Sensible at SemEval-2016 Task 11: Neural Nonsense Mangled in Ensemble Mess

Nat Gillin
Gillin Inc.
36 Natchez Street, Seahaven, USA
`gillin.nat@gmail.com`

Abstract

This paper describes our submission to the Complex Word Identification (CWI) task in SemEval-2016. We test an experimental approach to blindly use neural nets to solve the CWI task that we know little/nothing about. By structuring the input as a series of sequences and the output as a binary that indicates 1 to denote complex words and 0 otherwise, we introduce a novel approach to complex word identification using Recurrent Neural Nets (RNN). We also show that it is possible to simply ensemble several RNN classifiers when we are unsure of the optimal hyper-parameters or the best performing models using eXtreme gradient boosted trees classifiers. Our systems submitted to the CWI task achieved the highest accuracy and F-score among the systems that uses neural networks.

1 Introduction

The *Deep Learning Tsunami* has hit the Natural Language Processing (NLP) and Computational Linguistics field (Manning, 2016). Deep neural nets has shown to be the ultimate hammer in various NLP shared tasks, systems trained on neural nets often emerge as the top systems and/or beat state-of-the-art performance (Collobert et al., 2011; Mikolov et al., 2013; Pennington et al., 2014; Levy et al., 2014; Shazeer et al., 2016; Gupta et al., 2015; Jean et al., 2015; Kreutzer et al., 2015; Sultan et al., 2014; Sultan et al., 2015)

In the concluding remarks of the Google's Deep Learning course on Udacity[1], Vincent Vanhoucke said, *"What's really cool about those [neural net application] examples is that you don't have to know much about the problem you're trying to solve"*. Armed with basic knowledge of deep learning and neural nets and almost zero familiarity of the problem, we attempt to treat the Complex Word Identification (CWI) task as a binary classification task using Long Short-Term Memory (LSTM) Recurrent Neural Nets (RNN) with Gated Recurrent Units (GRU).

[1]https://www.udacity.com/course/deep-learning–ud730

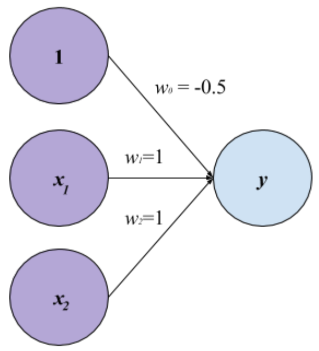
2 Neural Network and Deep Learning

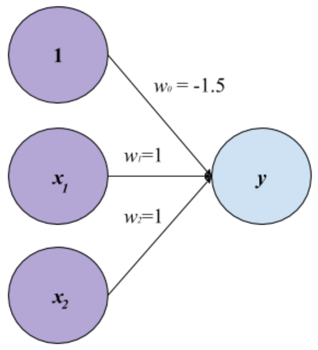
Neural Networks are powerful at modelling various modalities, e.g. signals, text, images, videos. As the name suggests, neural network is inspired by the brain's synaptic transmission mechanism that transmits signalling molecules (aka neurotransmitters) to different signal receptors (aka neurons) throughout our body. Metaphorically, we can emulate a neuron as a computational unit and consider the neurotransmitters as real number inputs and outputs that pass from a neuron to another. Each input to a neuron comes with a associated weight and the neuron will process the different inputs (often by summing them) and passing it to a non-linear function which will provide an output value.

For instance, we can think of a neuron as a typical `AND/OR` logic gate. Given two binary inputs x_1 and x_2 and a *bias* unit with input 1 and a varying weights, we pass it to a neuron that sums the product of the weights and input and passes the sum to a non-linear function that outputs a boolean y value of 0 if the sum is below 0 and 1 if the sum is above 0.

