Adversarial Examples in NLP Contexts

Alex Sax @asax, Dylan Moore @dmoore2

Abstract:

Many state of the art NLP algorithms currently use deep learning techniques. A 2014 paper by Goodfellow et al demonstrates that, at least in image classification, error rates from deep learning can be improved by adversarial training. Generating adversarial examples relies on having high-dimensional input spaces and many output labels. We present two methods of generating adversarial examples, and also introduce a new loss function for training word vectors in a CBOW model.

Using QVEC as a metric for our word vectors, these techniques both improve on the vanilla CBOW model, and these methods can be used in conjunction with other generalization techniques such as dropout and early stopping.

Introduction

Distributional word vectors recently exploded into the machine learning and NLP scene due to their ability to process large amounts of unlabeled data, and the speed of training. Any improvements made to word vectors will trickle downstream to other tasks.

Additionally, and also from Google, adversarial examples have been shown to demonstrate the limits of our current models, and also to improve the performance of models that train on these examples. Previously, though, adversarial examples have not been created or used for natural language domains. We believe this is because there have been no clear analogues to the adversarial examples in image classification domains.

In this paper we present a way to extend adversarial examples to natural language domains, and show that adding these examples to our data set improves word vector quality as measured by QVEC developed by Tsvetkov et. al [3]. We also present a new loss function that captures the spirit of adversarial examples, and also improves word vector quality without sacrificing asymptotic training time.

We implement our word vectors using a CBOW model in TensorFlow and train on the `text8` corpus to produce adversarial examples, which can be later combined with the initial training data to produce a more robust word vector representation than the original data alone. While we originally planned to measure how much additional data each adversarial example was worth, we found that on the `text8` corpus, actually produces *worse* vectors when we use the whole data set. This is true not just of our CBOW implementation, but also off-the-shelf ones such as Gensim. Consequently, we present just the QVEC scores for varying amounts of data, and with/without adversarial examples.

Word Vector Model Descriptions In Depth:

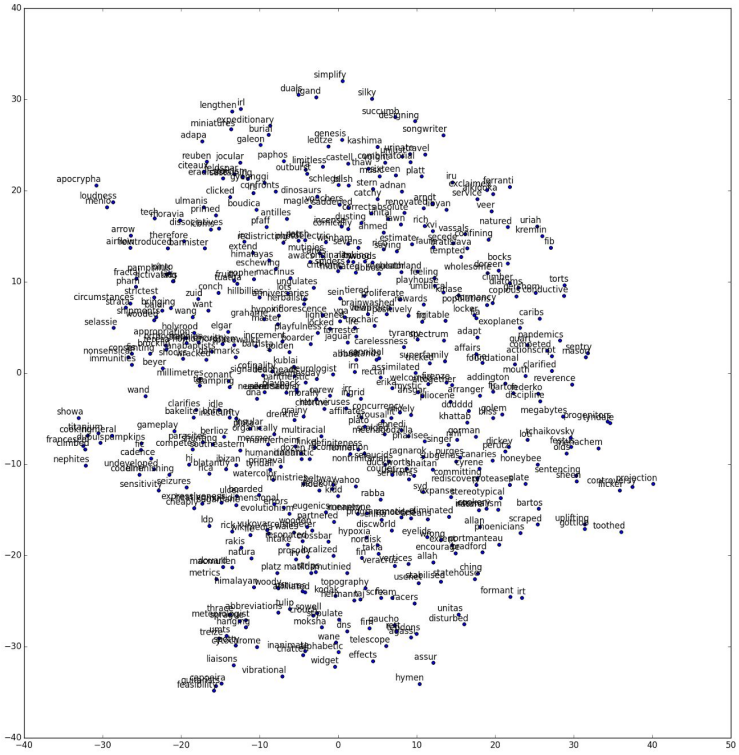
There are currently several models available for creating word vectors, and they are all roughly the same. The main ones are GloVe, Word2Vec-CBOW, and Word2Vec-skip-gram. For our project, we chose to use Word2Vec-CBOW because it is a predictive model that is efficient to train, and has a design that we believe lends itself well to adversarial training techniques. It trains a neural network (with one hidden layer of arbitrary dimensionality--i.e. how word vectors are internally represented) for word vector representations by predicting target words given the context. We train the model using noise-contrastive estimation as our loss function. Over multiple iterations, this maximizes the likelihood that model will predict the right candidate, which results in an accurate word vector representation.

