Neural networks for worker injury autocoding

By Alexander Measure

Each year millions of American workers are injured on the job. A better understanding of the characteristics of these incidents can help us prevent them in the future, but collecting and analyzing the relevant information is challenging. Much of the recorded information exists only in the form of short text narratives written on OSHA logs and worker's compensation documents.

The largest ongoing effort to collect and analyze this information is the Bureau of Labor Statistics' (BLS) Survey of Occupational Injuries and Illnesses (SOII), an annual survey of U.S. establishments that collects approximately 300,000 descriptions of these injuries each year. A typical narrative might resemble the following:

Narrative collected through survey

Job title: **RN**

What was the worker doing? **Helping patient get into wheel chair**

What happened? **Patient slipped and employee tried to catch her**

What was the injury or illness? **Strained lower back**

What was the source? **Patient**

Codes assigned by BLS

Occupation: 29-1141 (registered nurse)

Nature: 1233 (strain)

Part: 322 (lower back)

Event: 7143 (overexertion in catching)

Source: 574 (patient)

Secondary source: None

To facilitate aggregation and analysis, BLS assigns 6 detailed codes to each narrative; an occupation code assigned according to the Standard Occupational Classification system (SOC), and 5 injury codes assigned according to version 2.01 of the Occupational Injuries and Illnesses Classification System (OIICS). Historically BLS has relied entirely on staff to read and code these narratives, but this has begun to change in recent years. Starting with just over a quarter of occupation codes in 2014, BLS is on pace to automatically assign nearly half of all SOII codes for 2016 data.

Portion of SOII codes automatically assigned by survey year
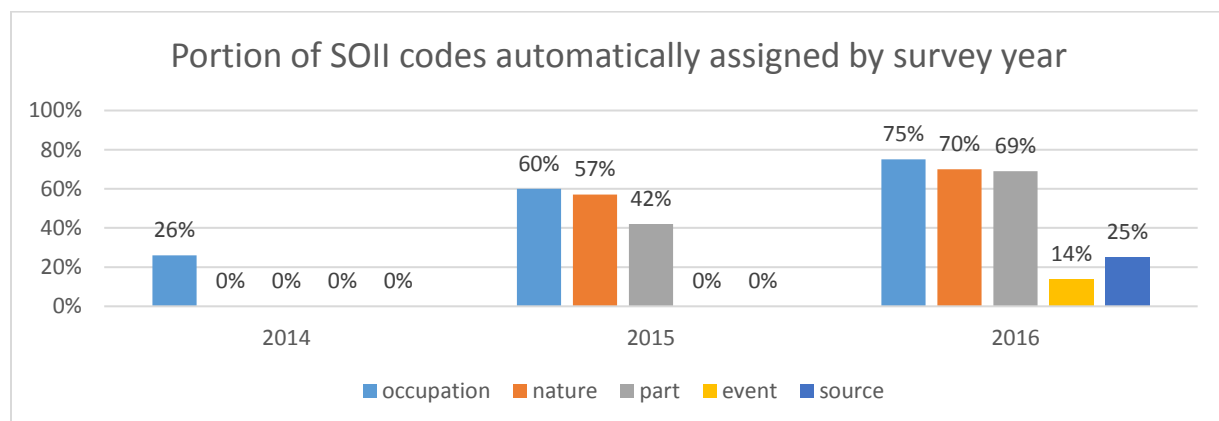
This advance has been made possible by the use of machine learning algorithms that learn from previously coded data. Although previous research suggests we can use these algorithms to automatically assign all SOII codes at near or better than human accuracy (Measure, 2014), it also shows that autocoders still make plenty of errors. Motivated by this, and by recent advances in machine learning, we briefly review the limitations of SOII's existing autocoders, and then introduce a new neural network autocoder that demonstrates substantial improvements.

Background

The autocoders BLS currently uses for SOII coding consist of 5 regularized multinomial logistic regression models, one for each of the 5 main coding tasks; occupation, nature, part, event, and source. Each was created using the free and open source scikit-learn library (Pedregosa, 2011) with bindings to Liblinear (Rong-En Fan, 2008) and is closely related to the logistic regression models described in (Measure, 2014), with only minor tweaks to the input features (see Appendix A for a complete description). Mathematically each is described by:

$$y = f(Wx + b)$$

where y is the n-dimensional vector of predicted code probabilities, n is the number of codes in our classification system, x is an input vector where each position contains a value that indicates some potentially relevant information about a case requiring classification (such as the occurrence of a particular word or pair of words), b is a learnable bias vector, W is a matrix of weights indicating how strongly each element of x contributes to the probability of each possible code represented in y, and $f(z)$ is the multinomial logistic function which constrains each element of y to be between 0 and 1, and the sum of all elements to total 1.

Given a collection of previously coded SOII data, the optimal weight and bias values are approximated by minimizing the L2 regularized cross entropy loss of the model's predictions on the data. Once calculated, cases can then be autocoded by, for a given input, calculating the probability of all possible classifications, and assigning the one with the highest probability. When only partial autocoding is desired, we set a threshold and only assign codes with predicted probabilities exceeding that threshold.

We can divide the errors made by these autocoders into two broad categories. The first, are those due to limitations in the data collected and/or the classification systems being used. Consider, for example, an occupation described simply as "firefighter/paramedic." The SOC defines two separate classifications, one for firefighter, one for paramedic, but here, there is no obvious single answer. As a result different coders often come to different conclusions, at least one of which is wrong according to the official coding guidelines. We have some evidence that this is common. For example, when we asked a select group of regional coding experts to code a random sample of 1000 SOII cases, we found that their average pair-wise agreement rate was only 70.4%. Unfortunately, any ambiguities responsible for this disagreement can only be resolved with changes to the data collection or classification systems.

