This article was published in Advanced Engineering Informatics, 47(January 2021), Xue, X., and Zhang, J., "Part-of-speech tagging of building codes empowered by deep learning and transformational rules." 101235, Copyright Elsevier (2021).
https://doi.org/10.1016/j.aei.2020.101235

Part-of-Speech Tagging of Building Codes Empowered by Deep Learning and Transformational Rules

Xiaorui Xue, S.M.ASCE [1]; Jiansong Zhang, Ph.D., A.M.ASCE [2]

Abstract

Automated building code compliance checking systems were under development for many years. However, the excessive amount of human inputs needed to convert building codes from natural language to computer understandable formats severely limited their range of applicable code requirements. To address that, automated code compliance checking systems need to enable an automated regulatory rules conversion. Accurate Part-of-Speech (POS) tagging of building code texts is crucial to this conversion. Previous experiments showed that the state-of-the-art generic POS taggers do not perform well on building codes. In view of that, the authors are proposing a new POS tagger tailored to building codes. It utilizes deep learning neural network model and error-driven transformational rules. The neural network model contains a pre-trained model and one or more trainable neural layers. The pre-trained model was fine-tuned on Part-of-Speech Tagged Building Codes (PTBC), a POS tagged building codes dataset. The fine-tuning of pre-trained model allows the proposed POS tagger to reach high precision with a small amount of available training data. Error-driven transformational rules were used to boost performance further by fixing errors made by the neural network model in the tagged building code. Through experimental testing, the authors found a well-performing POS

[1] Automation and Intelligent Construction (AutoIC) Lab, School of Construction Management Technology, Purdue University, West Lafayette, IN, 47907, PH (765) 430-2009. email: xue39@purdue.edu.
[2] Automation and Intelligent Construction (AutoIC) Lab, School of Construction Management Technology, Purdue University, West Lafayette, IN, 47907, PH (765) 494-1574; FAX (765) 496-2246. (corresponding author) email: zhan3062@purdue.edu.

tagger for building codes with one bi-directional LSTM trainable layer, utilized BERT_Cased_Base pre-trained model and was trained 50 epochs. This model reached a 91.89% precision without error-driven transformational rules and a 95.11% precision with error-driven transformational rules, which outperformed the 89.82% precision achieved by the state-of-the-art POS taggers.

Author keywords: Automated compliance checking; Automated information extraction; Natural language processing; Part-of-speech tagging; Automated construction management systems; Deep learning.

1. Introduction

Efforts to automate code compliance checking started more than half a century ago when Fenves (1966) developed decision tables to automatically check the design of steel structures [7]. The success of compliance checking decision table inspired more researches in this area. Examples include a computer-aided design (CAD) system for 2D and 3D steel structure called STEEL-3D [8], an expert system for reinforcement concrete design [9], a rule-based application for structure members [10], and a knowledge-based system for multiple building codes [11]. More advanced code compliance checking software was then developed. The Construction and Real Estate Network (CORENET) by Singapore Building Construction Authority was capable of checking 3D industry foundation classes (IFC) data model [12]. The Express Data Manager (EDM) Suite by Jotne EPM Technology allowed code checking on Building Information Modeling (BIM) data [13]. The BCAider by the Commonwealth Scientific and Industrial Research Organisation (CSIRO) in Australia enabled automatic compliance checking against Building Code of Australia (BCA) [14]. The Solibri Model Checker (SMC), a BIM-powered automated code compliance checking system, by Solibri achieved rule-based code compliance checking by user-customized plugins [15]. Patlakas et al. developed a BIM-based system to check code compliance of timber structure design automatically [16]. Fang et al. developed a deep learning-based method to automatically check if a site worker complies to code of their certification [17]. The combination of BIM and automated code compliance checking systems increases the theoretical benefit of BIM in the construction industry. However, according to a survey by Smits et al. (2017), the actual benefit of implementing BIM in construction projects is still limited [18]. The authors suggest that the narrow range of checkable codes of most recent automated code compliance checking tools may limit the actual benefit of BIM. Even for the narrow range of checkable codes, they are usually oversimplified. The oversimplified codes are not enough to support the increased project complexity and creativity of designers and, therefore, could negatively affect the benefit of adopting BIM for users and owners [19].

The narrow range of checkable codes also limit wide applications of these automated code compliance checking systems. Extending the range of checkable building code requirements emerges as an urgent need in the development of automated code compliance checking systems. Natural Language Processing (NLP) powered by Part-of-Speech (POS) tagging has been proposed to automate the building code requirements extraction and, therefore, extend the range of checkable building codes of automated code compliance checking systems and reduce the needed manual efforts in such extraction [20-22]. NLP and deep learning have many applications in the Architecture, Engineering, and Construction industry (AEC). For example, Fang et al. developed a text classification method with deep learning to spot near misses in safety reports [23]. Zhong et al. used a deep learning method to classify building quality problems [24]. Trappey et al. used attention mechanism to generate summary of engineering patents [25]. High performance was achieved but POS tagging error was identified as one major source of error of the whole system. Accurately POS-tagged building codes are desired to support such NLP-based automated building code compliance checking. Existing generic POS taggers, however, can not provide such high accuracy on processing building codes [26].

The authors are therefore proposing a new POS tagger that is tailored to building codes. The intent of the study is to improve the accuracy of POS tagging on building codes. Accurate POS tagging results are needed to support successful code requirements processing for accurate automated code compliance checking. The proposed POS tagger combines neural network model and error-driven transformational rules. Neural network model and error-driven transformational rules together make the proposed POS tagger outperformed the state of the art. The proposed POS tagger reached a 95.11% accuracy, which is higher than the 89.82% achieved by the state of the art.

In practice, this POS tagger plays an important role in those NLP-based automated code compliance checking system frameworks similar to [20] (Figure 1), and in NLP-based automation systems in the AEC domain in general. This research can boost the accuracy of the POS tagging therefore support automated building code compliance checking systems and NLP-based systems in the AEC domain. Accurate POS tagging results of building codes is vital to a high performance of the extraction of engineering knowledge embedded in the building codes. The background automated code compliance checking system framework in Figure 1 contains an automated regulatory information extraction component (which uses a POS tagger) that converts building code requirements to logic clauses, an automated building design information extraction component that extracts building design information from Building Information Models (BIMs), and an automated reasoning component that outputs the code compliance report. The automated regulatory information extraction component can use the proposed POS tagger, which is illustrated in Figure 3. This system is fully automated from the end-user's perspective. The automated building code compliance checking system takes a rule-based approach to extract information from building codes automatically. Although the POS tagger uses neural network model which is probabilistic in training, the developed POS tagger as a result of the training is deterministic. The weights of the neural network are fixed after the training, leading to determinist results when applying the POS tagger. Therefore, with a robust POS tagger and other well-performing components, the NLP-based automated building code compliance checking system has a better chance to detect all noncompliance cases in a building design without intervention from the user. Due to the imperfect (i.e., less than 100%) precision and recall in the state-of-the-art NLP-based building code compliance checking systems, some manual intervention will still be needed to fix errors in the extraction results of embedded engineering knowledge in the building codes. Such manual intervention is expected from the developers, not from end users. In addition, the amount of manual efforts needed to fix automatic extraction errors is minor comparing to those needed in manual extraction. In this paper, the authors propose to boost the performance of NLP-based automated code compliance checking systems by providing more accurate POS tagging results to such systems.

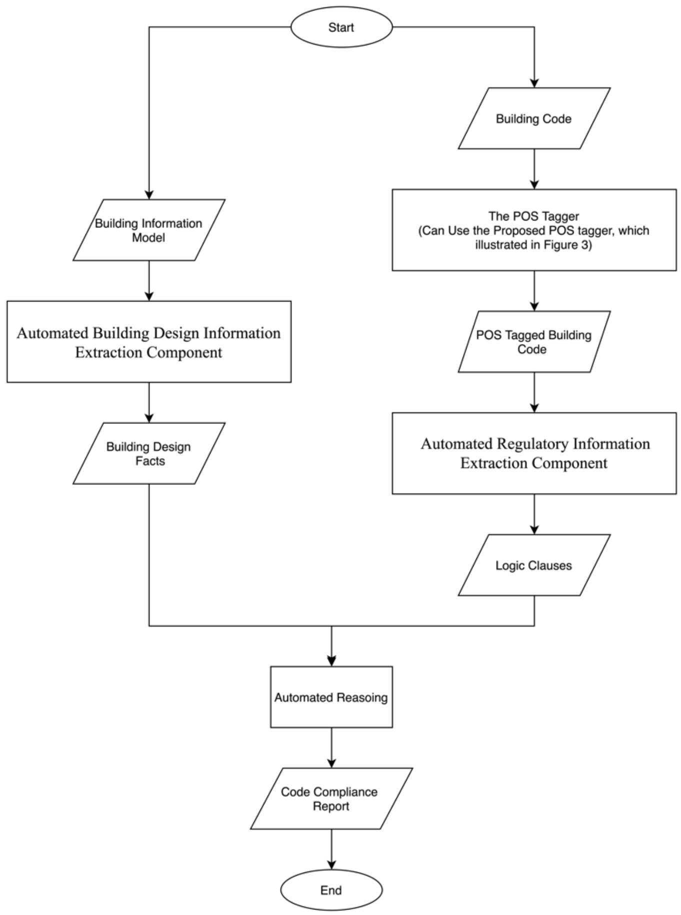

Figure 1. An NLP-based Automated Building Code Compliance Checking System Framework

The remainder of this paper is organized as follows. Section 2 explains technical details of part-of-speech tagging, error-driven transformational rules, recurrent neural network, and computing techniques to avoid overfitting, used in this research. Section 3 describes the proposed POS tagger. Section 4 presents the experiment to test the performance of the proposed POS tagger. Section 5 illustrates the result of the experiment. Finally, Sections 6, 7, 8 present the conclusion, limitation and contribution to the body of knowledge of this research, respectively.

