaningful features due to their structure, which constructs linear decision boundaries layer by layer. Our neural networks were built in with the tflearn library, which builds upon tensorFlow, optimizing the development time to implement the network structure. Each network was 3 layers deep and scaled depending on the size of there input

layer. The layer ratios were as follows: 1.0 for the input layer, .75 for the first internal layer, .5, and .25. The last layer of each network used a softmax activation function for multiclass classification and resulted in a probability distribution over each of our genres. Most of the model parameters were typical of most implementations, leaving most innovation for in exploration of different combinations of features and model structures.

Some other parameters that remained constant throughout all of our models were:

- The activation function - Each node in our networks used a linear activation function, which essentially fitted the sum of the incoming weighted input to the identity function leaving it unchanged.
- The optimization algorithm - We used ADAM as an optimization standard in our models, ADAM (Adaptive Moment Estimation), has become an industry standard and is included in the tflearn presets, this algorithm keeps track of different learning rates for each parameter in the network and adapts them separately as learning progresses, leading to much more efficient and accurate results compared to its predecessors (Kingama et al. 2015)
- The loss function - Categorical cross entropy was used as our loss function. As mentioned in class, this function calculates the entropy from a predicted softmax classification to the actual label of each example. This function was minimized with ADAM.

Some model attributes were modified across implementations. Our most standard model is a 3 layer feed forward, fully connected neural network, which had three internal layers along with input and softmax output layers, outlined above.

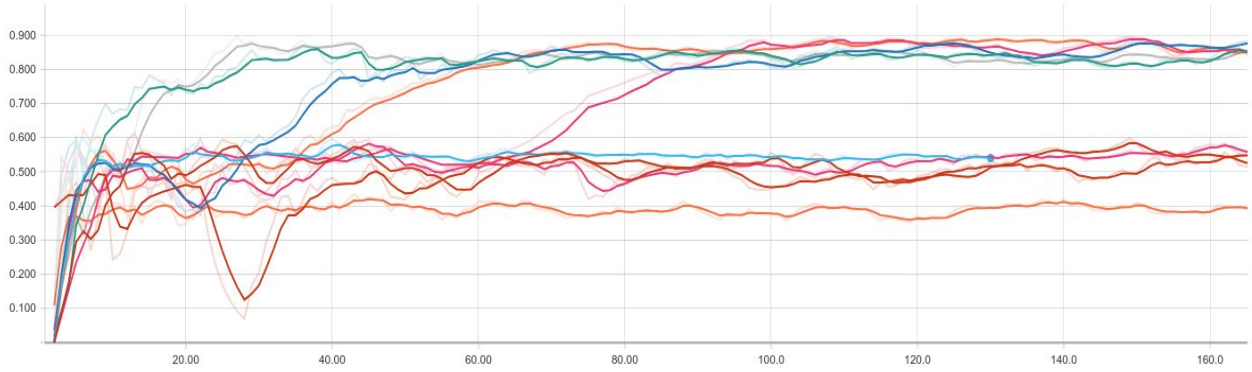
An adaptation to this model was the introduction of a pooling layer. The rationale for this step was to use pooling as a way of condensing more focused information from our rather large sparse feature representation. In the same vein, the last adaption we made to this initial feed-forward model was the use of two autoencoders to tease out concise representations of our original feature sets.

The last model was an adaption of a convolutional neural network for language classification modeled after the experiments of (Kim, 2014). Our hypothesis for this model was that it could work with POS N-grams as they were our only feature with an appreciable structure. This hypothesis turn out to be incorrect and is discussed below.

Each model had specialized training based on how long it took each to converge, though all of our models did reach convergence in 200 epochs or less due to their simple structure. Tflearn uses a training algorithm that selects the last 10% of the provided training samples as a validation set for training, which means that one epoch required ten training cycles through our data.

Results

The results of our experiments appear to confirm that feature representations of both syntax and semantics outperform either feature representation alone. Below is a table with the accuracies of each model against the features used.



	TF-IDF	POS (3-grams)	POS (3-grams) + TF-IDF
Feed Forward w/ pooling	86.3 (+3/-1)	84.5 (+ 5/ -2)	87 (+6/ -3)
Feed Forward	82.6 (+3/ -1)	83.3 (+ 2/ -1)	-- 36
Auto Encoder + Feed Forward	51 (+3 / -2)	53.1 (+2/ -4)	55.2 (+5/ -1)
Convolutional	X	54.8	X

Figure 4 Results table detailing the accuracy of each network structure based on the features used as input. Highest accuracy was achieved by the feed-forward with pooling network trained with both POS N-grams and TF-IDF frequency vectors.