We focus instead on the second set of errors, those attributable to some limitation of the autocoder. Since machine learning autocoders are a product of both their training data and the algorithm, either is a potential source of error. The training data may be insufficient or may contain too many errors, and the model may be very poor. In our case however, we have more than 1.3 million coded cases, and there is no simple way to find and correct the errors they may contain. Recent research in deep neural

networks however, suggests many potential improvements to the model. This is the direction we explore.

Overview of Deep Neural Networks

In recent years, deep neural networks have been responsible for rapid advances in a wide variety of fields, including natural language processing. A full review of this work is well beyond the scope of this paper, we refer the reader to (Goldberg) and (Ian Goodfellow, 2016) for broader and more detailed overviews. Instead, we briefly introduce the basic concepts upon which we build.

Artificial Neurons

The basic building block of the deep neural network is the artificial neuron, which is typically modeled as some variation of the simple linear model $y = f(w^T x + b)$ where x is an input vector, w is a weight vector, b is a bias vector, and $f(x)$ is a nonlinear function, typically either the logistic function, the hyperbolic tangent, or the rectified linear function defined as:

$$f(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases}$$

Although these neurons can be organized into any pattern, for convenience and computational efficiency they are often organized into layers with the outputs of some layers forming the inputs to others. A single layer of artificial neurons in which each neuron is connected to all outputs from the previous layer is called a densely connected layer. When such a layer also uses the multinomial logistic function as its nonlinear activation it is equivalent to the multinomial logistic regression model which we use for our existing SOII autocoders.

Convolutional Layer

Although fully connected layers are powerful models, they allow all inputs to interact with each other, which can often be ruled out beforehand. By constraining our model to only consider relationships we know to be more likely we can often improve the model's ability to learn the correct relationship. One way to accomplish this is with a convolutional layer, which applies spatial constraints to the interactions of the inputs.

A convolutional layer consists of filters, each typically modeled as $y = f(w^T x + b)$, which are applied to varying subsets of the input. A common approach is to simply apply the filters to each continuous subset of the input. For example, consider an input consisting of the word "manager". A convolutional layer consisting of 2 filters, applied to each contiguous 5 letter subsequence of the word, would produce 6 outputs which might resemble the following:

Step	Input to filter	Filter 1 output	Filter 2 output
1	manag	1.8	-.001
2	anage	-.2	.01
3	nager	-.5	1.5

Because there is often significant redundancy in the output of a convolutional layer, it is common to follow a convolutional layer with a pooling operation which aggregates or discards some of it. We use a

global max pooling layer, which simply discards all but the largest output of each filter. In our example, this would leave us only with the outputs 1.8 and 1.5 for filters 1 and 2, respectively.

LSTM Layer

Another technique, which is particularly popular for modeling sequential structure, is Long Short Term Memory (LSTM). Like the other layers we have discussed, an LSTM layer is composed of smaller computational units. Unlike the others however, the LSTM layer is designed specifically to operate over a sequence of inputs and uses, at each step of the sequence, its output from the previous step as one of its inputs. As a result, the LSTM performs a recurrent computation.

Suppose, for example, that we have a sequence of 4 words "she tripped and fell" and an LSTM layer consisting of a single LSTM unit. Let x_t denote the vector representing the tth word, and let h_t denote the output of our LSTM unit at step t. For each word in the sequence the LSTM will accept as input x_t and the output from the previous step h_{t-1} and produce a new output h_t. In our example, it might produce the following:

Step (t)	Input from sequence (x_t)	Input from previous output (h_{t-1})	Output (h_t)
1	she	0	0.001
2	tripped	0.001	2.400
3	and	2.400	-0.003
4	fell	-0.003	1.980

The output of each LSTM unit is controlled by 4 interconnected artificial neuron-like structures known as the input gate i_t, the forget gate f_t, the cell state c_t, and the output gate o_t. These are calculated as follows, where w and u denote weight vectors, b denotes a bias value, and $\cdot$ denotes the element-wise product.

$$f_t = logistic(w_f \cdot x_t + u_f \cdot h_{t-1} + b_f)$$

$$i_t = logistic(w_i \cdot x_t + u_i \cdot h_{t-1} + b_i)$$

$$o_t = logistic(w_o \cdot x_t + u_o \cdot h_{t-1} + b_o)$$

$$c_t = f_t c_{t-1} + i_t logistic(w_c \cdot x_t + u_c \cdot h_{t-1} + b_c)$$

$$h_t = o_t logistic(c_t)$$

A common modification to the LSTM which often improves performance (and which we use in our work) is to combine the outputs of 2 LSTM layers, each operating on the same input in opposite directions. This is known as a Bidirectional LSTM.

Highway Layer

Although not a recurrent layer, the highway layer none the less draws inspiration from the LSTM. A highway layer is a densely connected layer, with LSTM style gating operations added to encourage the flow of information through networks with many layers (Rupesh Kumar Srivastava, 2015). Let y be the vector of outputs from a highway layer. Each output in this vector is composed from 3 smaller artificial neuron like components, a standard nonlinear transformation $f(w^T x + b)$, a transform gate t, and a carry gate c. For a given input vector x, these are calculated as follows

$$t = logistic(w_t^T x + b_t)$$

$$c = 1 - t$$

$$h = f(w_h^T x + b_h)$$

$$y = ht + xc$$

When the transform gate is fully closed, i.e. 0, the highway layer simply outputs the original input. When it is fully open, it operates as a logistic unit. When it is partially open, it interpolates between the two.

Dropout

Although not a layer with learnable parameters, dropout is another important technique for regularizing neural networks that is sometimes conceptualized as a layer (Rupesh Kumar Srivastava, 2015). During training, dropout is a binomial mask that randomly sets a fraction of the inputs to the specified layer to 0. This has the effect of encouraging the following layer not to become overly reliant on any individual input or combination of inputs. With some modifications, dropout has also been successfully applied to the recurrent connections of the LSTM (Yarin Gal).