2. Background

2.1 Part-of-Speech

A word's POS category provides its syntactic information in a sentence [39]. In English, there are eight main POS categories: (1) noun, (2) verb, (3) adjective, (4) adverb, (5) pronoun, (6) preposition, (7) conjunction, and (8) interjection. POS taggers are systems that automatically assign POS categories to words according to their contextual information in a sentence [41]. POS taggers have a variety of applications in the AEC domain. For example, Le et al. POS tagged construction contracts to identify missed contract conditions from the perspective of contractors [43]. However, the reliance on manual feature extraction and manual rule generation creates challenges in large scale applications. Hassan and Le used POS tagging to spot contractual requirements from construction contract documents [44]. However, the Support Vector Machines (SVM) algorithm used to identify contractual requirements relies on manual feature engineering and may raise the concern of overfitting. Zhou and El-Gohary utilized POS tagging information to match design requirements in energy codes to their corresponding objects in BIMs [45]. The matching process takes a four-step approach: First, POS tagging information and other contextual information of design requirements and BIM objects are collected; Second, the Word2vec algorithm calculates the vectors of BIM objects and design requirements; Third, vector similarity algorithm calculates the vector similarity between BIM objects and design requirements; Fourth, a match is claimed if the vector similarity between a BIM object and a design requirement is higher than a predefined threshold, which was set arbitrarily to obtain the highest precision and recall empirically. In this four-step approach, errors could accumulate in each step, and the concern of overfitting also presents. Therefore, the authors suggest an end-to-end method that does not rely on manually generated rules or features. Neural network models could meet the above requirements [46].

In this research, the authors proposed an AEC domain specific POS tagger that combines Recurrent Neural Network (RNN), pre-trained models, and error-driven transformational rules. A simple deep learning model without man-made task specific features can outperform most state-of-the-art non-deep learning models even with cherry-picked features, in a wide range of NLP tasks such as part-of-speech tagging, chunking, named entity recognition, and semantic role labeling [57]. For example, Marques and Lopes (2001) utilized a simple feed-forward model to decrease the amount of data needed to train a POS tagger [58]. Yu et al. (2017) used two Convolutional Neural Network (CNN) models to capture morphological information of character-level n-grams and contextual information of word-level n-grams, which outperformed simple feed-forward model [59]. Recent developments in deep learning indicated that RNN is the “to-go” solution for NLP tasks [60]. Pre-trained models were pre-trained on a large body of text with unsupervised tasks, such as, predicting the next word given all previous words and predict if two sentences are from the same article [61]. The use of generally pre-trained models helped boost the performance of domain specific NLP tasks in biology [62], finance, and law [63]. It also reduced the amount of labeled data needed when applying deep learning in domain specific tasks [64].

2.2 Error-driven Transformational Rules

Error-driven transformational rules are introduced to boost POS taggers' accuracy [26, 65]. The rules are designed to transform the machine-generated POS tag of a word to its human-labeled gold standard. When the rule generation algorithm spots a difference between machine-generated POS tags and the human-labeled gold standard, it records the difference as an error and uses the context of the error (i.e., words and POS tags of words around the word) to generate a rule to fix the error. The generation of rules is automated. Rules are reusable once generated. Rules may have the risk to introduce new errors. The rule generation algorithm controls this risk by dropping rules that have a high risk of introducing errors.

2.3 Recurrent Neural Network

Like any machine learning model, neural networks predict categories of given inputs. In the context of POS tagging, neural networks predict POS categories of each word in a given input text, according to the word itself and its context (Figure 2). Neural networks learn a relationship between words and POS tags during their training and use this relationship to predict POS tags of words during their application. Traditional neural networks consider all words in a sentence to be independent from each other and do not consider words surrounding them in this prediction task. In contrast, Recurrent Neural Network (RNN) keeps a vector that represents other words in the sentence (which is called hidden state) and considers them in the prediction task. RNN processes sequential information by taking elements in the sequence one by one while maintaining a representation of all information it has seen so far [60]. RNN is able to process sentences with arbitrary length [66]. The way that RNN processes sequential information gives it the ability to capture semantic meaning of a word based on words before/after it in the sentences [56]. For example, it is able to differentiate the meaning of the word “bank” in the phrase “river bank” and “blood bank”. The sequential nature of RNN makes it widely adopted in many subfields of NLP, such as: (1) information extraction [67, 68], (2) machine translation [69, 70], (3) speech recognition [71, 72], (4) POS tagging [73, 74], and (5) sentiment analysis [75, 76]. There is also an RNN encoder-decoder model which has a high accuracy in sequence-to-sequence tasks [77]. In this structure, the encoder is an RNN model that converts a variable-length sequence to a fixed-length vector representation and the decoder is another RNN model that converts the fixed-length representation to a variable-length sequence. Neural network models are deterministic when applied (i.e., in making predictions). One neural network model makes the same prediction result with the same input.

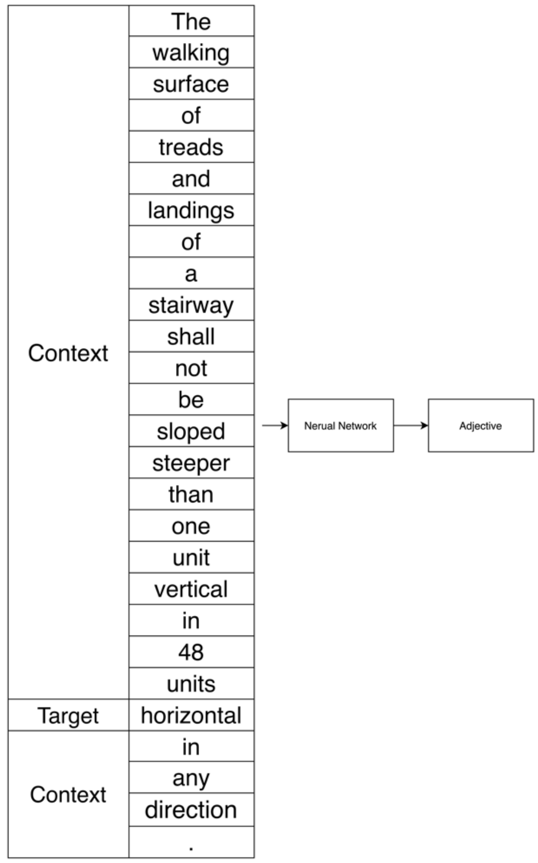

Figure 2. Example Application of a Neural Network POS Tagger

2.3.1. Simple RNN

A simple RNN keeps a hidden state that represents all previous words in the sentence. Therefore, the hidden state allows the simple RNN to take into consideration all words before the target word in POS tagging. A simple RNN contains an input layer *x*, a hidden layer h, and an output layer *y* [78]. The hidden layer has weight W_h and a bias vector b_h. The input layer has a weight W_i. The output layer has a weight W_o and a bias vector b_o. In time step *t* of the training, the input to the RNN is denoted as x_t, the hidden state is denoted as h_t, and the output is denoted as Y_t. The hidden state at the time step *t* ($i.e., h_t$)

is the sum of: (a) the input of current step x_t multiples the weight of the input layer W_i, (b) the hidden state of the last time step h_{t-1} multiplies its weight W_h, and (c) the bias vector of hidden layers b_h, after some non-linear transformation [Eq. (1)].

$$h_t = f(W_i x_t + W_h h_{t-1} + b_h) \tag{1}$$

The output at the time step t ($i.e., Y_t$) is the sum of: the weights of output layer W_o multiples the hidden state at this time step h_t, and the bias vector of output layer b_o [Eq. (2)].

$$Y_t = g(W_o h_t + b_o) \tag{2}$$

In Eqs. (1) and (2), *f* and *g* are activation functions that perform non-linear transformations. Some commonly used activation functions include sigmoid, Tanh, and Rectified Linear Unit (ReLU) [79, 80].

Simple RNN suffers from the vanishing gradient problem [81]. The hidden state of a word is influenced more by words near it than words far away. In other words, simple RNN does not have a "long-term memory". This problem makes simple RNN difficult to train and hard to capture long-term dependencies in a sentence. The long-term dependencies between words are important in POS tagging. Many variations of simple RNN were therefore developed to solve this problem.

2.3.2. Long Short-Term Memory

Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) alleviates the vanishing gradient problem by having a forget gate layer to decide which words to "remember" and which words to "forget". It has a cell state to keep long-term dependencies, so it has "long-term memory". The cell state allows LSTM-RNN to use long-term dependencies in POS tagging. LSTM-RNN [82] has an additional forget gate layer *f* to decide which information to keep or abandon, and a cell state *C* to capture long-term dependencies. The weight of the forget gate layer is W_f and its bias vector is b_f. The cell state has a weight W_C and a bias vector b_C. LSTM-RNN also has an input layer *x*. The input layer has a weight W_i and a bias vector b_i. The output layer has a weight W_o and a bias vector b_o. In time step *t* of the training, the input to the RNN is denoted as x_t, the hidden state is denoted as h_t, the output is denoted as Y_t, and the cell state is denoted as C_t, the value to update is denoted as i_t. Input to the neural network is first fed into the forget gate layer. The forget gate layer generates a vector f_t to represent the amount of information to keep, and f_t is calculated by Eq. (3):

$$f_t = \sigma\left(W_f * [h_{t-1}, x_t] + b_f\right) \tag{3}$$

where σ is the sigmoid function.

Then, the input layer calculates the candidate cell state by Eq. (4) and Eq. (5):

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{4}$$

$$\widetilde{C_t} = \tanh(W_C * [h_{t-1}, x_t] + b_C) \tag{5}$$

Then, the cell state C_t is calculated by Eq. (6):

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{6}$$

After that, the output layer Y_t and hidden state h_t are calculated by Eq. (7) and Eq. (8), respectively:

$$Y_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{7}$$

$$h_t = Y_t * \tanh(C_t) \tag{8}$$

There is also a bi-directional variant of LSTM, which can capture information in a sequence from both directions. Simple RNN and LSTM-RNN have one hidden state that represents all words before the target word. Bi-directional LSTM-RNN additionally has an extra hidden state that represents all words after the target word. Therefore, simple RNN and LSTM RNN predict the POS tag of the target word solely by words before it, whereas bi-directional LSTM RNN predicts POS tag of the target word by the words both before and after it.

2.3.3. Gated Recurrent Unit

Gated Recurrent Unit (GRU) [83] is another way to address the vanishing gradient problem. It does not have a forget gate to control the flow of information, so it can access the entire hidden state. It has an update gate U and a reset gate R. The weight of the update get is W_U, the weight of the reset gate is W_R, and the weight of the output layer is W_o. At time step t, the cell state of the update gate, reset state, and the hidden state are U_t, R_t, and h_t, respectively. GRU is calculated using Eqs. (9), (10), (11), and (12):

$$U_t = \sigma\left(W_U * X_t + W_{U,t-1} * h_{t-1}\right) \tag{9}$$

$$R_t = \sigma\left(W_R * X_t + W_{R,t-1} * h_{t-1}\right) \tag{10}$$

$$h'_t = tanh\left(W_o + R_t * W_{U,t-1} * h_{t-1}\right) \tag{11}$$

$$h_t = U_t * h_{t-1} + (1 - U_t) * h'_t \tag{12}$$

GRU can take long-term dependencies of words into the POS tagging task by accessing hidden states of every words in a sentence. There is also a bi-directional variant of GRU, which can use words both before and after a target word to predict its POS category.