Stack Description

We chose to use the new TensorFlow library in python because of its robust documentation, superior speed, and functional versatility. TensorFlow uses graphs to represent the mathematical computations in a neural network, where edges in a graph represent tensors and nodes represent endpoints or intermediary results dependent on input propagating across the edges through tensor operations.

After training our neural network for 100,000 iterations, we store the resulting word vector representations in a text file, which we pass into Yulia Tsvetkov's python implementation of QVEC.

Since TensorFlow makes it easy to do, we were able to visualize our model in 2-Dimensions using t-SNE. The following diagram is the output of our own TensorFlow Word2Vec using CBOW model.



Adversarial Examples For Word Vectors

In the paper that inspired this project, Goodfellow et al [2] generated adversarial examples for an image classification task. These examples were input images that had been modified in a nearly imperceptible way by changing the values of particular pixels. The examples were special because even though they were all but indistinguishable from the original (which the NN correctly classified), they caused the neural network to make an incorrect classification with high confidence.

One of the crucial challenges to our project was coming up with a reasonable NLP extension to Goodfellow et al's approach. After consulting with Goodfellow et al [2] and Jon Gauthier about different approaches, we pursued a Word2Vec-CBOW model, but there are analogues in the others. We present two methods of generating adversarial examples for word vectors. Each one is designed to take a context and modify it to predict a chosen (but wrong) label.

We also considered generating adversarial examples in other tasks such as NER and POS tagging. Some hypothetical adversarial examples are outlined below, but we ultimately

chose to use them for word vectors because the benefits would trickle downstream to other tasks.

Task	Adversarial Example	correct candidate	Predicted
Movie sentiment	This movie was great!	+	-
Word prediction in context	**fly** sat ___ the **fire**	by	**dragon** (with high confidence)
NER	Barack Obama spoke	[Barack Obama](PERSON)	[Barack Obama](ORG)
POS tagging	The complex houses soldiers	The/DT complex/NN houses/VBG soldiers/NNS	The/DT complex/JJ houses/NNS soldiers/NNS

Fig. 1. Some hypothetical adversarial examples for different NLP tasks

Specific Requirements for a CBOW Adversarial Example:

We base our requirements for an adversarial example on those defined in the Goodfellow et al paper. However, we modify the requirements of an adversarial example slightly so that they makes sense for the context-label pairs in our CBOW model. The three requirements that we enforce are:

1. The new context made from the perturbed word vectors must contain at least word that is not exactly the same as the corresponding word in the original context. To help ensure that the new context is still reasonable, we additionally require that the words in the new context are not all identical to one another.
2. The word vectors in the original context must be perturbed minimally, so that the correct candidate makes sense in both the new and old contexts (this requirement is discussed in the previous section)
3. The new context must predict that the randomly chosen false word was originally considered less likely in the old context, and is now considered more likely than the correct label.

All three requirements have direct corollaries to Goodfellow et al's requirements for an adversarial example. Although our third requirement is weaker than the analogue in Goodfellow, is strong enough given the nature of the vector space we are working in. Due to the fact that the word vector space is quite sparse, our model tends to assign extremely high confidence (often

over 99%) to the first word predicted - regardless of whether and example was adverarial. From there, its confidence ratings sharply decline for the remaining words in the vocabulary. According to our tests, it is very rare for either the correct candidate word or the negatively sampled candidate to in the set of words that have reasonably high confidence ratings (greater than $1 * 10^{-5}$). Further, the negatively sampled candidate is typically a word which has a much lower confidence rating than the correct candidate (several orders of magnitude). These findings suggest that for a good adversarial example, it is sufficient for us to require that the adversarial context will cause our model to predict the negatively chosen candidate with a higher confidence than the correct candidate.