Our neural network

The neural network techniques described above provide a rich set of building blocks for modeling a variety of phenomena, but still leave open the question of how best to apply them to our task. In attempting to answer this, we are guided in particular by the following considerations:

1. Our previous work on logistic regression based SOII autocoders found that sub-word information was often useful for combatting the misspellings and unusual abbreviations that frequently occur in SOII data. A successful neural network autocoder must be able to address similar issues. We draw particular inspiration from the work of (Yoon Kim), which showed that character level convolutions followed by highway layers produce effective word representations for translation tasks.
2. SOII autocoding is a text classification task in which the context in which words appears is frequently important. It should be important, therefore, that the model be able to capture this context. Our model is particularly inspired by the LSTM with hierarchical attention proposed by Yang, which is, to our knowledge, the current state of the art for text classification (Zichao Yang).
3. SOII autocoding is inherently multi-output, each case receives 6 codes and there are clear relationships between these codes. Previous work by (Ronan Collobert) has shown the benefits of jointly learning a single model for multiple tasks, we seek to create a neural network that can do this as well.

With these considerations in mind, we propose a single neural network autocoder for all SOII coding tasks, consisting of the following 4 major components:

1. a single word encoder, which, for each word in a SOII text field generates a vector representation of that word based on the characters it contains
2. 3 field encoders, specifically:
   o a narrative encoder, which generates context aware vector representations for each word embedding in a narrative field

- o an occupation encoder, which generates context aware vector representations for each word vector embedding in the "job title" or "other text" fields
- o a company encoder, which generates context aware vector representations for each word vector embedding in a company name or secondary name field
3. a NAICS encoder, which generates a vector representation of a NAICS code
4. 6 task specific output modules, one for each of the coding tasks (occupation, nature, part, event, source, and secondary source). Each module consists of:
   - o an attention layer which weights and aggregates the concatenated outputs of the field encoders to form a vector representation of the document
   - o two highway layers, which accept as input the concatenation of the NAICS embedding and the document embedding
   - o a softmax layer, which produces predicted probabilities for each possible code from the output of the last highway layer

We describe the operation of the network and its components in more detail below.

Input and Preprocessing

The key inputs for SOII coding consist of the following information collected for each case:

Name of field	Description	Example
occupation_text	The worker's job title	Registered nurse
other_occupation_text	An optional field indicating the worker's job category.	Elder care
primary_estab_name	The primary name of the worker's establishment.	ACME hospitals inc.
secondary_estab_name	The secondary name of the worker's establishment.	ACME holding corp
nar_activity	A narrative answering "what was the worker doing before the incident occurred?"	Helping patient get out of bed
nar_event	A narrative answering "What happened?"	The patient slipped
nar_nature	A narrative answering "What was the injury or illness?"	Employee strained lower back
nar_source	A narrative answering "What object or substance directly harmed the employee?"	Patient and floor
naics	The 2012 North American Industry Classification System (NAICS) code for the establishment	622110

Before this data is fed into our neural network, we pre-process it as follows:

- Each text field (all inputs except naics) is separated into a sequence of its component words using the NLTK TreebankTokenizer. Each sequence is then right padded (or, when necessary, truncated) to a fixed length of 15, for occupation and establishment fields, and 60 for the others.
- Each word in each text field is further separated into its component characters. Special <start> and <end> tokens are added to the start and end of the sequence. A special <field indicator> token is also added to the start of the sequence to indicate the field from which the word came. The character sequence is then right padded (or truncated) to a length of 15.

- Each of the 6 digits of the NAICS code is mapped to one-hot 10 dimensional vectors, these are then concatenated to create a 60 dimensional representation of the NAICS code.

Word encoder

The word encoder operates at the lowest level of our network, and converts each sequence of characters representing a word into a 700 dimensional vector. This is accomplished by mapping each character to a 15 dimensional embedding and then applying convolutional layers with input lengths ranging from 1-7 across the input. Each convolutional layer is given a number of filters equal to its input length times 15. The output of each convolutional layer is then max pooled, concatenated with the output of other convolution layers, and fed through a highway network. Our word encoder differs from the small version proposed in (Yoon Kim), in only a few respects. Instead of using character level convolutions of lengths 1-6, we use lengths of 1-7. We also found our encoder performs better when the convolutional layers have no nonlinear activation, and when we follow them with only one highway layer.

After being processed by the word encoder, each text field is now represented by the sequence of vectors representing the words it contains. The occupation_text field, for example, is represented by a 15 x 700 dimension matrix.

Field encoders

Each of our 3 field encoders is a bidirectional LSTM with 512 cells in each direction (1024 total), applied to the sequence of word embeddings generated by applying our word encoder to each word in an input field. Each field encoder operates on the embeddings from the associated fields as indicated in the table below.

Field encoder	Input fields
narrative encoder	nar_activity, nar_event, nar_nature, nar_source
occupation encoder	occupation_text, other_occupation_text
company encoder	primary_estab_name, secondary_estab_name

Sharing the encoders across fields in this way is motivated partly by experimentation and partly by intuition. The nar_activity, nar_event, nar_nature, and nar_source fields frequently contain similar information, often a sentence or two describing some aspect of the injury or a few key words. The occupation_text and other_text fields normally contain only job titles, and the primary_estab_name and secondary_estab_name fields generally contain names of establishments.

Each field encoder also uses Gal (Yarin Gal) style dropout of 50% between the input to hidden, and hidden to hidden connections. The outputs produced by the bidirectional LSTM at each time step are concatenated and fed as output to the next layer.

NAICS encoder

The NAICS encoder accepts as input the 60 dimensional vector representation of the NAICS code. This is then passed through 2 highway layers with rectified linear activations, and finally a linear dense layer to produce a 60 dimensional embedding intended to map similar NAICS codes to similar vector representations.