From the left graph and table in Figure 1, we see

Proceedings of SemEval-2016, pages 963–968,
San Diego, California, June 16-17, 2016. ©2016 Association for Computational Linguistics

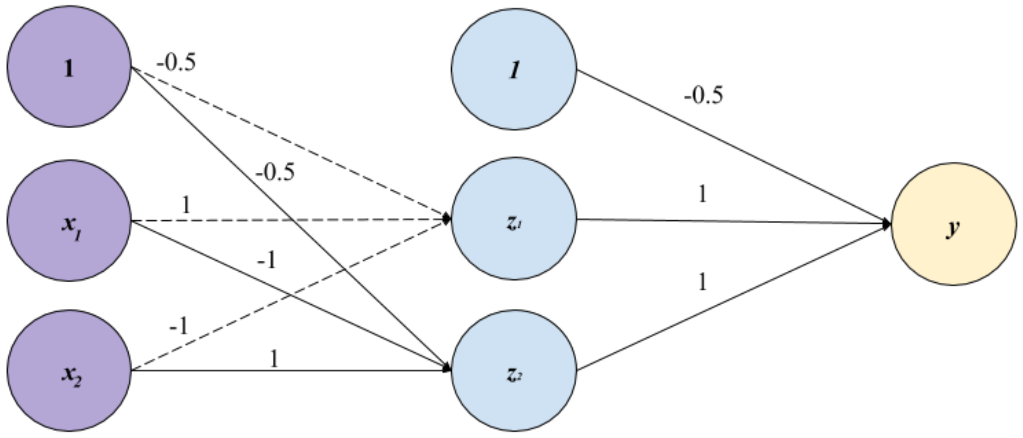




x_1	x_2	*sum = w_0*1 + w_1*x_1 + w_2*x_2*	*y = 1 if sum > 0* *y = 0 if sum <= 0*
0	0	-0.5*1 + 1*0 + 1*0 = -0.5	0
0	1	-0.5*1 + 1*0 + 1*1 = 0.5	1
1	0	-0.5*1 + 1*1 + 1*0 = 0.5	1
1	1	-0.5*1 + 1*1 + 1*1 = 1.5	1

x_1	x_2	*sum = w_0*1 + w_1*x_1 + w_2*x_2*	*y = 1 if sum > 0* *y = 0 if sum <= 0*
0	0	-1.5*1 + 1*0 + 1*0 = -1.5	0
0	1	-1.5*1 + 1*0 + 1*1 = -0.5	0
1	0	-1.5*1 + 1*1 + 1*0 = -0.5	0
1	1	-1.5*1 + 1*1 + 1*1 = 0.5	1

Figure 1: Single Neuron to Emulate an `OR` Gate (left) and `AND` Gate (right)



x_1	x_2	*sum₁ = w_0*1 + w_1*x_1 + w_2*x_2*	z_1	*sum₂ = w_0*1 + w_1*x_1 + w_2*x_2*	z_2
0	0	-0.5*1 + 1*0 + -1*0 = -0.5	0	-0.5*1 + -1*0 + 1*0 = -0.5	0
0	1	-0.5*1 + 1*0 + -1*1 = -1.5	0	-0.5*1 + -1*0 + 1*1 = 0.5	1
1	0	-0.5*1 + 1*1 + -1*0 = 0.5	1	-0.5*1 + -1*1 + 1*0 = -1.5	0
1	1	-0.5*1 + 1*1 + -1*1 = -0.5	0	-0.5*1 + -1*1 + 1*1 = -0.5	0

z_1	z_2	*sum₁ = w_0*1 + w_1*x_1 + w_2*x_2*	*y*
0	0	-0.5*1 + 1*0 + 1*0 = -0.5	0
0	1	-0.5*1 + 1*0 + 1*1 = 0.5	1
1	0	-0.5*1 + 1*1 + 1*0 = 0.5	1
0	0	-0.5*1 + 1*0 + 1*0 = -0.5	0

Figure 2: Emulate an `XOR` Gate with Feed-forward Network

that the neuron emulates an `OR` logic gate where it outputs 1 when either of the input is a positive input and outputs 0 when both inputs are 0s. Similarly, right graph and table in Figure 1 presents a neural depiction of a `AND` logic gate.

If we consider the 2nd row in the left table of Figure 1, the *bias* with the value of 1 and inputs x_1 = 0 and x_1 = 1 are fed into neuro to produce the y output. Within the neuron, it first sums the inputs and the associated weights up (i.e. w_0 * *bias* + w_1 * x_1 + w_1 * x_1). Then using a non-linear thresholding function, the neuron outputs $y = 1$ since the sum is larger than 0. Thus the neuron fulfils the function of an `OR` that accepts a 1 and 0 input bit to produce a positive bit.