Two Methods for Generating Adversarial Examples

Equation 1.

$$W_{adversarial} = nearestWord(V_{original\ word} - \eta_{gold} * \nabla_{gold} + \eta_{negative\ sampled\ candidate} * \nabla_{negative\ sampled\ candidate})$$

where:

$V_{original\ word}$	= the word vector of the original context word
$W_{adversarial}$	= the adversarial replacement word for the context word with the word vector $V_{original\ word}$
$nearestWord$	= a function that returns the nearest vocabulary word to the given word vector, this may be the original context word
∇_{gold} , $\nabla_{negative\ sampled\ candidate}$	= the gradients of the word vectors of the gold candidate and the randomly chosen candidate word (from negative sampling) respectively
η_{gold} , $\eta_{negative\ sampled\ candidate}$	= two constants, for step sizes

This iterates over each of the words in the CBOW context, modifying the word vector that word to make it more likely to produce a chosen (and wrong) label. It first moves the vector away from the correct label, and then towards the chosen label. We then find the nearest neighbor to the modified context word.

If the η constants are correctly tuned, this small perturbation can result in a new context that our model finds more indicative of a false word than the correct word. In these cases, we store the modified context, and the correct label - this is an adversarial example.

This method is the most intuitive for finding adversarial examples because it adds a small fixed amount of noise to the original vector, and then another small amount to push the output towards the desired direction. Unfortunately, this step size is quite hard to choose, especially since word vectors are often of different lengths, and what makes sense for one word may not work for another. As the step size increases, this method will eventually choose whatever vector lies in the direction of the gradient, obliterating the information in the original word. Large step sizes end up generating contexts that are just one word over and over again.

This problem was addressed in Goodfellow et al [2] paper in way that doesn't have a simple or clear corollary in the word vector space. Goodfellow's team made sure that the adversarial image they created was an imperceptibly modified version of the original by allowing each pixel value in the image to change only by the smallest incremental value possible (one bit in each pixel).

We present another option with better asymptotic behavior. **Equation 2** finds the vector lies in the direction of our gradient from the original word vector. We then add the constraint that the new vector must be within a certain radius from the original.

Equation 2.

$$W_{adversarial} = argmax\left(\frac{(V_{adversarial\ word} - V_{original\ word}) \cdot \nabla_J}{|(V_{adversarial\ word} - V_{original\ word}| \cdot |\nabla_J|}\right)$$

where:

$V_{original\ word}$	= the word vector of the original context word
$V_{adversarial\ word}$	= the word vector of the original context word
$W_{adversarial}$	= the adversarial replacement word for the context word with the word vector $V_{original\ word}$
∇_{gold}	= gradient of the loss function
η	= maximum distance

We further require that $norm(V_{adversarial\ word} - V_{original\ word}) < \eta$ so that our adversarial word is not too far away from our original word. This is one way of requiring that our new context is not so different from our original, or is near the 'just noticeable difference' for words.

This new equation has the desirable property that the new vector lies in the direction of the gradient, and also the desirable property that the new vector not be too far away from the original. The norm can be euclidean if we want to measure the euclidean distance between the two, or we can take the negative infinity norm if we want each dimension to be close. The negative infinity norm was the one used in the original Goodfellow paper.

Both these methods suffer the drawback that they require the embedding matrix to be multiplied against something. This is a very expensive operation that requires $O(|V| \cdot d^2)$ time. It is therefore prohibitively expensive to generate large numbers (megabytes) of adversarial examples.

After examining the examples generated from both equations, we suggest using **equation 2** and tuning η with an iterative approach. We increase this parameter from 0 by very small amounts for many iterations and repeated the process of adversarial example generation until we were able to successfully produce a minimum number *k* of adversarial examples for a given batch of input contexts. In other words, we perturbed the the context words as little as possible in order to produce valid adversarial examples. After looking through many batches of generated adversarial examples, we were ultimately satisfied that the new contexts were similar enough to the original to be considered adversarial.