Attention layer

The attention layer is a multilayer neural network in the style of (Dzmitry Bahdanau), applied to the concatenated outputs of the field encoders. For each step in the concatenated outputs it calculates an attention weight between 0 and 1, such that the total attention across steps sums to 1. The attention weighted average of these outputs becomes the output of the attention layer. Our use of attention was inspired by the work of (Zichao Yang), who found attention layers were useful for text classification. In contrast to their work however, we did not find any benefit from using a hierarchical system of attention that operates at both the word and sentence level. This is likely because SOII narratives are rarely longer than two sentences. The attention layers for OIICS coding operate only over the concatenated outputs of the narrative encoder. The attention layer for SOC coding operates over the concatenated outputs of the occupation, company, and narrative encoders.

Evaluation

To evaluate our approach, we use the approximately 1.3 million SOII cases collected, coded, and deemed usable for estimation between 2011 and 2015. We divide this data into 2 non-overlapping sets, a test set consisting of a random sample of 1000 SOII cases from survey year 2011, and the rest, which we use for training and evaluation.

Although each of the codes in these cases was assigned by trained BLS coders (or reviewed by BLS coders, for those that were automatically assigned), and each is subject to a variety of automatic consistency checks and reviews in the regional and national offices, we know that errors none the less make it through. To determine the best possible codes for the test set we hid the codes that were originally assigned and then distributed the cases to coding experts in regional offices in such a way that each case would be given to 3 different experts, each in a different region. We then instructed the experts to recode the cases from scratch as accurately as possible without referencing previously coded data or discussing code assignments with others. After the cases were recoded, the codes were reviewed by experts in the national office who resolved any coding disagreements. During this process one of our national office experts determined that 2 cases could not receive a reliable SOC code, these cases were excluded from the SOC portion of our evaluation. We refer to the remaining codes as our gold standard, and use it as our measure of truth.

To train our neural network autocoder, we randomly divide the remaining data into a training set of 1,200,000 and a validation set of 100,000. We then train the neural network on the training set using the Adam optimizer, with an initial learning rate of .004 and a 50% decay in learning rate after 2 epochs with no decrease in cross-entropy loss as measured against the validation set. Training was stopped after loss on the validation set failed to decrease for 5 consecutive epochs.

For comparison, we also evaluate the current SOII logistic regression autocoders and the coding produced by the manual process, as it existed in 2011. The logistic regression autocoders were trained and validated on the same 1,300,000 cases used for the neural network. We use the codes originally assigned to our gold standard cases to assess the quality of our manual coding process, which consists of human coding followed by automated consistency checks and review at the state and national level.

We rely on two metrics to assess the quality of coding, accuracy, and macro-F1. Accuracy is calculated as the number of correct code assignments divided by the total number of cases requiring codes. Macro-

F1-score is calculated as the unweighted average of the code specific F1-scores, which can be thought of as code specific accuracies. Each code specific F1-score is calculated as follows:

$$F1\ score = 2 * \frac{precision * recall}{precision + recall}$$

where

$$precision = \frac{number\ of\ times\ code\ i\ was\ correctly\ assigned}{number\ of\ times\ code\ i\ was\ assigned}$$

$$recall = \frac{number\ of\ times\ code\ i\ was\ correctly\ assigned}{number\ of\ times\ code\ i\ should\ have\ been\ assigned}$$