The problem gets more complicated when we want to use neurons to emulate an exclusive OR `XOR` logic gate, to get a positive binary output, only one of the inputs can be positive and `XOR` returns a negative output when there are more than one or less than one positive input(s).

We can split the `XOR` problem into small logical expressions:

`XOR` = [x_1 `AND NOT` x_2] OR [`NOT` x_1 `AND` x_2]

As shown in Figure 2, we can emulate an `XOR` gate by stacking two layers of neurons. On the first layer, we solve for

(i) z_1 to represent [x_1 `AND NOT` x_2] with weights -0.5, 1 and -1 attached to the *bias*, x_1 and x_2

(ii) z_2 to represent [`NOT` x_1 `AND` x_2] with weights -0.5, -1 and 1 attached to the *bias*, x_1 and x_2.

At the second layer, we apply the same weights we use for the `OR` gates and feed z_1 and z_2 as the input to produce the `XOR` outputs. Interestingly, if we look at the first layer, we notice that the z_1 and z_2 will never both be 1s. Often the network architecture we use to solve the `XOR` example is referred to as a *feed-forward multi-layered network*.

The logic gates examples motivate the simple use of single neurons and the effect of stacking layers of neurons to produce the desired outcome[2]. Hence the notion, "*Deep Learning*".

[2]The logic gates examples are taken from Emily Fox and Carlos Guestrin machine learning course on Coursera, https://www.coursera.org/learn/ml-foundations

In all the logic gate examples we have manually assigned the weights that are associated with the neurons and it perfectly predicts the desired `XOR` outputs. In practice, these weights has to be trained using pairs of input bits and their respective outputs.

The rest of the paper will not go through the neural network architecture used in our submission in the same level of detail as this section. Goldberg (2015) and Cho (2015) provides a great read on using deep learning and neural networks for NLP tasks and the formal mathematical descriptions of how to train the networks.

3 Recurrent Neural Net

A Recurrent Neural Net (RNN) is an architecture of deep neural network that chains up neurons in a sequential manner. The Elman Network is the simplest formulation of RNN; it allow arbitrarily sized structured inputs to be represented by a fixed-size vector observing the structured properties of the input (Elman, 1990).

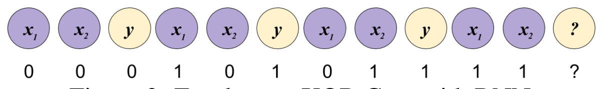


Figure 3: Emulate an XOR Gate with RNN

Returning to the `XOR` problem, instead of using a feed-forward multi-layered network, we can change the problem into a sequential one. Figure 3 shows how the inputs can be chained in a sequential manner in candence, x_1, x_1, y, x_1, x_1, y, ... (*input input output, input input output,* ...). And at the end of the sequence, the network can predict the output of the last set of inputs without an output.

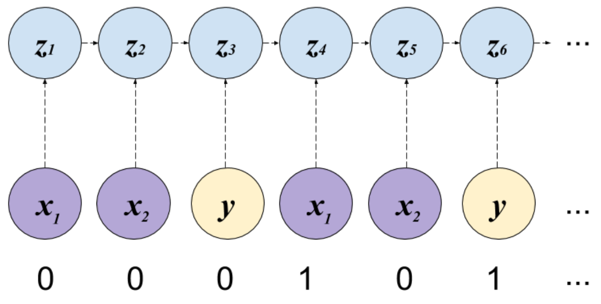


Figure 4: Training an RNN to solve XOR

Figure 4 presents a naive method to chain the inputs x_i, outputs y and hidden units z_i to train an RNN and the dotted lines presents trainable weights.

Despite its simplicity, RNN produces competitive results for sequence tagging (Xu et al., 2015) and language modelling (Mikolov et al., 2010). However, it is hard to train effectively due to the vanishing gradient problem; the gradients in the later steps of the sequence quickly diminishes during back-propagation (Rumelhart et al., 1988) and they don't reach the earlier inputs.