Analysis of Generated Adversarial Examples

Because of our small data set, we used a context window of size one. With unlimited data, this would limit the quality of our word vectors, but on a small data set this does not present such an issue.

From just two words, it's impossible to correctly predict the target label each time. Even on non-adversarial examples, the model gets it wrong almost all the time. What makes these *adversarial* examples is that in the original context, the correct word is ranked above the adversarial label - but this is reversed in the new context (even though it shouldn't be!). These seem to be pretty good, but sometimes the adversarial contexts don't make sense, or it's hard to imagine a sentence that would permit such a context.

Here are some example results.

Original Context	Adversarial Context	Correct Label	Adversarial
he ___ possession	interwiki ___ possession	called	among
of ___ time	haer ___ sapowski	working	ruskes
his ___ in	his ___ of	ancestors	invention
of ___ american	the ___ american	indigenous	invention

Fig. 2. Generated adversarial examples.

The adversarial examples are of mixed quality. Some examples are excellent (1 and 4) - the correct word fits in both contexts and the adversarial fits in neither. Other examples are not adversarial at all: the first context fits the correct word only, and the second fits the adversarial one only. Other examples are utter nonsense. Sometimes both words fit to varying degrees. The breakdown is roughly 25/5/40/30. This is better than the one we achieved with **equation 1**, and we believe this is good enough to justify the generation procedure.

One improvement might be to increase the size of our context window. Another solution could be to impose more stringent constraints on the generation process so that only grammatical contexts are allowed.

On the other hand, we only successfully generate adversarial examples 3% of the time. That is, most of the time when we generate an example, we either do not change the context because there is no neighbor near enough, or else we change the context in a way that is not adversarial enough. More constraints would further reduce this yield.

Folding in the examples and training on the augmented dataset significantly improves performance (Fig. 3). We note, however, that because we have a limited number (only 128) examples, the training ends up converging with the vanilla version. It's also possible that we are overfitting to these examples because we train on them multiple times. We also note that the examples were created against the Vanilla version at 100000 iterations. It therefore makes sense that they would the the most performance gains as the model approaches the vectors it would have at 100000 iterations. What's more interesting is that the model has performance gains some earlier stages. This is the result of the interesting property that adversarial results tend to be adversarial across models (Goodfellow et al [2]).

We are also running a version where we generate adversarial examples for the model every 10000 iterations, and the train on these examples.

A Modified Loss Function

Folding data back into the training set acts as a set of regularization examples that help our model generalize. We found that it is possible to achieve a similar effect without the slowdown by adding an adversarial regularization parameter into the loss function.

Our training function was Noise Contrastive Estimation, which assumes the data is drawn from a 'true' distribution and then compares the data against a number of bad examples pulled from a distribution of noise.

The parameters are like so:

```
J = nce_loss(weights, biases, input, correct_label,
        num_sampled, vocab_size)
```

Recall that the guiding principle behind adversarial examples is that a seemingly similar input is wrongly interpreted to produce a drastically altered output. These examples are chosen to be the most damaging to the model. The way for a model to remain robust to adversarial examples

is therefore to maximize the distance between the correct label and other labels. We therefore want to build this maximization into the loss function in the same way that we want to maximize margins in a hinge loss. The new loss function has the same parameters, but with the input modified to be damagingly noisy.

$$\tilde{J} = \texttt{alpha*nce\_loss(weights, biases, input +} \eta * \nabla_J \texttt{,}$$
$$\texttt{correct\_label, num\_sampled, vocab\_size)+ (1-alpha)*}J$$

We push the input in the direction that makes it hardest to predict the correct label, and then use our original loss.

We chose `alpha` to be 0.5. Other values may work better, but this worked well enough for us that we did not change it.

We found that this improved our QVEC scores at every stage of training.