One ambiguity in calculating the F1-score is that precision and recall are sometimes 0, making the F1-score undefined. When this occurs, we adopt the convention used in scikit-learn and simply treat the F1-score as 0.
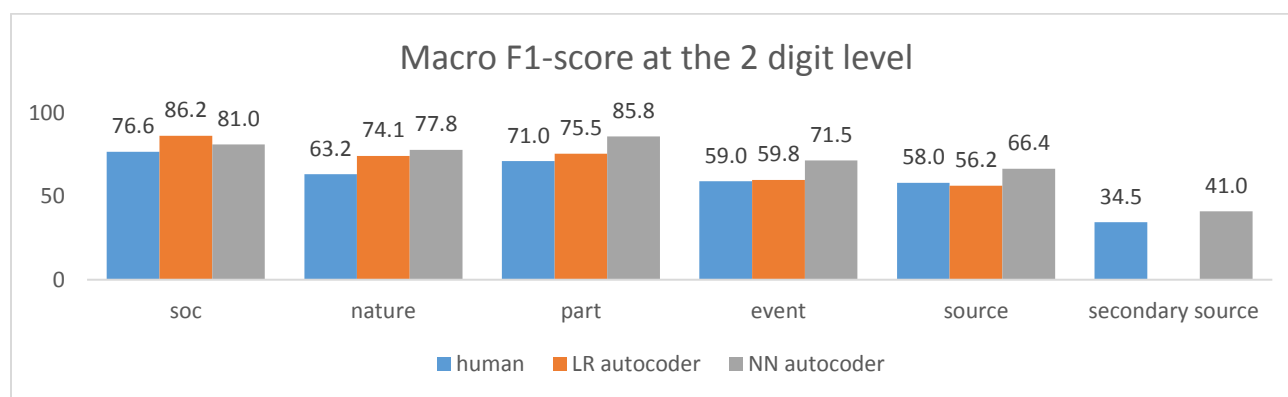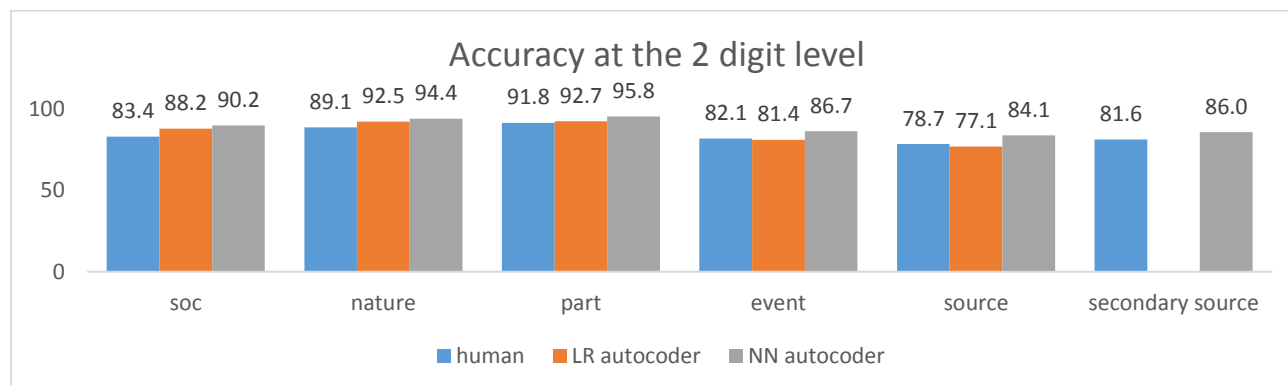
Comparison of the accuracy and macro-F1-scores shows that the neural network autocoder outperforms the alternatives across all coding tasks, and makes an average of 24% fewer errors than our logistic regression autocoders (excluding secondary source, for which we do not currently have a logistic regression autocoder), and an estimated 39% fewer errors than our manual coding process.
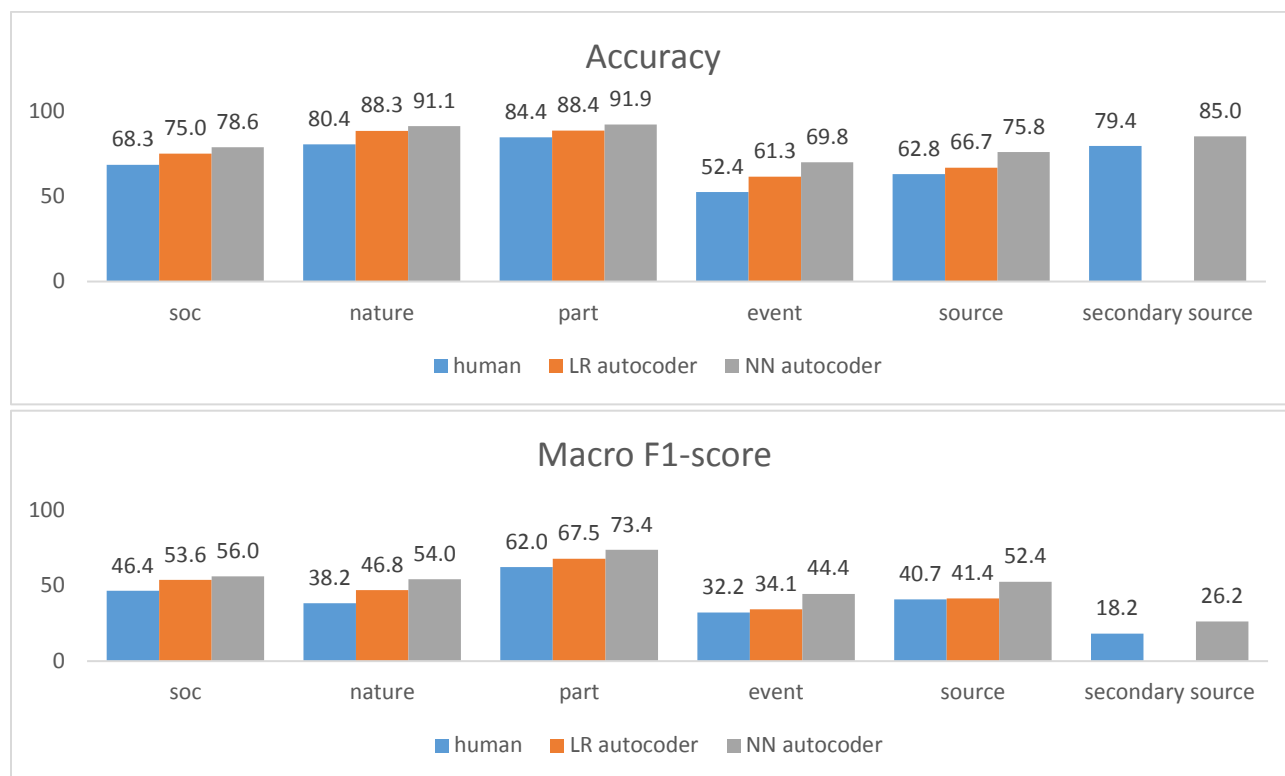
Evaluation at the 2 digit level of coding detail[1], which is a popular level of coding aggregation, shows better performance for the neural network autocoder here as well, except for the macro-F1-score on

---

[1] Evaluation at the 2 digit level of detail was accomplished by truncating the codes to the 2 left most digits.

occupation coding. The macro-F1 scores are quite noisy at the 2 digit level of detail however, as even a few data points can have a large impact on the score, so we urge caution in interpreting them.

Accuracy at the 2 digit level

	soc	nature	part	event	source	secondary source
human	83.4	89.1	91.8	82.1	78.7	81.6
LR autocoder	88.2	92.5	92.7	81.4	77.1	
NN autocoder	90.2	94.4	95.8	86.7	84.1	86.0

Macro F1-score at the 2 digit level

	soc	nature	part	event	source	secondary source
human	76.6	63.2	71.0	59.0	58.0	34.5
LR autocoder	86.2	74.1	75.5	59.8	56.2	
NN autocoder	81.0	77.8	85.8	71.5	66.4	41.0

Model analysis and future challenges

Although improved coding performance is an important goal of our research, SOII autocoding also faces other challenges which we hope to partially address with our use of neural networks. One of these is the transition to new SOC and OIICS classification systems, which is scheduled to occur in within the next few years. When this happens, SOII autocoders will need to be retrained for the new systems but we may not have sufficient training data coded under these new systems, requiring a return to manual coding until such data can be acquired.