2.3.4. Attention Mechanism

Attention mechanism can capture long-term dependencies with arbitrary lengths by calculating attention scores between all words in two sequences and feed the attention scores to a RNN [84]. Therefore, it does not suffer from the vanishing gradient problem. LSTM RNN and GRU still suffer from the vanishing gradient problem when the dependencies are long enough. The attention mechanism predicts the POS tag of a word with its long-term dependencies. Attention mechanism shares the same encoder-decoder structure with the encoder-decoder RNN. The structure of attention mechanism brings its successful application in many sequence-to-sequence (Seq2Seq) tasks such as: (1) machine translation [85], (2) question-and-answering [86], and (3) text entailment [87]. The attention mechanism allows the decoder to access hidden states of the encoder to track back the input sequence [88]. There are many variants of attention mechanisms. For example, global attention focuses on all words in the input including each target word, while local attention only focuses on words in a certain range [89]. Two-way attention allows bi-directional attention between the source and target [87]. This property of two-way attention makes it successful in non-sequence-to-sequence tasks as well, such as sentiment analysis [90].

2.3.5. Transformer

Transformer has a similar encoder-decoder structure as the attention mechanism, but it does not have an RNN [91]. Transformer, like attention mechanism, can capture dependencies in any length. With fewer parameters than the attention mechanism, it is more resistant to overfitting. Therefore, transformer can make POS taggers more generalizable. The encoder and decoder of the transformer are stacks of multi-head attention layers and feed-forward layers with some add-and-normal layers. The multi-head attention is the concatenation of multiple self-attention matrices. The multi-head attention is used to capture different dependencies in a sentence. The first step to calculate the self-attention Z is to calculate: the Query Q, Key K, and Value V matrices with the embedding matrix X, the weight of Query W_Q, the weight of Key W_k, and the weight of Value W_V [Eqs. (13) to (15)].

$$Q = X * W_Q \tag{13}$$

$$K = X * W_k \tag{14}$$

$$V = X * W_V \tag{15}$$

Then, the self-attention matrix, or one head of the multi-head attention, is calculated by Eq. (16):

$$Z = softmax\left(\frac{Q * K^T}{\sqrt{d_k}}\right) * V \tag{16}$$

where d_k is the dimension of Key.

After that, multiple self-attention matrices are concatenated together to form a multi-head attention matrix Z_{multi} [Eq. (17)]. The multi-head attention is then multiplied to a weight matrix W_o to get a new attention matrix Z_{new} that captures information from all attention heads [Eq. (18)]. W_o is trained with the matrix Z_{multi}.

$$Z_{multi} = [Z_i, \dots Z_n] \tag{17}$$

$$Z_{new} = W_o * Z_{multi} \tag{18}$$

2.3.6. BERT

Bidirectional Encoder Representations from Transformers (BERT) [61] is a language representation model of the transformer. This model was pre-trained on the BooksCorpus [92] and the English Wikipedia data. Through pre-training, BERT introduces knowledge about general English into the POS tagger. Knowledge about general English is helpful to increase the POS tagger's performance on building codes, because these building codes are written in English. BERT is trained to predict masked words in a sentence and decide if the second sentence in a pair of sentences is actually the sentence after the selected sentence in the training text or just a randomly selected sentence. The BERT model achieved the state-of-the-art performance in 11 NLP tasks with fine-tuning. Information of the different available versions of BERT is provided in Table 1. "Large" models have more layers, larger hidden states, more heads, and more parameters than "base" models. The fine-tuning of pre-trained models allows the neural network model to reach high accuracy on a small dataset [93].

Table 1. Available Versions of BERT

Cased	Size	Number of Layers	Size of Hidden State	Number of Heads	Number of Parameters	Comments
Uncased	Large	24	1024	16	340M	Mask the same word.
Cased	Large	24	1024	16	340M	Mask the same word.
Uncased	Base	12	768	12	110M	
Uncased	Large	24	1024	16	340M	
Cased	Base	12	768	12	110M	
Cased	Large	24	1024	16	340M	
Cased	Base	12	768	12	110M	Trained on 104 Languages
Uncased	Base	12	768	12	110M	Trained on 102 Languages
N/A	Base	12	768	12	110M	Trained on Chinese

3. Methodology

To develop a POS tagger tailored to building codes, the authors combined multiple state-of-the-art techniques such as error-driven transformational rules, recurrent neural networks, dropout layers, and pretrained models. At the core, the proposed POS tagger has two main components, a neural network model and a set of error-driven transformational rules. The neural network model initially predicts the POS tag of a word. The error-driven transformational rules fix errors made by the neural network model. The neural network model has a pre-trained model and multiple trainable layers (i.e., bi-directional LSTM-RNN layer, GRU layer, dropout layer, and TimeDistribute layer). The pre-trained model brings the general linguistic knowledge (i.e., English grammar) into the POS tagger. The authors fine-tune the pre-trained model on a dataset of building codes to customize the pre-trained model with AEC domain knowledge. The bi-directional LSTM-RNN layer and GRU layer capture task-specific information (i.e., how building codes were drafted, and construction terminologies). The dropout layer alleviates overfitting. The TimeDistribute layer outputs the result. A POS tagger search strategy was proposed in this research to efficiently search for a well-performing POS tagger configuration.

3.1. POS Tagger Architecture

The architecture of the proposed POS tagger is shown in Figure 3, which illustrates: (1) an overview of the POS tagger components, and (2) how information flows between components. The inputted building codes are firstly tagged by the neural network model and afterwards processed by the error-driven transformational rules to fix errors made by the neural network model.

The neural network model has two parts, a pre-trained model and additional trainable layers. The pre-trained model uses existing models published by other researchers or commercial/non-profit organizations. These were trained on large bodies of corpus. Many widely used pre-trained models can be inserted here such as Open AI GPT-2 [94], BERT [84], and ELMO [95]. This design allows the comparison between different pre-trained models in this context and the selection of the best-performing model. Weights of the pre-trained model were locked, which made them untrainable in the current context. The untrainable nature of the pre-trained models preserves the cross-domain, cross-application and cross-task information they collected in the original training process. On top of the pre-trained models, there are trainable layers. Weights of trainable layers will be updated in the training process, allowing trainable layers to capture the domain-specific, application-specific, and task-specific information in building code POS tagging. The architecture of this model allows substitution and therefore comparison between different types of neural network layers. The error-driven transformational rules are designed to correct errors of a neural network model.

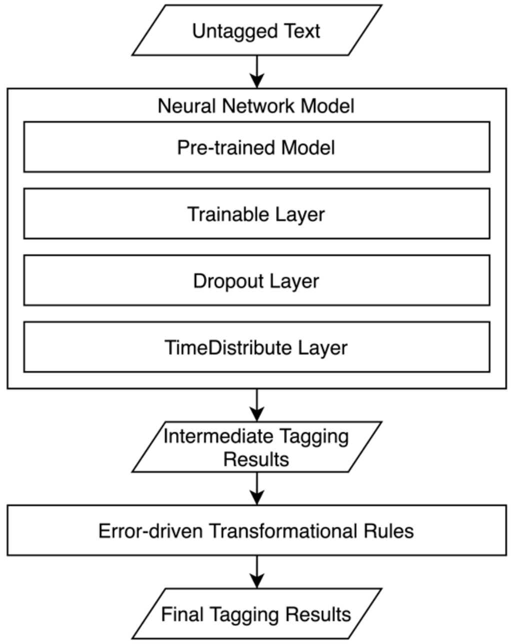

Figure 3. The Architecture of the Proposed POS Tagger

3.2. POS Tagger Search Strategy

Grid search is the most comprehensive way to find the optimal combination of pre-trained models, trainable layers and the number of training epochs by exhaustively searching every possible combination. A global grid search is inefficient, however, because many combinations that are unlikely optimal will be attempted. The authors developed a three-step searching strategy (Figure 4) that can reduce the time to find the optimal combination by ruling out combinations that have low probabilities of being optimal. The first step of this search strategy is finding the best performing combination of epochs of training and trainable layers by attempting all possible combinations of them while replacing the pre-trained model with a random number embedding layer. Because the pre-trained model has been replaced with a random number embedding layer to save training time, grid search is made possible and efficient. An embedding layer converts text strings to vectors of numbers based on the context of the text string and the nature of the embedding layer (e.g., the algorithm used in the layer and the size of the output vector). The pre-trained models will be used to instantiate the embedding layer later in the proposed method. A random number embedding layer is a type of embedding layer that directly maps words to vectors of the random numbers without considering the words' context. It is much smaller and simpler than the pre-trained models and requires significantly less time to train. In this step, the authors intend to find a well performing combination of epochs of training and trainable layers in a short timeframe, so the random number embedding layer is used to help achieve that. In the second step, the random number embedding layer is substituted with different pre-trained models in the locally best-performing combination of number of epochs and trainable layers that was identified in the first step. This step is aimed to find a well performing pre-trained model. In the last step, the authors increase the number of trainable layers until the accuracy of the POS tagger stops increasing to identify the optimal number of trainable layers. The selection of the hyper-parameters ceases when the authors cannot increase the performance of the model further in a meaningful way or if the performance is satisfactory.