To solve the vanishing gradients problem, Hochreiter and Schmidhuber (1997) introduced the Long Short-Term Memory (LSTM). The intuition is to introduce a "memory cell" to preserve the gradients; at every input state, a gate is used to control how much of the new input should be kept in the memory cell how much it should forget. Cho et al. (2014) proposed a similar "memory" device, Gated Recurrent Unit (GRU), to control that add extra weight matrices to learn what long-distance relationships to remember or forget.

4 Complex Word Identification

Complex Word Identification (CWI) is task of identifying difficult words in a text automatically. Usually, it is structured as a subtask prior to lexical simplification where a difficult words from a text is substituted to simpler ones (Specia et al., 2012; Shardlow, 2013). The inputs of the task is a target word and the context sentence in which it occurs. For example, given the underlined word and the context sentence,

> *The short words math or maths are often used for <u>arithmetic</u> , geometry or basic algebra by young student and their schools.*

The desired output for the inputs would either be a 1 to indicate that the target word is complex and 0 if the target word is not.

4.1 Complex Word Identification with RNN

Neural network has opened a Pandora box where engineers can stack the network in different architectures to train their desired models for almost any NLP task. The sequential nature of language production fits the recurrent structure of the RNN and engineers can easily redesign any NLP task into a sequence prediction task.

Knowing little about the task, we lemmatize and lowercase the sentence[3] and restructure the CWI inputs as a sequence where the target word is separated by a placeholder symbol $< s >$ followed by the context sentence, e.g.

> *arithmetic $< s >$ the short word math or math are often use for arithmetic , geometry or basic algebra by young student and their school .*

Then we fit all the training instances into an RNN network with GRU to output the binary labels. However, we are unsure of the optimal hyper-parameters for the model, so we trained 180 models with varying *embedding sizes* (10, 20, 50, 100, 200, 1000), *GRU sizes* (10, 20, 50, 100, 200, 1000) and *no. of epoch* (1, 3, 5, 7, 10).

We select the top model with the lowest labelling error on the training labels as our `Baseline` submission. Since we do not know the variance between the training and evaluation data, we select outputs from the top 5 models with the lowest training error and train an eXtreme Boosted Trees regressor (Friedman, 2001; Chen and Guestrin, 2016) to produce a single output label.

The open-source implementation of our system can be found on `https://github.com/alvations/stubboRNNess`. It is based on the Passage RNN[4] and XGBoost[5] Ensemble libraries.

5 Results

Table 1 presents the results of the best systems and the neural network systems from the CWI task in SemEval-2016 (Paetzold and Specia, 2016). We have submitted our systems under the team name, *Sensible*.

The CWI task was evaluated based on classic accuracy, precision recall and F-score metric. Additionally, the organizers decided to account for the harmonic mean between the accuracy and recall and they called it the G-Score.

The top teams used a variety of heuristics and classificatoin based techniques. PLUJAGH-SEWDFF uses frequency thresholding where they

[3]Using lemmatizer from `PyWSD` (Tan, 2014)
[4]https://github.com/IndicoDataSolutions/Passage
[5]https://github.com/dmlc/xgboost

Team	**Submission**	**Accuracy**	**Precision**	**Recall**	**F-Score**	**G-Score**
PLUJAGH	SEWDFF	**0.922**	**0.289**	0.453	**0.353**	0.608
LTG	System2	0.889	0.22	0.541	0.312	0.672
SV000gg	Soft	0.779	0.147	0.769	0.246	**0.774**
SV000gg	Hard	0.761	0.138	**0.787**	0.235	0.773
Sensible	Baseline	0.591	0.078	0.713	0.140	0.646
Sensible	Combined	0.737	0.072	0.390	0.122	0.510
AmritaCEN	w2vecSim	0.627	0.061	0.486	0.109	0.547
CoastalCPH	NeuralNet	0.693	0.063	0.398	0.108	0.506
AmritaCEN	w2vecSimPos	0.743	0.060	0.306	0.100	0.434

Table 1: Results of Best (Upper) and Neural Network Systems (Lower) in the SemEval-2016 CWI Task

consider any word that occurs less than 147 times in the simple English Wikipedia to be complex. Similarly, LTG-System2 uses threshold features to train a decision tree classifier. SV000gg combines 23 different systems uses soft and hard voting ensemble, their pre-ensembled systems includes threshold- and lexicon-based heuristics and machine learning classifiers trained on 69 distinct linguistic features.