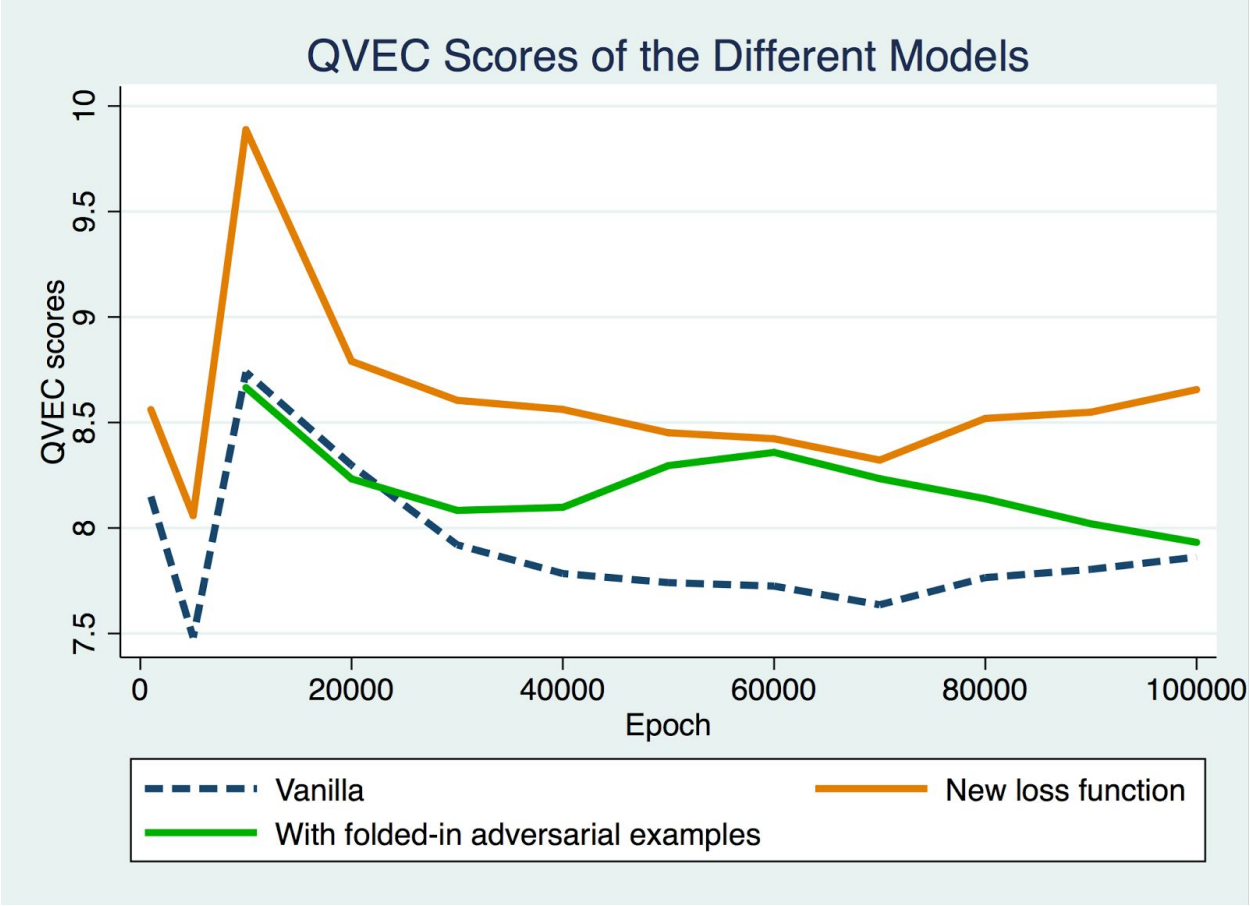


Fig. 3. Chart of the performance of the different models.