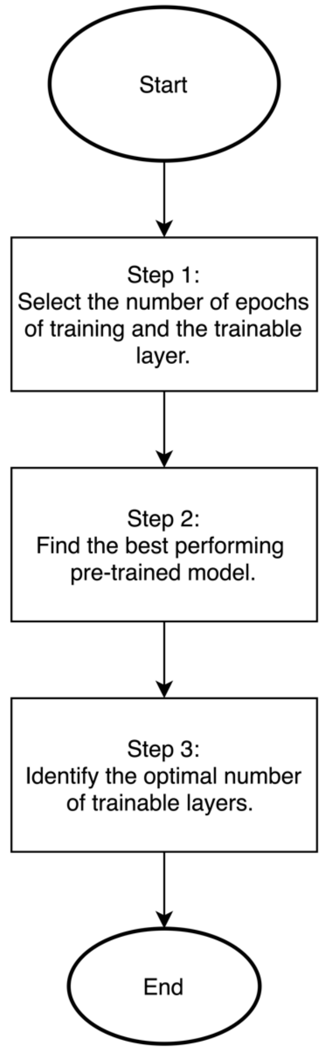

Figure 4. The Three-step Approach for Efficient Grid Search

4. Experiment

4.1. Textual Data

The proposed POS tagger was trained on the POS tagged building codes (PTBC) dataset [96], a dataset that consists of 1,522 POS tagged sentences in chapters 5 and 10 of the 2015 International Building Code (IBC). In total, the PTBC dataset has 39,875 tokens. A token is the smallest unit in POS tagging, such as a word or a punctuation. For example, the word "means" and the period are two tokens in the sentence "The means of egress shall have a ceiling height of not less than 7 feet 6 inches." which has 18 tokens in total. The split of the dataset into training, validation, and testing data is shown in Figure 5: 40% of the dataset as training data, 10% of the dataset as validation data, and 50% of the dataset as testing data. Furthermore, the first 90% of the testing data was further used as the training data of the error-driven transformation rules, which was then tested on the rest of the data. Seven state-of-the-art machine taggers were used to tag the textual data, including: (1) the NLTK tagger [97], (2) the spaCy tagger [98], (3) the Standford coreNLP tagger [99], (4) A Nearly-New Information Extraction System (ANNIE) tagger in the General Architecture for Text Engineering (GATE) tool [37], (5) the Apache OpenNLP tagger [100], (6) the TreeTagger [41], and (7) the RNNTagger [41, 101]. The seven machine taggers were selected because of their high-accuracy, ease of use, and free availability. The most commonly chosen POS tag of words by the machine taggers formed the machine-tagged result. Five human annotators then independently POS tagged the textual data and the most commonly seen tag was chosen for each word. All human annotators are proficient in English and have sufficient background knowledge to understand building codes. POS tags of words by the human annotators formed the gold standard. In both the machine-tagged result and the gold standard, the most commonly chosen POS tag is selected by highest count, meaning that the POS tag that is selected by the most machine taggers or human annotators is selected. For example, if four machine taggers tag the word “doorways” as Plural Noun (NNS), one machine tagger tags the word as 3rd person singular present verb (VBZ). The most commonly chosen POS tag of the word “doorways” is selected to be Plural Noun (NNS), in the machine-tagged result. If there is a tie, the authors break the tie by selecting the tag they deem most appropriate. In the generation of the gold standard, the authors developed a new labeling method in which human annotators address the differences between tagging results of different machine taggers. If all machine taggers tag a word identically, human annotators do not need to change the tag by machine taggers. For words that different machine taggers select different POS tags, human annotators are presented with all tags assigned by machine taggers as options to select from. To account for the risk that a word is not correctly tagged by any machine taggers, human annotators are allowed to assign a POS tag outside the provided tags as well. Human annotators also can change the POS tag of words that machine taggers reached a consensus on. Such changes will need to be discussed and get consensus from all human annotators [102]. The human annotators' tagging results reached an initial inter-annotator agreement of 0.91, which ensured the quality of the gold standard. The dataset contains the POS tags given by all seven machine POS taggers and five human annotators, the most commonly chosen tag by machine POS taggers and human annotators. In this experiment, the proposed POS tagger was trained to tag the textual data as closely as possible to the most commonly chosen tag by human annotators (Figure 6).

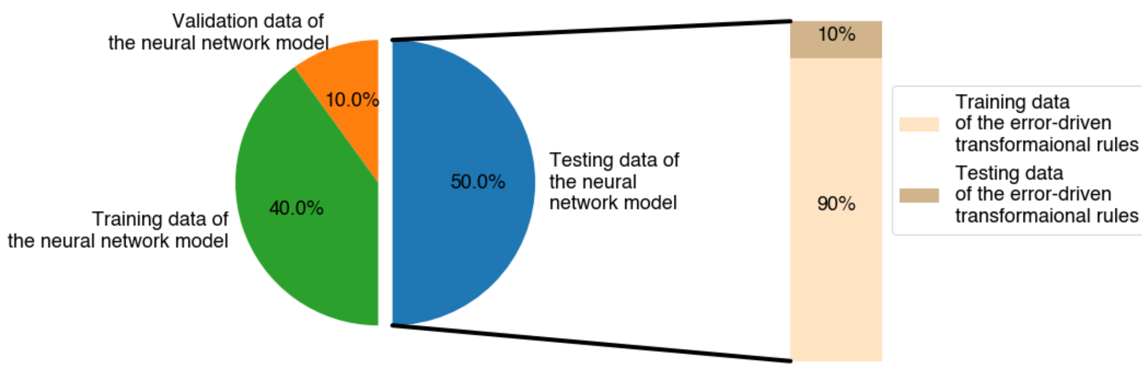

Figure 5. Split of Training, Validation, and Testing Data

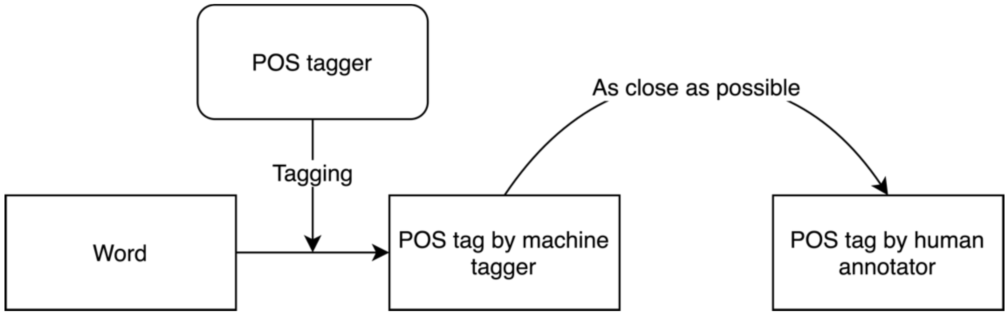

Figure 6. POS Tagger Goal

4.2. Step 1: Select the Number of Epochs of Training and the Trainable Layer

There were two types of trainable layers studied in this research: (1) bidirectional LSTM, and (2) bidirectional GRU. The number of epochs of training cannot be predicted before training [103]. The authors decided to train the model 15 epochs and 50 epochs (arbitrarily selected numbers) to analyze the impact of epochs of training on the performance of the model. The trainable layers were layers of bidirectional LSTM or bidirectional GRU. The size of trainable layers was 128. Between trainable layers, there were dropout layers with a dropout rate of 0.4. The authors selected hyper-parameters such as epochs of training, trainable layer size, and dropout rate based on their past experience in deep learning. Neural network models with these hyper-parameters generally perform well on a wide range of tasks. Although it is possible to do a more thorough search on hyper-parameters, it is out of the scope of this paper. The random number embedding layer significantly saved the training time and allowed grid research in this step. The authors attempted four possible combinations (Figure 7): (1) one layer of bidirectional GRU model that was trained 15 epochs, (2) one layer of bidirectional GRU model that was trained 50 epochs, (3) one layer of bidirectional LSTM model that was trained 15 epochs, and (4) one layer of bidirectional LSTM model that was trained 50 epochs.

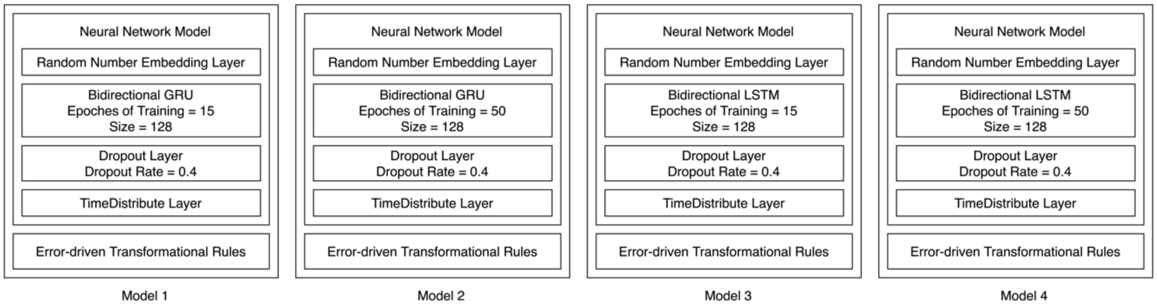

Figure 7. Models Trained in Step 1

4.3. Step 2: Search a Well-performing Pre-trained Model

Although there were multiple potentially well-performing pre-trained models available, the authors selected BERT, which had achieved the state-of-the-art performance on multiple NLP tasks with little fine-tuning needs [61]. The authors tested the eight available versions of BERT: (1) BERT-Large, Uncased (Whole Word Masking), (2) BERT-Large, Cased (Whole Word Masking), (3) BERT-Base, Uncased, (4) BERT-Large, Uncased, (5) BERT-Base, Cased, (6) BERT-Large, Cased, (7) BERT-Base, Multilingual Cased, and (8) BERT-Base, Multilingual Uncased. Therefore, eight models were trained in this step, corresponding to the eight versions of BERT (Figure 8). All of them shared the same trainable layers and were trained the same number of epochs.

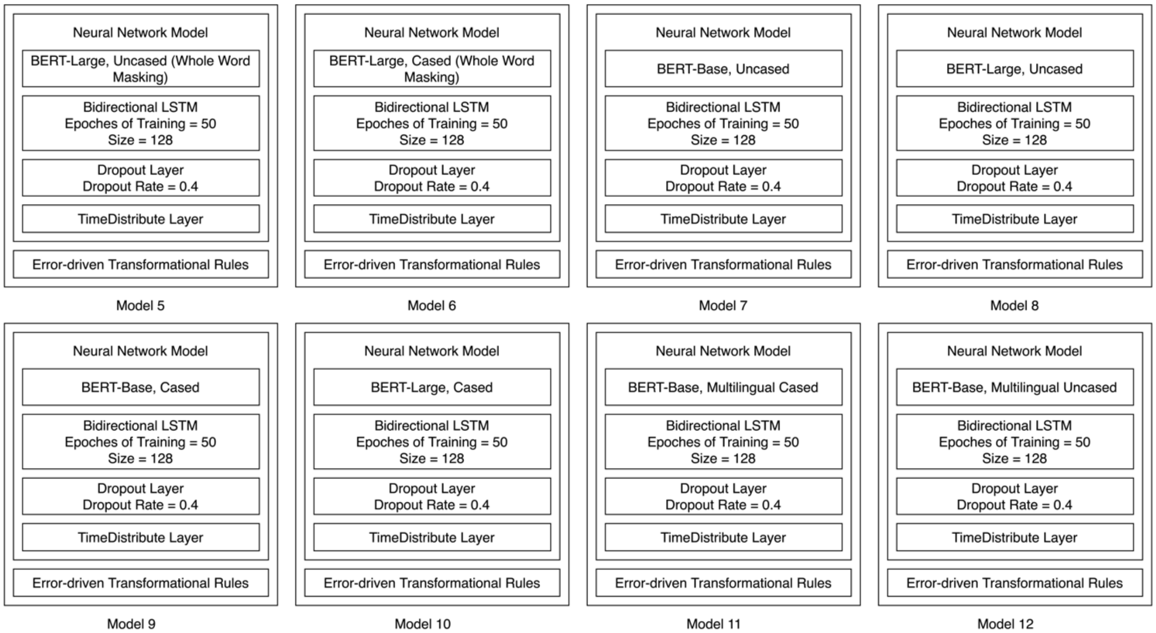

Figure 8. Models Trained in Step 2

4.4. Step 3: Search the Optimal Number of Trainable Layers

Stacking multiple trainable layers could possibly achieve higher precision by capturing more features in the textual data. However, too many trainable layers may lead to overfitting. To find the optimal number of trainable layers, the authors decided to increase the number of trainable layers and dropout layers until the precision stops increasing. There were two models trained in this step: Model 13, which has two bidirectional LSTM layers and Model 14, which has three bidirectional LSTM layers (Figure 9).