Although there are already a variety of techniques for transferring machine learning knowledge across tasks, including some that are algorithm agnostic, such as (III, n.d.), neural networks have unique advantages. Unlike SOII's logistic regression autocoders, for example, which essentially learn a direct mapping from inputs (i.e. words) to code probabilities, neural networks learn a variety of intermediate computations which may be useful for a variety of tasks. To the extent this is the case, the layers responsible for these computations can be readily transferred directly into autocoders intended for the new task, reducing the amount of data needed to learn an effective model. Similar approaches are in fact already widely used in text processing, most extensively in the form of word embeddings (see (Tomas Mikolov, 2013) and (Jeffrey Pennington, 2014)), but there is evidence supporting the transferability of other components as well. For examples, see (Lili Mou, 2016) and (Ji Young Lee, 2017).

Although we cannot fully determine how useful the intermediate computations of our network will be for classification systems that do not yet exist, a closer examination provides some clues and sheds light

on the operation of the neural network more generally. In particular, we examine the behavior of our word encoder, NAICS encoder, and attention mechanisms.

To better understand our NAICS and word encoders, we take the 500 most frequently occurring inputs to these encoders, calculate the vector representations they produce, then project those representations to 2 dimensional space using the TSNE algorithm (van der Maaten & Hinton, 2008), which is designed to preserve the local distance relationships between vectors. Subsets of the results for the job title and NAICS fields, are plotted below. The complete visualizations appear in the Appendix.

Job title embeddings
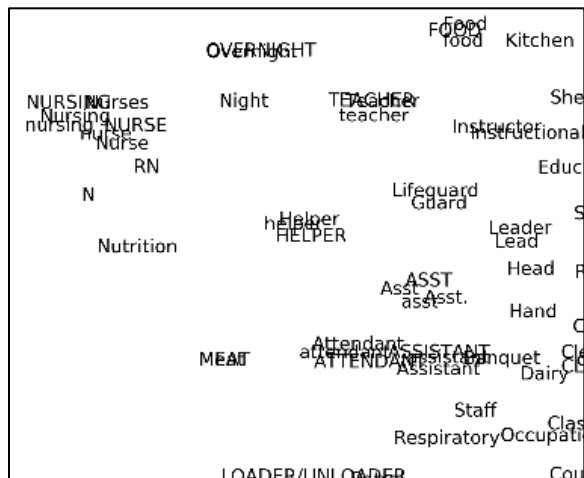
Our visualization of the 500 most frequently occurring words in the job title field indicate that the word encoder has learned to perform a variety of normalizations. For example, in the upper left portion of our visualization we find a cluster of words that all represent some variation of the word "nurse", including "NURSE", "nursing", "Nurses", and "RN", which is a popular acronym for "registered nurse". The embeddings also reflect similarities purely in meaning. In the top right, variations of the words "food" and "kitchen" appear next to each other.

NAICS embeddings
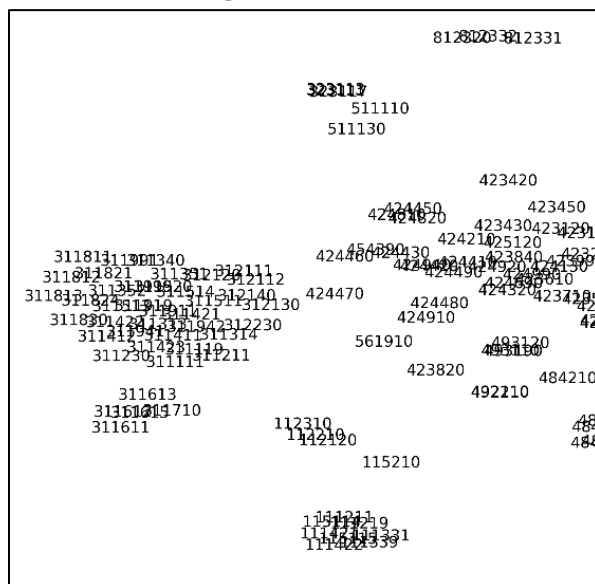
Our visualizations also indicate that the NAICS encoder has learned to capture similarities in meaning. Although the NAICS embeddings are mostly projected into sector level (2 digit) and subsector level (3 digit) clusters, there are interesting and informative exceptions. In the top middle of the visualization, for example, we see a cluster of 4 NAICS codes, 511110 (Newspaper Publishers), 511130 (Book Publishers), 323117 (Commercial Screen Printing), and 323111 (Books Printing). These are clearly related industries, and the NAICS embeddings accurately reflect this, but interestingly, the official NAICS hierarchy does not. In fact, the official hierarchy separates "Publishing" and "Printing" into 2 entirely separate super sectors, the largest possible separation. In this sense, the NAICS encoder has learned a better representation than the official manual, suggesting that similar embedding techniques might be well suited for creating improved hierarchies.

Attention

The attention layers provide another window into the functioning of our neural network. Recall that for each word in an input field, the field encoder produces a vector output. This results in a sequence of vectors, which are then fed to the attention layer. The purpose of the attention layer is to weight and then aggregate (i.e. sum) these vectors so that the following highway layers and softmax layer can accurately predict the correct code. A side effect is that the attention weights provide clues as to what inputs the model is focusing on. In the visualization below, we align the word inputs of our model with the attention weights produced by the "nature code" attention layer and then highlight those words to according to the weight they receive. To make the attention more visible, we normalize the alpha of our yellow highlighting so that 0 is the lowest attention weight assigned and 1 is the highest.