The feed-forward neural network with pooling outperformed every other model with every set of feature vectors, implying that the pooling layers had some effect on accuracy relative to the model without pooling. This makes sense given the large amount of whitespace in our initial sparse vectors. Most of the entries in input vector added no positive value in the next layer, so it appears that pooling out only the highest values emphasized their effect on the next layers. Within this model TF - IDF representation of word count and POS N-gram combinations was the most successful feature, providing an accuracy of 87% on average and was often well above 90%.

The auto encoder was less successful than other models. It is not immediately apparent why this is the case. Some of our poor results with this model may be explained by the dimensions chosen. Our strategy of choosing model dimensions that seemed sensible, without hand picking integer sizes for each layer, may have found its limits with this model. This may have resulted in too much or too little compression in our data to elicit positive results. In future testing, we recommend trying some of many pruning methods for deep neural networks that systematically single out the least utilized nodes in each layer for removal.

Through our convolutional model performed well above chance, we still have reason to believe that our initial hypothesis that POS N-grams were structured enough to take advantage of the network window was incorrect. This may have been because of the window sizes. Since we are not dealing with an

actual image, teasing out windows sizes that were effective while remaining efficient proved to be a challenge and we generally saw an increase in accuracy as the windows were enlarged, though not much.

The reason for CNNs' phenomenal performance with pictures is because adjacent pixels in images have much to do with each other, and this was not the case in our data representations. It turned out that even the permutation relation between indices in our feature vectors was not enough to elicit a classification that was more accurate than our other models.

Moving forward with CNN's, our focus will remain on better data representation that can take full advantage of the network architecture. One way to tackle this issue would be to use word similarity to arrange our feature vector. This would mean choosing a starting word and then progressively building the vector by appending the word in our dictionary that has yet to be chosen and is most similar to the preceding word doing this iteratively until all of the words in our dictionary acuppy a unique space in our vector could result in a representation structured enough to take advantage of a CNN's sliding window.

Conclusion

We demonstrate that in the case of classifying genre from a language processing standpoint features representation of syntax and semantics work better when combined then they do alone. This may not come as a surprise to some given our data but there can still be a tendency of researchers to rely too heavily on semantics or on syntax alone, or to throw data that hasn't been conscientiously represented at models relying too much on neurals neworks ability to tease out features on their own. By remaining too simplistic but still relevant models and focusing on data representations that were innovative and took into account a range of approaches to natural language (syntax and semantics) we showed a little feature engineering can go a very long way.

References

Gjerdingen, R., Perrott, D., "Scanning the Dial: The Rapid Recognition of Music Genres." *Journal of New Research*, 2008. http://faculty-web.at.northwestern.edu/music/gjerdingen/Papers/PubPapers/Scanning.pdf

Haggblade, M., Hong, Y., Kao, K., "Music Genre Classification," n.d.http://cs229.stanford.edu/proj2011/HaggbladeHongKao-MusicGenreClassification.pdf

Laurier, C., Grivolls, Grivolla, J., Herrerra, P., "Multimodal Music Mood Classification using Audio and Lyrics" *Seventh International Conference on Machine Learning and Applications*, 2008. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4725050

Mahedero, J., Martinez, A., Cano, P., Koppenberger, M., Gouyon, F., "Natural language processing of lyrcs" *Proceedings of the 13th annual ACM international conference on Multimedia*, 2005. https://dl.acm.org/citation.cfm?id=1101255

Mellon, R., Spaeth, D., Theis, E., “Genre Classification Using Graph Representations of Music”, 2014. http://cs229.stanford.edu/proj2014/Rachel%20Mellon,%20Dan%20Spaeth,%20Eric%20Theis,%20Genre%20Classification%20Using%20Graph%20Representations%20of%20Music.pdf

Nadkarni, P., Ohno-Machado, L., Chapman, W., “Natural language processing: an introduction.” *Journal of the American Medical Informatics Association*, Volume 18, Issue 5, 2011 https://doi.org/10.1136/amiajnl-2011-000464

Tzanetakis, G., Essl, G., Cook, P., “Automatic Musical Genre Classification Of Audio Signals.” *IEEE Transactions on Speech and Audio Processing*, Vol. 10, Iss. 5, 2002. http://ieeexplore.ieee.org/document/1021072/

Kingma, D., Lei Ba, J., “ADAM: A Method for Stochastic Optimization” 3rd International Conference on Learning Representations https://arxiv.org/pdf/1412.6980.pdf

Kim, Y., “Convolutional Neural Networks for Sentence Classification”, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1746–1751, http://www.aclweb.org/anthology/D14-1181

Packages

François C., “Keras”, *GitHub*, 2016.

Damien, A., et al. “TFLearn”, *GitHub*, 2016. https://github.com/tflearn/tflearn

“Genius API.” Genius. https://docs.genius.com/

Lamere, P. “Spotipy”, *GitHub*. https://github.com/plamere/spotipy

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

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

Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing

with Python. O’Reilly Media Inc.