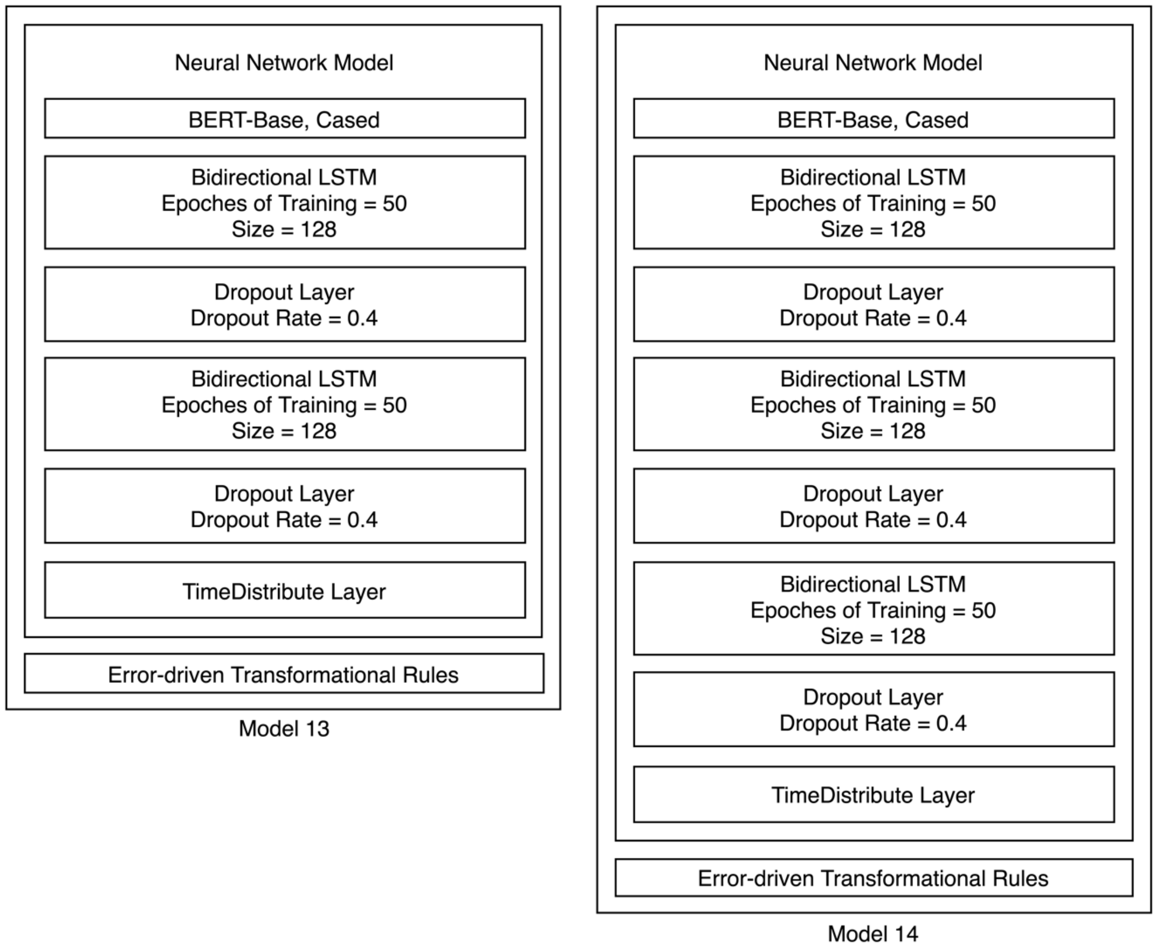

Figure 9. Two Models Trained in Step 3

5. Results and Discussion

To find a well-performing combination of epochs of training, pre-trained models, and trainable layers to use in the POS tagger, the authors trained 14 models (Table 2). The best-performing POS tagger had a combination of one bi-directional LSTM trainable layer, BERT_Cased_Base pre-trained model, and was trained for 50 epochs. This model (Model 9 in Table 2) reached the highest accuracy after applying transformational rules. The optimization of the deep learning component of this POS tagger is out of the scope of this paper, which may be pursued in future research.

Table 2. Summary of the Performance of Models

Model	Before Applying Rules			After Applying Rules		
	Precision	Recall	F1-score	Precision	Recall	F1-score
1	39.02%	17.91%	19.88%	61.59%	51.94%	43.71%
2	89.67%	87.65%	88.14%	93.68%	93.78%	93.64%
3	36.45%	17.41%	20.37%	61.82%	49.93%	43.62%
4	90.15%	87.76%	88.34%	93.53%	93.44%	93.41%
5	90.57%	88.60%	88.87%	94.98%	94.99%	94.88%
6	91.06%	88.64%	89.01%	94.73%	94.75%	94.63%
7	90.40%	88.37%	88.68%	94.16%	94.32%	94.14%
8	89.29%	87.24%	87.60%	93.50%	93.70%	93.49%
9	**91.89%**	**89.71%**	**90.06%**	**95.11%**	**95.42%**	**95.20%**
10	91.49%	89.32%	89.78%	94.50%	94.70%	94.51%
11	89.70%	87.56%	87.80%	94.23%	94.56%	94.33%
12	87.84%	85.92%	86.12%	93.31%	93.03%	93.04%
13	91.81%	89.81%	90.19%	95.04%	95.32%	95.08%
14	91.43%	89.82%	90.07%	94.64%	94.89%	94.70%

5.1. Step 1 Result: Epochs of Training and Trainable Layers Combination

Figure 10 demonstrates the influence of the trainable layer and the epochs of training on the accuracy of POS tagging. For both trainable layers, increasing the number of epochs can increase the precision. However, when the number of epochs was 15, the precision of the bi-directional LSTM model was lower than that of the bi-directional GRU model. When the number of epochs was 50, the precision of the bi-directional LSTM surpassed that of the bi-directional GRU model. This shows that the optimal number of epochs for different pre-trained models could be different.

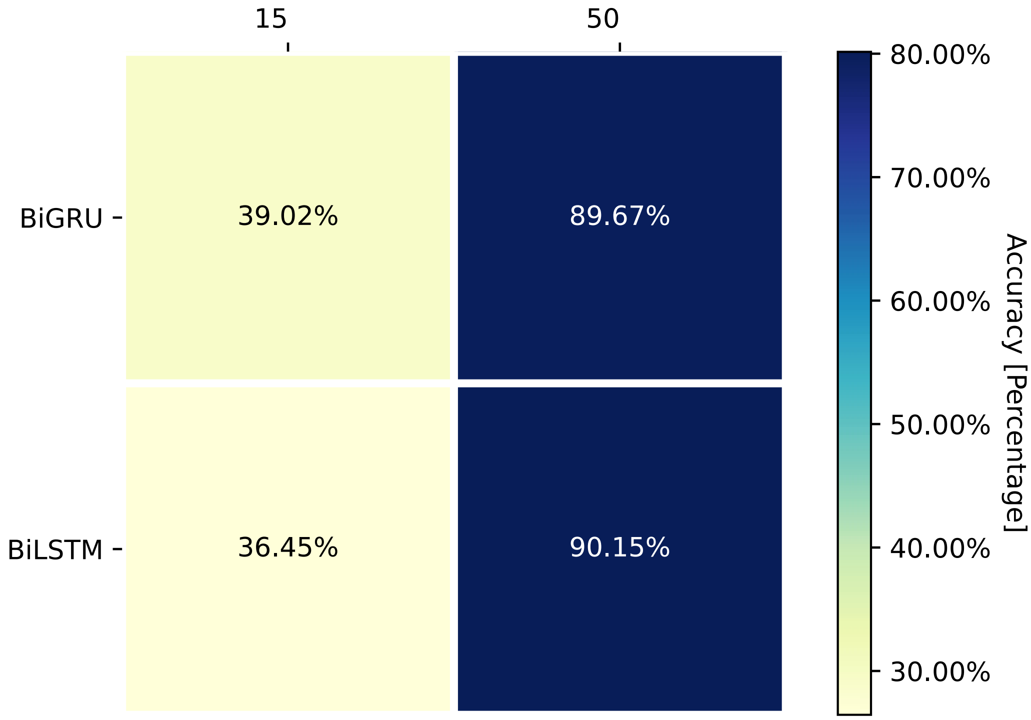

Figure 10. Influence of Epochs of Training and Trainable Layers to Precision

5.2. Step 2 Result: The Best-performing Pre-trained Model

The precision, recall, and F1-score of models with different pre-trained models are shown in Figure 11. All models trained in this step share the same trainable layer and the same number of epochs of training (50). The BERT_Base_Cased model achieved the highest precision, recall and F1-score. The average precision for models with cased models is 91.03% and that for models with uncased models is 89.53% (Figure 11). It shows cased information is useful in the POS tagging of building codes. The average precision for models with large models is 90.60% and that for models with base models (excluding multilingual models) is 91.15%. The two multilingual models were excluded in the comparison because there is no large multilingual model and the current POS tagging task is not multilingual. It may be counterintuitive because larger models generally achieve higher accuracy than smaller models. The authors suggest that more training data is needed to release the full potential of large pre-trained models.