Nature code attention

```
NAR_ACTIVITY
Conducting  beverage  service  _ _ _ _ _ _ - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - -
NAR_EVENT
She  strained  her  back  in  pushing  a  beverage  cart  through  the  aircraft  aisle  .  _ _  _  _ _  _ _  _ _  _
- - - - - - - - - - - - - - - - - - - - - - - - - -
NAR_NATURE
Spondylothesi  Lower  Back  _  _  _  _  _  _  _  _  _  - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - -
NAR_SOURCE
Aircraft  beverage  cart  _  _  _  _  _  _  _  - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - -
```

In this example, our attention layer has placed extra weight on the vectors coinciding with the words, "strained" and "Spondylothesi" [sic], both of which indicate potential injury natures. We find that this holds in general, the part of body attention layer focuses on parts of body, the event layer focuses on words associated with events, and so on. Unexpectedly, we also find that the attention mechanisms place a lot of weight on the vectors produced immediately after the word inputs stop and padding (denoted by "_") begins. We notice this phenomenon in all of our attention layers on all of the narrative fields, but not the occupation and company name fields. This likely reflects some sort of summary computation being performed in the field encoder, which is capable of storing information collected during the reading of a narrative and then releasing that information when it detects that the word sequence has ended. Earlier LSTM research often relied entirely on the summary vector computed by the LSTM. Our work suggests that our model, despite having access to the intermediate steps in the LSTM computation, still does this to some extent.

Conclusion

We have introduced a deep neural network autocoder that makes 24% fewer coding errors than our logistic regression autocoders and 39% fewer errors than our manual coding process. Our analysis suggests the neural network accomplishes this in part by performing a variety of intermediate computations including text and NAICS code normalization and attention, which are likely to be useful for a variety of tasks, and future SOII classification systems. Our work adds to the growing evidence that automatic coding is not merely a tool for easing workload but also a powerful tool for improving the overall quality of coding. This has clear implications for a wide variety of similar coding tasks performed not only in the health community but in the broader economic statistics community.

Acknowledgements

References

Chollet, F. a. (n.d.). *Keras*. Retrieved from https://github.com/fchollet/keras

Dzmitry Bahdanau, K. C. (n.d.). *Neural Machine Translation by Jointly Learning to Align and Translate.* Retrieved from https://arxiv.org/pdf/1409.0473.pdf

Goldberg, Y. (n.d.). *A Primer on Neural Network Models for Natural Langauge Processing.* Retrieved from http://u.cs.biu.ac.il/~yogo/nnlp.pdf

Ian Goodfellow, Y. B. (2016). Deep Learning. Retrieved from http://www.deeplearningbook.org/

III, H. D. (n.d.). Retrieved from Frustratingly Easy Domain Adaptation: http://www.umiacs.umd.edu/~hal/docs/daume07easyadapt.pdf

Jeffrey Pennington, R. S. (2014). GloVe: Global Vectors for Word Representation., (pp. 1532-1543). Retrieved from http://www.aclweb.org/anthology/D14-1162

Ji Young Lee, F. D. (2017, May 17). *Arxiv*. Retrieved from Transfer Learning for Named-Entity Recognition with Neural Networks: https://arxiv.org/pdf/1705.06273v1.pdf

Lili Mou, Z. M. (2016, October 13). *Arxiv*. Retrieved from How Transferable are Neural Networks in NLP Applications?: https://arxiv.org/pdf/1603.06111.pdf

Martin Abadi, A. A. (n.d.). *TensorFlow: Large-scale machine learning on heterogeneous systems*. Retrieved from http://tensorflow.org

Measure, A. (2014). *Automated Coding of Worker Injury Narratives.* Retrieved from https://www.bls.gov/iif/measure_autocoding.pdf

Nitish Srivastava, G. H. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Retrieved from Journal of Machine Learning Research: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Pedregosa, F. V. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2825-2830.

Ronan Collobert, J. W. (n.d.). *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.* Retrieved from https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf

Rong-En Fan, K. W.-J.-R.-J. (2008). LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*.

Rupesh Kumar Srivastava, K. G. (2015). *Highway Networks.* Retrieved from https://arxiv.org/pdf/1505.00387.pdf

Tomas Mikolov, K. C. (2013, September 7). *Arxiv*. Retrieved from Efficient Estimation of Word Representations in Vector Space: https://arxiv.org/pdf/1301.3781.pdf

van der Maaten, L., & Hinton, G. (2008). Visualizing High-Dimensional Data. *Journal of Machine Learning Research*, 2579-2605.

Yarin Gal, Z. G. (n.d.). *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.*
Retrieved from https://arxiv.org/pdf/1512.05287.pdf

Yoon Kim, Y. J. (n.d.). *Character-Aware Neural Language Models.* Retrieved from
https://arxiv.org/pdf/1508.06615.pdf

Zichao Yang, D. Y. (n.d.). *Hierarchical Attention Networks for Document Classification.* Retrieved from
https://www.cs.cmu.edu/~diyiy/docs/naacl16.pdf

Appendix A

Features used in SOII's existing logistic regression autocoders. All features are represented by a position in the feature vector for each unique value they take in the training data. When the feature is present in an input it is represented by a value of 1 in the corresponding position of the feature vector, otherwise it takes a value of 0

Feature name	Description
occ_char4_un	4 letter substrings in words in the occupation_text field
full_company_wrd	Word unigrams in the primary_estab_name and secondary_estab_name fields
occ_wrd	Word unigrams in the occupation_text field
other_wrd	Word unigrams in the other_occupation_text field
cat	1 of 12 job categories
naics	6 digit NAICS code for the establishment
naics2	2 digit NAICS code for the establishment
fips_state_code	2 digit FIPS state code for the establishment
occ_n2	Concatenation of occupation_text unigrams and the 2 digit NAICS code of the establishment
occ_bi	Word bigrams in the occupation_text field
nar	Word unigrams in the nar_activity, nar_event, nar_nature or nar_source fields
nar_bi	Word bigrams in the nar_activity, nar_event, nar_nature, or nar_source fields
nar_nature	Word unigrams in the nar_nature field
nature_char4	4 letter substrings in words in the nar_nature field
nature_conj	Alphabetically sorted concatenation of words appearing in the nar_nature field that are separated by 'and' or '/'.
nature_list	Alphabetically sorted concatenation of words appearing in the nar_nature field that are separated by commas.
nar_source	Word unigrams in the nar_source field.
source_char4	4 character substrings in words in the nar_source field.