CoastalCPH's NeuralNet system extracted an array of features (including parts-of-speech, frequencies, character perplexity and embeddings) and they train a deep neural network with 2 hidden layers. AmritaCEN's w2vecSim trained an SVM classifier using Word2Vec embeddings and the similarity between the target word, in addition, they used character and token based features to train the classifier. Their w2vecSimPossystem added a POS feature to train the classifier.

Among the neural network systems, our baseline system achieved the highest F- and G-score. We are also the only team that restructured the target word and sentence to train recurrent neural net to predict the output label. One possible reason for the poor performance of our systems is due to the training data size of the task. The training data contains 2,237 labelled instances while the test data contains 88,221 instances. Given more data, we believe that our system can scale towards accuracies comparative to the top systems.

Although our ensemble system performed poorly in the harmonic scores, we see that it achieves reasonably high accuracy close to the top systems. Our ensemble system was penalized due to the low recall and rate. Provided that we have more training data, the recall should proportionally increase and improve our ensemble system.

6 Conclusion

In this paper, we motivated the use of deep learning and neural nets in NLP applications and introduced basic notions of feed-forward and recurrent neural nets through the `XOR` example. And as expected, we can easily build a relatively competitive system with little understanding of the task by restructuring the inputs as a sequence to train an RNN classifier.

We have introduced a novel approach using RNN to solve the complex word identification task and showed that we can easily ensemble several RNN classifiers if we are unsure of the optimal hyperparameters or the best performing models using extreme gradient boosted trees classifiers.

Acknowledgments

We thank Jimmy Callin, Jon Dehdari and Yoav Goldberg for the pointers on the literature review sections on (Recurrent) Neural Nets.

References

T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *ArXiv e-prints*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Kyunghyun Cho. 2015. Natural language understanding with distributed representation. *arXiv preprint arXiv:1511.07916*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Yoav Goldberg. 2015. A primer on neural network models for natural language processing. *arXiv preprint arXiv:1510.00726*.

Rohit Gupta, Constantin Orasan, and Josef van Genabith. 2015. Reval: A simple and effective machine translation evaluation metric based on recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1066–1072, Lisbon, Portugal, September.

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

Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. Montreal neural machine translation systems for wmt15. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 134–140, Lisbon, Portugal, September.

Julia Kreutzer, Shigehiko Schamoni, and Stefan Riezler. 2015. Quality estimation from scratch (quetch): Deep learning for word-level translation quality estimation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 316–322, Lisbon, Portugal, September. Association for Computational Linguistics.

Omer Levy, Yoav Goldberg, and Israel Ramat-Gan. 2014. Linguistic regularities in sparse and explicit word representations. In *CoNLL*, pages 171–180.

Christopher D Manning. 2016. Computational linguistics and deep learning. *Computational Linguistics*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, volume 2010, pages 1045–1048.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Gustavo Henrique Paetzold and Lucia Specia. 2016. Complex word identification. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, San Diego, California.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Matthew Shardlow. 2013. A comparison of techniques to automatically identify complex words. In *51st Annual Meeting of the Association for Computational Linguistics Proceedings of the Student Research Workshop*, pages 103–109, Sofia, Bulgaria, August.

Noam Shazeer, Ryan Doherty, Colin Evans, and Chris Waterson. 2016. Swivel: Improving embeddings by noticing what's missing. *arXiv preprint arXiv:1602.02215*.

Lucia Specia, Sujay Kumar Jauhar, and Rada Mihalcea. 2012. Semeval-2012 task 1: English lexical simplification. In *Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 347–355.

Md Arafat Sultan, Steven Bethard, and Tamara Sumner. 2014. DLS@CU: Sentence Similarity from Word Alignment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 241–246, Dublin, Ireland, August.

Md Arafat Sultan, Steven Bethard, and Tamara Sumner. 2015. DLS@CU: Sentence Similarity from Word Alignment and Semantic Vector Composition. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 148–153, Denver, Colorado, June.

Liling Tan. 2014. Pywsd: Python implementations of word sense disambiguation (wsd) technologies [software]. https://github.com/alvations/pywsd.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. Ccg supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 250–255, Beijing, China, July.