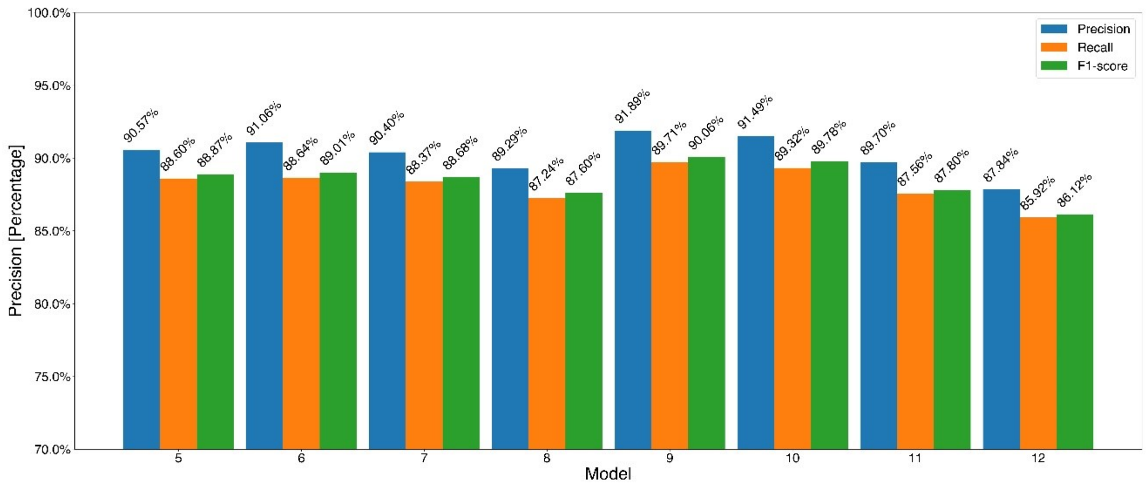

Figure 11. Precision, Recall and F1-score of Models with Different Pre-trained Models

5.3. Step 3 Result: The Optimal Number of Trainable Layers

After the best-performing pre-trained model was identified, the authors started to identify the optimal number of trainable layers. Result of this attempt is illustrated in Table 3. The model with one layer of bidirectional LSTM reached the highest precision. Precision of models decreases as the number of layers increases. The authors concluded that more data is needed to leverage the power of additional trainable layers.

Table 3. Number of Trainable Layers vs. Precision

Layers of Trainable Layers	Precision
1	91.49%
2	89.79%
3	87.84%

5.3.1 Effectiveness of Error-driven Transformational Rules.

This research also confirmed the effectiveness of error-driven transformational rules (Figure 12). The average precision after applying transformational rules is 94.57%. Although the precision before applying transformational rules varied with pre-trained models and trainable layers, the precision after applying the transformational rules all increased. Moreover, POS taggers with higher pre-rule-application precision will also have a higher post-rule-application precision. The transformational rules increase the precision of POS tagger by a margin of 4.02%. The average training accuracy and testing accuracy of all models that use pre-trained models are 95.45% and 94.57%, respectively. The average training accuracy of the models was only 0.88% higher than their average testing accuracy (Figure 13), which alleviated overfitting concerns. The authors also compared the performance of the proposed tagger against the performance of other state-of the-art POS taggers on the PTBC dataset [102] (Figure 14).

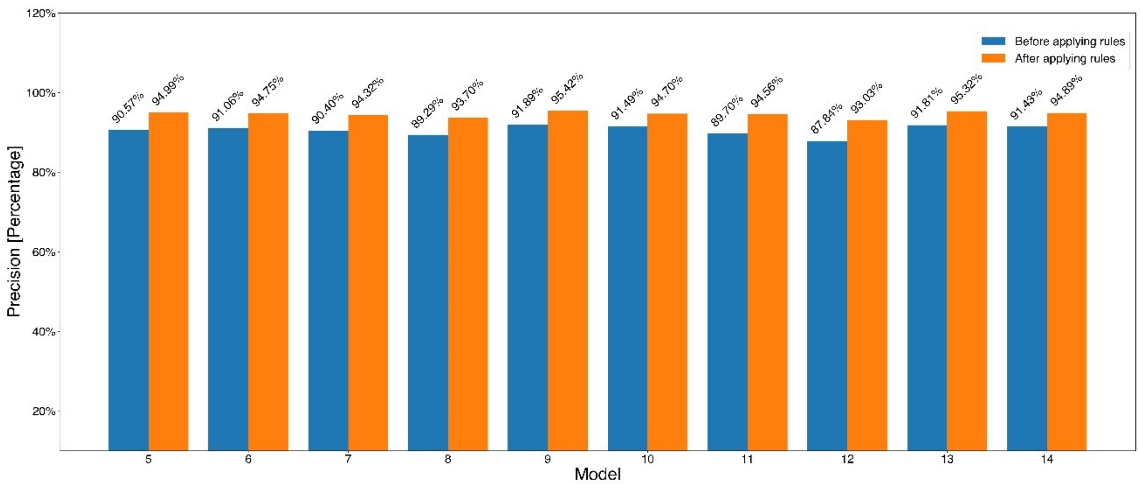

Figure 12: Precision of Each Model Before and After Applying Transformational Rules

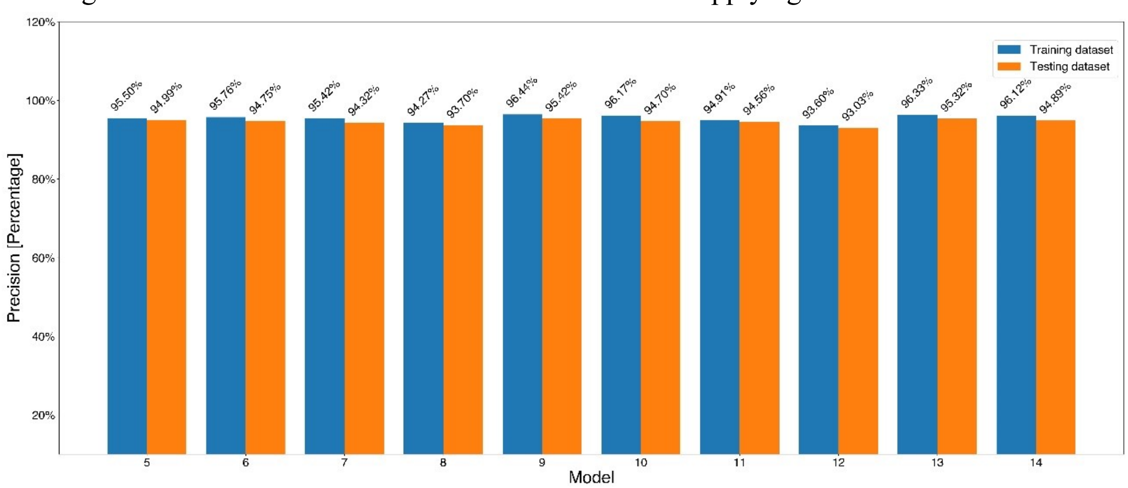

Figure 13: Training and Testing Accuracy of Models

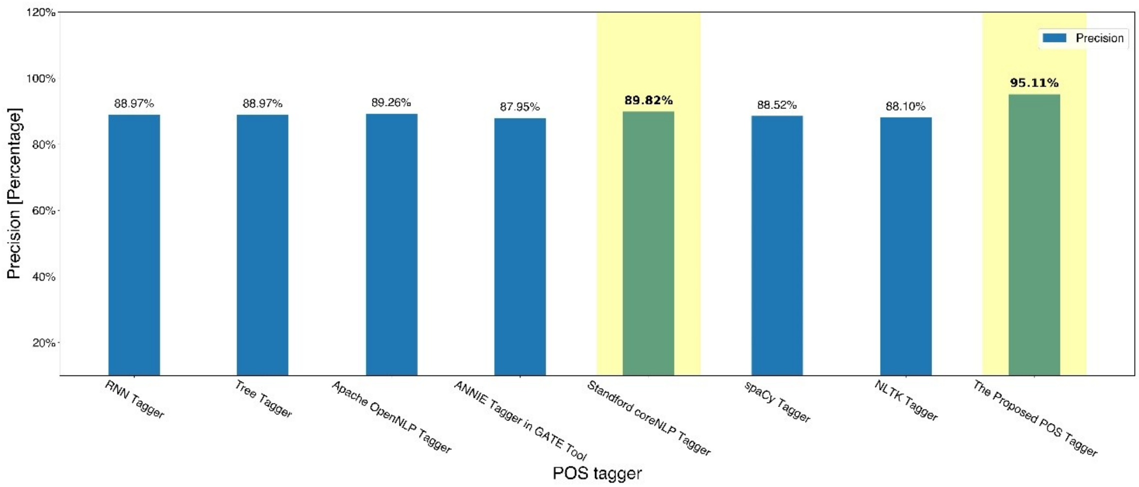

Figure 14. Comparison with State-of-the-art POS Taggers

5.3.2. Effectiveness of GRU

The bi-directional GRU model without BERT can achieve a precision that is comparable to bi-directional LSTM model that is enhanced by BERT. A significant amount of training time can be saved if there is no pre-trained model to fine-tune. The hardware requirement to fine-tune pre-trained models is also significantly higher than that of the random embedding layer. Directly using the bi-directional GRU model can save training time and cut hardware investment while the compromise on the precision of the POS tagger is within an acceptable range.

5.3.3 Tagging Example

To validate this POS tagger, the authors compared the POS tagging result of this POS tagger to a baseline tagger which is a state-of-the-art generic POS tagger. As an example, the baseline tagger incorrectly labeled “horizonal” as a noun. This error may lead to incorrect extraction of embedded engineering knowledge in building codes. In contrast, the proposed POS tagger correctly labeled the word as an adjective. The automated code compliance checking system has a better chance to correctly extract the embedded engineering knowledge in the building codes by the proposed POS tagger, compared to state-of-the-art generic POS taggers.

5.3.4 Impact of Data Split Scenarios

To analyze the impact of different training/testing data split scenarios on the precision, recall, and f1-score, the authors reported the precision, recall, and f1-score of the proposed POS tagger on two other training/testing split methods. The second training/testing split method is using: (1) 60% of the entire dataset as the training dataset of the neural network model, (2) 20% of the entire dataset as the validation dataset of the neural network model, (3) 20% of the entire dataset as the testing dataset of the neural network model, (4) 80% of the entire dataset as the training dataset of the error-driven transformational rules, and (5) 20% of the entire dataset as the testing dataset of the error-driven transformational rules (Table 3). The third training/testing split method is using: (1) 60% of the entire dataset as the training dataset of the neural network model, (2) 20% of the entire dataset as the validation dataset of the neural network model, (3) 20% of the entre dataset as the testing dataset of the neural network model, (4) 90% of the testing dataset of the neural network model as the training dataset of error-driven transformational rules, and (5) 10% of the testing dataset of the neural network model as the testing dataset of error-driven transformational rules (Table 4). Results in all training/testing split scenarios showed consistency in: (1) the improvements of performance when using error-driven transformational rules, (2) the improvement of performance over the state of the art.