Features used in each logistic regression model

| Task specific model	Features
occupation	occ_char4_un, full_company_wrd, occ_wrd, other_wrd, cat, naics, fips_state_code, occ_n2, occ_bi
nature	nar_nature, nature_char4, nar_bi, naics
part	nar_nature, nature_char4, nar, nature_conj, nature_list,
event	nar, nar_bi, naics
source	nar_source, source_char4, nar_bi, naics2, fips_state_code

Appendix B

Job title word embeddings

TSNE visualizations of vectors produced by the word encoder representing frequently occurring words in the occupation_text field. The visualization suggests that the word encoder has learned to produce similar vectors for words and symbols with similar meanings. For example, the encoder represents the symbol "&" and the word "and" with very similar vectors. Other related words also receive similar representations, for example "Associate" and "Aide", "Administrator" and "Secretary", "Manager" and "Director".

Nature narrative word embeddings

TSNE visualization of embeddings of words frequently occurring in the nar_nature field. Words indicating similar natures and parts of body often have similar representations. For example, the embeddings for variations of the words "bruises" and "contusions." Similarly, "arms", "forearm" and "bicep" all receive similar embeddings, as do "pinky", "thumb", "finger", "digit", and "index."
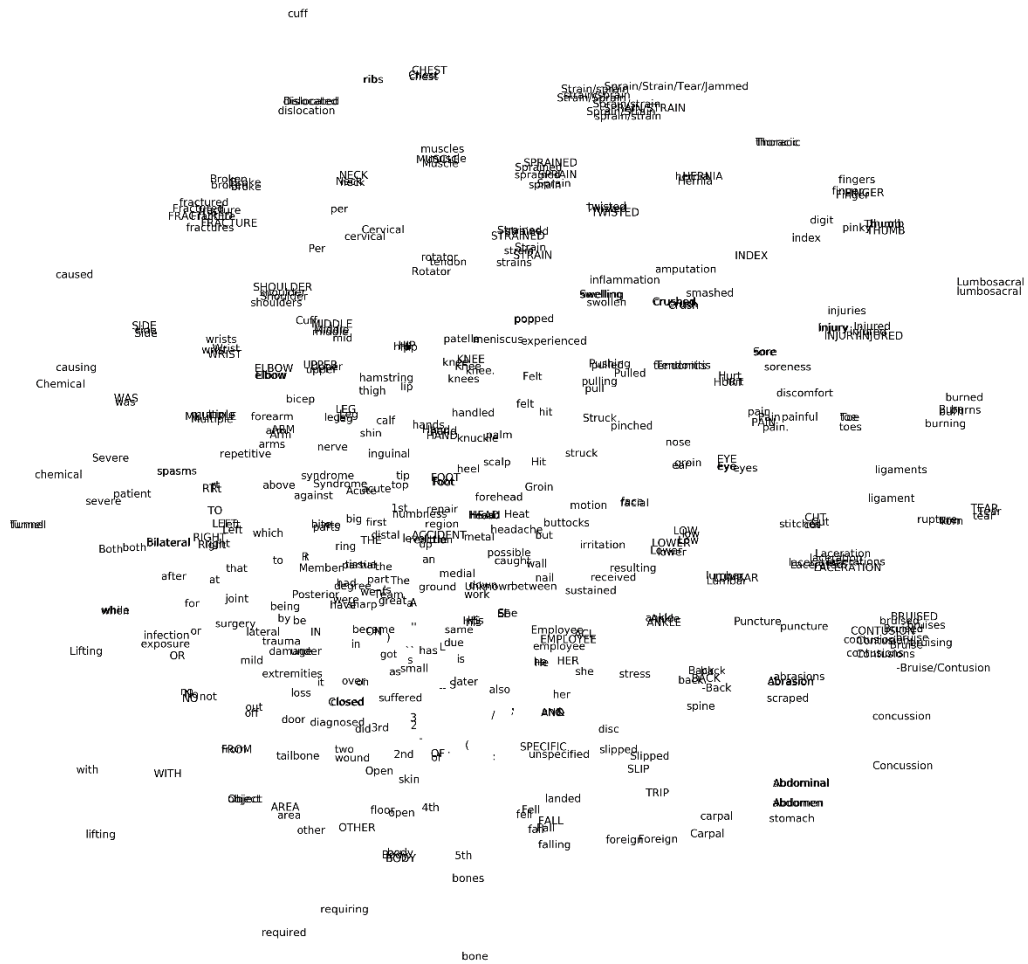
NAICS code embeddings

Our visualization of NAICS embeddings suggests they closely match the hierarchy defined by NAICS, with codes from similar sectors and subsectors mostly grouped together. The embeddings also reflect similarities not captured by the official NAICS 2012 hierarchy, however. For example, the embeddings for 511110 (Newspaper Publishers) and 511130 (Book Publishers), are close to 323113 (Commercial Screen Printing) and 323117 (Books Printing). Similarly, the embeddings for 311 (Animal Product Manufacturing) are near 112 (Animal Production) and 1152 (Support Activities for Animal Production), but 11310 (Logging) is located far away, near 337 (Furniture) and 321 (Wood Product Manufacturing).