CBOW Error Analysis

Surprisingly, not only did our QVEC scores level off from 10MB to 100MB of input data - they actually got worse. We thought this was a bug in our CBOW implementation, but even an off the shelf solution, Gensim, had the same problem. We therefore think that this behavior is due to the structure of the `text8` corpus. The `text8` corpus is comprised of the first 100MB of data from Wikipedia, and it's possible that after 10-20MB, the topics of the articles change abruptly. This might be why all implementations see QVEC score declines after this point. It's also possible that this is an irregularity of QVEC, though we do now know the cause in this case. Another possibility is this is just an artifact of word vectors when expanding their vocabulary for core words to auxiliary words. Nevertheless, our word vectors seem to be good. Here are some randomly chosen words from the 100 most common, and their nearest neighbors:

```
three: seven, eight, four, one, zero, six, nine, five,
or: a, the, crete, zero, bloodless, macneille, is, fsi,
being: glacial, surfer, genome, queer, parry, fuelled, shrewsbury, falkland,
other: some, spices, many, umbilicus, uat, weekends, zhukovsky, various,
four: three, seven, eight, nine, five, one, six, crashed,
been: were, already, lubango, statically, logia, sinned, friar, duvall,
american: lemma, namadgi, americans, initiator, laker, actuarial, whitman,
than: servile, visibly, inclination, much, fidei, qualitative, get,
were: been, newly, that, verily, prosecuting, are, triglycerides, amazonian,
known: used, seen, such, leandro, described, inspiring, defined, remembered,
nine: seven, one, eight, zero, three, four, five, and,
english: letters, spellings, kaliningrad, peloponnesus, irish, written,
on: in, mystically, electorate, united, that, sandwich, murabits, eidgen,
```

Fig. 4. Nearest neighbors to some randomly selected words (cosine distance).

Conclusion

Adversarial examples expose the limits of our models by providing inputs that are similar, but which produce vastly different outputs. Unfortunately, with the current CBOW model, they are also prohibitively expensive to generate because they require us to multiply our entire vocabulary embeddings matrix by another matrix. One of the great advantages of word2vec is that it updates the model stochastically, and it's fast. This allows it to process huge amounts of data quickly, and to produce high quality vectors. Adversarial examples are slow to generate, but produce more bang-for-the-buck than does normal data. In this way, adversarial examples are akin to gradient descent, while normal data is like stochastic gradient descent.

When data is scarce, it may make sense to generate adversarial examples to increase the size of the data set. On the other hand, when data is plentiful the time required to produce adversarial examples can be prohibitive. In this case it still makes sense to use the modified loss function which captures the intuition of adversarial examples.

References

[1] DYER, C. Notes on noise contrastive estimation and negative sampling. *CoRR abs/1410.8251* (2014).

[2] GOODFELLOW, I. J., SHLENS, J., and SZEGEDY, C. Explaining and harnessing adversarial examples. *CoRR abs/1412.6572* (2014).

[3] TSVETKOV, Y., FARUQI, M., LING, W., LAMPLE, G., and DYER, C. Evaluation of word vector representations by subspace alignment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (Lisbon, Portugal, September 2015), Association for Computational Linguistics, pp. 2049–2054.

Appendix

Epoch	Vanilla	W/Adversarial Examples	New Loss Function	Gensim*
1000	8.14893		8.56106	21.045
5000	7.48099		8.05868	11.1778
10000	8.73868	8.66568	9.88762	10.7389*
20000	8.29735	8.23276	8.79005	9.32261
30000	7.92008	8.08329	8.60511	9.03799
40000	7.783993	8.0977	8.56187	8.87582
50000	7.74087	8.29563	8.45141	9.11336
60000	7.72459	8.35831	8.42307	8.94156
70000	7.63592	8.23368	8.32155	9.02092
80000	7.76487	8.13825	8.51937	8.99128
90000	7.80403	8.01979	8.54888	9.01636
100000	7.86228	7.93189	8.65598	8.95044

*Gensim, due to omitting words that occur too few times, will have a higher QVEC than expected in a CBOW model. If we remove this restriction, then our vocabularies will be the same size, but the actual vocabularies will be different. Gensim then achieves a QVEC of 8.51691 at 10000 iterations.