Table 4. Results of Second Training/Testing Split Method

Model	Before Applying Rules			After Applying Rules		
	Precision	Recall	F1-score	Precision	Recall	F1-score
1	91.15%	89.39%	89.95%	93.10%	92.80%	92.82%

2	92.86%	91.21%	91.72%	94.82%	94.60%	94.64%
3	77.80%	72.13%	71.64%	83.58%	85.35%	83.37%
4	92.98%	91.20%	91.76%	94.62%	94.25%	94.31%
5	91.97%	90.30%	90.76%	96.04%	95.84%	95.56%
6	92.26%	90.28%	90.84%	96.25%	96.22%	95.99%
7	91.93%	90.32%	90.70%	96.00%	95.94%	95.65%
8	90.49%	89.28%	89.49%	95.85%	95.67%	95.37%
9	**93.18%**	**91.82%**	**92.18%**	**96.43%**	**96.35%**	**96.08%**
10	92.58%	91.17%	91.51%	96.31%	96.27%	96.00%
11	91.70%	89.90%	90.40%	95.79%	95.77%	95.44%
12	89.56%	87.93%	88.28%	95.04%	95.02%	94.70%
13	93.02%	91.65%	92.01%	96.40%	96.22%	95.94%
14	92.90%	91.77%	92.00%	96.83%	96.62%	96.28%

Table 5. Results of Third Training/Testing Split Method

Model	Before Applying Rules			After Applying Rules		
	Precision	Recall	F1-score	Precision	Recall	F1-score
1	91.17%	89.86%	90.23%	92.48%	92.32%	92.25%
2	92.83%	90.59%	91.27%	93.60%	93.19%	93.32%
3	77.91%	69.31%	69.47%	80.81%	80.24%	78.11%
4	92.88%	90.65%	91.34%	93.25%	92.97%	93.03%
5	92.07%	90.49%	90.90%	95.11%	94.71%	94.85%
6	92.06%	90.01%	90.61%	94.61%	94.27%	94.32%
7	91.62%	90.17%	90.43%	93.18%	92.62%	92.79%
8	90.79%	89.28%	89.61%	93.87%	93.50%	93.59%
9	**93.23%**	**91.47%**	**91.96%**	**96.12%**	**95.70%**	**95.84%**
10	92.25%	90.82%	91.20%	94.73%	94.49%	94.55%
11	91.90%	90.14%	90.51%	95.26%	94.93%	95.06%
12	90.31%	88.79%	89.29%	93.07%	92.62%	92.70%
13	92.83%	91.12%	91.49%	95.99%	95.48%	95.65%
14	92.73%	91.30%	91.60%	95.51%	95.26%	95.32%

6. Contributions to the Body of Knowledge

This research has contributions in both theory and practice. Theoretically, it has two main contributions to the body of knowledge. First, it provides a hybrid deep-learning and rule-based method to enhance performance of POS taggers on domain specific texts. The combination of deep learning neural network models and error-fixing transformational rules makes the proposed POS tagger outperform the state-of-the-art POS taggers with limited amount of training data. Many current state-of-the-art POS taggers were trained on the Penn Treebank (PTB) corpora which has 2,499 articles (each article contains tens, if not hundreds, of sentences). This POS tagger was trained on a dataset of only 1,522 sentences. Second, this research shows the potential of deep learning in automated building code information extraction. The promising results of deep learning on the POS tagging of building codes paved the way to more applications of deep learning in automated building code compliance checking and engineering tasks in the AEC domain in general. In practice, the impact of this work on the AEC domain could be profound. It provides a more accurate POS tagger for building codes comparing to the state of the art, which will help automated code compliance checking systems to check more building code requirements automatically. The extension of checkable building code requirements could bring automated code compliance checking systems one step closer to a wide real-world deployment.

7. Limitations and Future Work

One main limitation of this work is acknowledged: the POS tagger still is not error-free. In spite of its improvement over the state of the art, this POS tagger may still not be accurate enough to support an error-free extraction of embedded engineering knowledge in building codes. Errors in POS tagging may have negative effect on the performance of NLP-based automated building code compliance checking systems that leverage it. The authors suggest that research to further increase the accuracy of POS taggers is still needed. The authors also plan to develop automated code compliance checking systems that have the robustness to tolerate a small amount of POS tagging errors.

8. Conclusion

The ability to provide accurate POS tagging results of building codes paves the way to automated regulatory information extraction and widens the possible range of applicable code requirements of automated code compliance checking systems. The authors proposed a new POS tagger to support such systems. This is the first POS tagger that is tailored to building codes. The POS tagger gained information on general English by incorporating pre-trained deep learning models and captured AEC domain specific knowledge by fine-tuning on a domain-specific corpus. The POS tagger directly maps inputted words to POS tags without feature engineering. This nature of deep learning allows future domain experts to enhance the performance of this tagger by directly leveraging more training data. The experiment showed that the bi-directional GRU model without pre-trained models can reach a high precision that is comparable to the precision of the bi-directional LSTM models with pre-trained models. Using bi-directional GRU model can save time and cost to train a POS tagger, without significantly compromising precision. Although more training data may help unleash the full potential of pre-trained models and further improve performance, the authors were able to achieve a 95.11% precision using one bi-directional LSTM trainable layer and BERT_Cased_Base pre-trained model in combination with error-driven transformational rules, which significantly increased over the state-of-the-art.

9. Acknowledgement

The authors would like to thank the National Science Foundation (NSF). This material is based on work supported by the NSF under Grant No. 1827733. Any opinions, findings,

and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF.

10. Reference

[1] Indiana Department of Homeland Security, Codes, Standards and Other Rules, 2020, 2020.

[2] Upcodes, International Building Code 2015 (IBC 2015), 2020.

[3] C.o. Chicago, Average Time for Permit Issuance, 2020.

[4] City of Chicago, Permit Fee Calculator, 2020.

[5] S. Moon, G. Lee, S. Chi, H. Oh, Automatic Review of Construction Specifications Using Natural Language Processing, (2019).

[6] N.O. Nawari, Generalized Adaptive Framework for Computerizing the Building Design Review Process, Journal of Architectural Engineering, 26 (2020) 04019023.

[7] S.J. Fenves, Tabular decision logic for structural design, Journal of the Structural Division, 92 (1966) 473-490.

[8] C.I. Pesquera, S.L. Hanna, J.F. Abel, Advanced graphical CAD system for 3D steel frames, Computer aided design in civil engineering, ASCE, 1984, pp. 83-91.

[9] V.E. Saouma, S.M. Doshi, M. Pace, Architecture of an expert-system-based code-compliance checker, Engineering Applications of Artificial Intelligence, 2 (1989) 49-56.

[10] P.M. Evans, Rule-based applications for checking standards compliance of structural members, Building and Environment, 25 (1990) 235-240.

[11] P. Fazio, C. Bédard, K. Gowri, Knowledge‐Based System Development Tools for Processing Design Specifications, Computer‐Aided Civil and Infrastructure Engineering, 3 (1988) 333-344.

[12] L. Khemlani, CORENET e-PlanCheck: Singapore's automated code checking system, AECbytes, October, (2005).

[13] Q. Yang, IFC-compliant design information modelling and sharing, Journal of Information Technology in Construction (ITcon), 8 (2003) 1-14.

[14] L. Ding, R. Drogemuller, M. Rosenman, D. Marchant, J. Gero, Automating code checking for building designs-DesignCheck, (2006).

[15] C. Eastman, J.-m. Lee, Y.-s. Jeong, J.-k. Lee, Automatic rule-based checking of building designs, Automation in construction, 18 (2009) 1011-1033.

[16] P. Patlakas, A. Livingstone, R. Hairstans, G. Neighbour, Automatic code compliance with multi-dimensional data fitting in a BIM context, Advanced Engineering Informatics, 38 (2018) 216-231.
[17] Q. Fang, H. Li, X. Luo, L. Ding, T.M. Rose, W. An, Y. Yu, A deep learning-based method for detecting non-certified work on construction sites, Advanced Engineering Informatics, 35 (2018) 56-68.
[18] W. Smits, M. van Buiten, T. Hartmann, Yield-to-BIM: impacts of BIM maturity on project performance, Building Research & Information, 45 (2017) 336-346.
[19] J.K. Whyte, T. Hartmann, How digitizing building information transforms the built environment, Taylor & Francis, 2017.
[20] J. Zhang, N.M. El-Gohary, Automated information transformation for automated regulatory compliance checking in construction, Journal of Computing in Civil Engineering, 29 (2015) B4015001.
[21] S. Li, H. Cai, V.R. Kamat, Integrating natural language processing and spatial reasoning for utility compliance checking, Journal of Construction Engineering and Management, 142 (2016) 04016074.
[22] X. Xu, H. Cai, Semantic approach to compliance checking of underground utilities, Automation in Construction, 109 (2020) 103006.
[23] W. Fang, H. Luo, S. Xu, P.E. Love, Z. Lu, C. Ye, Automated text classification of near-misses from safety reports: An improved deep learning approach, Advanced Engineering Informatics, 44 (2020) 101060.
[24] B. Zhong, X. Xing, P. Love, X. Wang, H. Luo, Convolutional neural network: Deep learning-based classification of building quality problems, Advanced Engineering Informatics, 40 (2019) 46-57.
[25] A.J.C. Trappey, C.V. Trappey, J.-L. Wu, J.W.C. Wang, Intelligent compilation of patent summaries using machine learning and natural language processing techniques, Advanced engineering informatics, 43 (2020) 101027.
[26] X. Xue, J. Zhang, Evaluation of Eight Part-of-Speech Taggers in Tagging Building Codes: Identifying the Best Performing Tagger and Common Sources of Errors, The ASCE Construction Research Congress, Construction Research Council of the ASCE Construction Institute, 2020.
[27] V. Getuli, S.M. Ventura, P. Capone, A.L. Ciribini, BIM-based code checking for construction health and safety, Procedia engineering, 196 (2017) 454-461.
[28] X. Tan, A. Hammad, P. Fazio, Automated code compliance checking for building envelope design, Journal of Computing in Civil Engineering, 24 (2010) 203-211.

[29] J. Choi, J. Choi, I. Kim, Development of BIM-based evacuation regulation checking system for high-rise and complex buildings, Automation in Construction, 46 (2014) 38-49.
[30] T.-H. Nguyen, Integrating building code compliance checking into a 3D CAD system, Computing in Civil Engineering (2005)2005, pp. 1-12.
[31] N.O. Nawari, Automating codes conformance, Journal of architectural engineering, 18 (2012) 315-323.
[32] J. Dimyadi, G. Clifton, M. Spearpoint, R. Amor, Computerizing Regulatory Knowledge for Building Engineering Design, Journal of Computing in Civil Engineering, (2016) C4016001.
[33] J. Dimyadi, R. Amor, Automated building code compliance checking - where is it at, Proceedings of CIB WBC, 6 (2013).
[34] S. Singaravel, J. Suykens, P. Geyer, Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction, Advanced Engineering Informatics, 38 (2018) 81-90.
[35] M. Héder, E. Bártházi, T. Vámos, Natural language understanding in governance service automation, IFAC Proceedings Volumes, 44 (2011) 6385-6390.
[36] X. Xu, H. Cai, Semantic frame-based information extraction from utility regulatory documents to support compliance checking, Advances in Informatics and Computing in Civil and Construction Engineering, Springer2019, pp. 223-230.
[37] H. Cunningham, GATE, a general architecture for text engineering, Computers and the Humanities, 36 (2002) 223-254.
[38] A.R. Coden, S.V. Pakhomov, R.K. Ando, P.H. Duffy, C.G. Chute, Domain-specific language models and lexicons for tagging, Journal of biomedical informatics, 38 (2005) 422-430.
[39] L. Abzianidze, J. Bos, Towards universal semantic tagging, arXiv preprint arXiv:1709.10381, (2017).
[40] J. Kupiec, Robust part-of-speech tagging using a hidden Markov model, Computer Speech & Language, 6 (1992) 225-242.
[41] H. Schmid, Part-of-speech tagging with neural networks, Proceedings of the 15th conference on Computational linguistics-Volume 1, Association for Computational Linguistics, 1994, pp. 172-176.
[42] M. Pota, F. Marulli, M. Esposito, G. De Pietro, H. Fujita, Multilingual POS tagging by a composite deep architecture based on character-level features and on-the-fly enriched Word Embeddings, Knowledge-Based Systems, 164 (2019) 309-323.
[43] J. Lee, Y. Ham, J.-S. Yi, J. Son, Effective Risk Positioning through Automated Identification of Missing Contract Conditions from the

Contractor' s Perspective Based on FIDIC Contract Cases, Journal of Management in Engineering, 36 (2020) 05020003.

[44] F.u. Hassan, T. Le, Automated Requirements Identification from Construction Contract Documents Using Natural Language Processing, Journal of Legal Affairs and Dispute Resolution in Engineering and Construction, 12 (2020) 04520009.

[45] P. Zhou, N. El-Gohary, Automated matching of design information in BIM to regulatory information in energy codes, Construction Research Congress 2018, 2018, pp. 75-85.

[46] J. Wang, Y. Chen, S. Hao, X. Peng, L. Hu, Deep learning for sensor-based activity recognition: A survey, Pattern Recognition Letters, 119 (2019) 3-11.

[47] J. Zhang, N. El-Gohary, Semantic NLP-Based Information Extraction from Construction Regulatory Documents for Automated Compliance Checking, Journal of Computing in Civil Engineering, 30 (2013) 141013064441000.

[48] J.R. Finkel, C.D. Manning, A.Y. Ng, Solving the problem of cascading errors: Approximate Bayesian inference for linguistic annotation pipelines, Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2006, pp. 618-626.

[49] E.D. Brill, A corpus-based approach to language learning, IRCS Technical Reports Series, (1993) 191.

[50] A. Ratnaparkhi, A maximum entropy model for part-of-speech tagging, Conference on Empirical Methods in Natural Language Processing, 1996.

[51] T. Brants, TnT: a statistical part-of-speech tagger, Proceedings of the sixth conference on Applied natural language processing, Association for Computational Linguistics, 2000, pp. 224-231.

[52] K. Toutanova, D. Klein, C.D. Manning, Y. Singer, Feature-rich part-of-speech tagging with a cyclic dependency network, Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, Association for computational Linguistics, 2003, pp. 173-180.

[53] J. Giménez, L. Marquez, Fast and accurate part-of-speech tagging: The SVM approach revisited, Recent Advances in Natural Language Processing III, (2004) 153-162.

[54] C. Biemann, Unsupervised part-of-speech tagging employing efficient graph clustering, Proceedings of the 21st international conference on computational linguistics and 44th annual meeting of the association for computational linguistics: student research workshop, Association for Computational Linguistics, 2006, pp. 7-12.

[55] F. Dell'Orletta, Ensemble system for Part-of-Speech tagging, Proceedings of EVALITA, 9 (2009) 1-8.
[56] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, ieee Computational intelligenCe magazine, 13 (2018) 55-75.
[57] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, Journal of machine learning research, 12 (2011) 2493-2537.
[58] N.C. Marques, G.P. Lopes, Tagging with Small Training Corpora, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 63-72.
[59] X. Yu, A. Faleńska, N.T. Vu, A general-purpose tagger with convolutional neural networks, arXiv preprint arXiv:1706.01723, (2017).
[60] F. Chollet, Deep Learning with Python, (2017).
[61] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805, (2018).
[62] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C.H. So, J. Kang, BioBERT: a pre-trained biomedical language representation model for biomedical text mining, Bioinformatics, 36 (2020) 1234-1240.
[63] B. He, D. Zhou, J. Xiao, Q. Liu, N.J. Yuan, T. Xu, Integrating graph contextualized knowledge into pre-trained language models, arXiv preprint arXiv:1912.00147, (2019).
[64] W. Tai, H. Kung, X.L. Dong, M. Comiter, C.-F. Kuo, exBERT: Extending Pre-trained Models with Domain-specific Vocabulary Under Constrained Training Resources, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, 2020, pp. 1433-1439.
[65] C.D. Manning, Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?, in: A.F. Gelbukh (Ed.) Computational Linguistics and Intelligent Text Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 171-189.
[66] D. Tang, B. Qin, T. Liu, Document modeling with gated recurrent neural network for sentiment classification, Proceedings of the 2015 conference on empirical methods in natural language processing, 2015, pp. 1422-1432.
[67] X. Rao, Z. Ke, Hierarchical rnn for information extraction from lawsuit documents, arXiv preprint arXiv:1804.09321, (2018).
[68] N. Bhutani, Y. Suhara, W.-C. Tan, A. Halevy, H. Jagadish, Open Information Extraction from Question-Answer Pairs, arXiv preprint arXiv:1903.00172, (2019).

[69] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A.N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, Tensor2tensor for neural machine translation, arXiv preprint arXiv:1803.07416, (2018).

[70] A.V.M. Barone, J. Helcl, R. Sennrich, B. Haddow, A. Birch, Deep architectures for neural machine translation, arXiv preprint arXiv:1707.07631, (2017).

[71] W. Chan, N. Jaitly, Q. Le, O. Vinyals, Listen, attend and spell: A neural network for large vocabulary conversational speech recognition, 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 4960-4964.

[72] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N.E.Y. Soplin, R. Yamamoto, X. Wang, A comparative study on transformer vs rnn in speech applications, arXiv preprint arXiv:1909.06317, (2019).

[73] Y. Shao, C. Hardmeier, J. Tiedemann, J. Nivre, Character-based joint segmentation and POS tagging for Chinese using bidirectional RNN-CRF, arXiv preprint arXiv:1704.01314, (2017).

[74] B. Plank, A. Søgaard, Y. Goldberg, Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss, arXiv preprint arXiv:1604.05529, (2016).

[75] A. Agarwal, A. Yadav, D.K. Vishwakarma, Multimodal sentiment analysis via RNN variants, 2019 IEEE International Conference on Big Data, Cloud Computing, Data Science & Engineering (BCD), IEEE, 2019, pp. 19-23.

[76] K. Baktha, B. Tripathy, Investigation of recurrent neural networks in the field of sentiment analysis, 2017 International Conference on Communication and Signal Processing (ICCSP), IEEE, 2017, pp. 2047-2050.

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

[78] J.L. Elman, Finding structure in time, Cognitive science, 14 (1990) 179-211.

[79] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, pp. 315-323.

[80] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, Activation functions: Comparison of trends in practice and research for deep learning, arXiv preprint arXiv:1811.03378, (2018).

[81] S. Hochreiter, The vanishing gradient problem during learning recurrent neural nets and problem solutions, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6 (1998) 107-116.

[82] H. Sak, A.W. Senior, F. Beaufays, Long short-term memory recurrent neural network architectures for large scale acoustic modeling, (2014).

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

[84] D. Hu, An introductory survey on attention mechanisms in NLP problems, Proceedings of SAI Intelligent Systems Conference, Springer, 2019, pp. 432-448.

[85] O. Firat, K. Cho, Y. Bengio, Multi-way, multilingual neural machine translation with a shared attention mechanism, arXiv preprint arXiv:1601.01073, (2016).

[86] J. Lu, J. Yang, D. Batra, D. Parikh, Hierarchical question-image co-attention for visual question answering, Advances In Neural Information Processing Systems, 2016, pp. 289-297.

[87] T. Rocktäschel, E. Grefenstette, K.M. Hermann, T. Kočiský, P. Blunsom, Reasoning about entailment with neural attention, arXiv preprint arXiv:1509.06664, (2015).

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

[89] M.-T. Luong, H. Pham, C.D. Manning, Effective approaches to attention-based neural machine translation, arXiv preprint arXiv:1508.04025, (2015).

[90] A. Ambartsoumian, F. Popowich, Self-attention: A better building block for sentiment analysis neural network classifiers, arXiv preprint arXiv:1812.07860, (2018).

[91] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems, 2017, pp. 5998-6008.

[92] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, S. Fidler, Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, Proceedings of the IEEE international conference on computer vision, 2015, pp. 19-27.

[93] A. Zhang, Z.C. Lipton, M. Li, A.J. Smola, Dive into deep learning, Unpublished Draft. Retrieved, 19 (2019) 2019.

[94] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners, OpenAI Blog, 1 (2019) 9.
[95] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, arXiv preprint arXiv:1802.05365, (2018).
[96] X. Xue, J. Zhang, Part-of-Speech Tagged Building Codes (PTBC), 2019.
[97] E. Loper, S. Bird, NLTK: the natural language toolkit, arXiv preprint cs/0205028, (2002).
[98] A. Explosion, spaCy-Industrial-strength Natural Language Processing in Python, URL: https://spacy. io, (2017).
[99] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, D. McClosky, The Stanford CoreNLP natural language processing toolkit, Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, 2014, pp. 55-60.
[100] J. Kottmann, B. Margulies, G. Ingersoll, I. Drost, J. Kosin, J. Baldridge, T. Goetz, T. Morton, W. Silva, A. Autayeu, Apache opennlp, Online (May 2011), www. opennlp. apache. org, (2011).
[101] H. Schmid, Deep Learning-Based Morphological Taggers and Lemmatizers for Annotating Historical Texts, Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage, ACM, 2019, pp. 133-137.
[102] X. Xue, J. Zhang, Evaluation of Seven Part-of-Speech Taggers in Tagging Building Codes: Identifying the Best Performing Tagger and Common Sources of Errors, ASCE Construction Research Congress 2020, 2020.
[103] F. Chollet, Deep Learning with Python, Manning Publications Co.2017.