Semantic role labeling using a neural network

Jens Hellström

January 14, 2008

Abstract

This paper presents a try at determining semantic roles with a neural network. The function learnt is a mapping between words and semantic roles. This direct mapping means faster execution speed and less data structures and stages involved than in conventional semantic role labeling software.

1 Introduction

Semantic role labeling is needed for the success of any application dealing with information extraction and question answering. Depending on the application there are constraints concerning the time needed to determine roles and the size of the textual domain the application works in. A dialog system might for example have a limited domain but need for fast answers. Contrary to this are web crawlers that deal with large amounts of data but where the time constraint can be less stressed. There is also the problem of extracting features from the data to be used in the learning algorithm. Semantic role labeling software does in general divide the problem into several layers. Each layer is a subproblem handled separately. An example is a system that uses a syntactic parse tree to extract features, and then gives these features to a classifier which does the actual semantic labeling. Between the syntactic and semantic layers there may be an arbitrary number of layers handling disambuiguation, coreferences and so on. These kind of systems will typically be slow. To learn a function mapping directly from words or syntactic labels onto semantic roles without the need of a parse tree would be much faster. This is what is tried with the neural network approach. The neural network can even be constructed to do feature extraction and by that further simplifying the problem. Advantageous with letting the neural network do the feature extraction is that only the features necessary for learning the function will be needed. Which features to use is a common problem when dealing with sparse data and machine learning.

The role labling presented in this paper is a simplified version of semantic role labling. Roles are only determined for the first predicate in sentences and the assumption have been made that if these roles can be functionally determined, all roles could be determined by further improvements and extensions of the application. There is no part of speech tagging of the input so the input need already be tagged correctly. The part of speech tag set is the penn tree bank and the semantic roles are from propbank. Because of the relaxation of the problem the application is in no way ready to use for semantic role labeling but the results give some indication that extended it could be.

2 Implementation

2.1 Data

The example data file goes through processing before it can be used for training. The original

data is row based and divided into sentences by blank lines. Each sentence is a sequence of rows, where each row holds a word, that words part of speech and its semantic roles in relation to the predicates in the sentences. Because of limited hardware resources, the neural networks have only been trained with processed portions of the original example data file.

2.2 Architecture

Each word gets labeled separately. So there need to be as many runs through the network as there are words in the sentence to label each word in a sentence. Since words are lexical and neural networks deals with real numbers, the words need to be represented with integers. Each numerical representation will be a member of a subset of the natural numbers. To make this subset controllable in size, words with any of the verb part of speech tags will get the same numerical representation. The same goes for nouns, adverbs, adjectives and numbers. Each word has a window around itself of changable size. If there aren't enough words to the left or the right then there will be out-of-sentence markers in these positions of the window. As numbers lack dimension, each word (actually its numerical representation) in the window will be translated into a one-hot-binary vector of predetermined size before being fed to the neural network. So it is a window with a one-hot-binary vector at each position that is being fed to the first layer of the network. See Figure 1 on the next page for a picture of the architecture.

First layer

The first layer extracts feature vectors from the window of one-hot-binary vectors. That means that each one-hot-binary vector in the first layer will be mapped onto one feature vector in the next layer. The second layer will be a vector of feature vectors. One way to get the projection of the first layer onto the second layer to achieve this is to share the weights between each one-hot-binary vector in the first layer. In a classic linear layer, each unit in the preceeding layer is connected to each unit in the next layer. In a weight sharing neural layer the input gets divided into frames. Every one-hot-binary vector in the input layer is a frame. These frames gets projected onto frames in the second layer. The projection is done pairwise between a frame in the first layer and its corresponding frame in the second layer. There will be as many frames in the second layer as in the first layer.That the weights in the layer are shared means that each frame mapping will be done with the same weights. By using such a layer feature vectors can be extracted. The size of the feature vectors is a parameter determining the network's functionality.

Second layer

The second layer concatenates the vectors of feature vectors into one vector. This concatenation gets fed to the third layer.

Third layer

The third layer is a classic linear layer, i.e each unit in this layer is connected by weights to each unit in the fourth layer. The number of units in the fourth layer is the number of hidden units in the network and just like the size of the feature vectors a parameter that can be changed to influence the working of the network. The third layer uses the tanh activation function to project the data onto the fourth layer.

Fourth layer

The number of units at the fourth layer is called the number of hidden units. This layer does like the third layer; a classical linear summation of its input onto the next layer. The next layer is the fifth and last layer. This layer has the same number of units as there are semantic roles. Both the third and the fourth layer are classic linear layers. The fourth layer uses the softmax activation function when projecting its output to the fifth layer.

Fifth layer

At this layer each unit (as many as there are classes) will have an activation level that is a numerical value. The unit with the greatest numerical value will also be the correct semantic role (as far as the network computes). Each unit will actually have a value that is equal to the likelihood of being the correct semantic role.

Output

Since each unit at the last layer has its value equal to the softmax at the corresponding unit in the fourth layer the values at the last layer will be like a ranking with the highest value being the most likely semantic role. It has been noticed that in most cases when the neural network computes the wrong semantic role, the unit associated with the correct semantic role will have a numerical value not far from the value of the unit with the highest value(which as said before is what the neural network thinks is the correct semantic role). The unit associated with the correct semantic role is in this way very often among the three or four highest valued units (most probable roles).

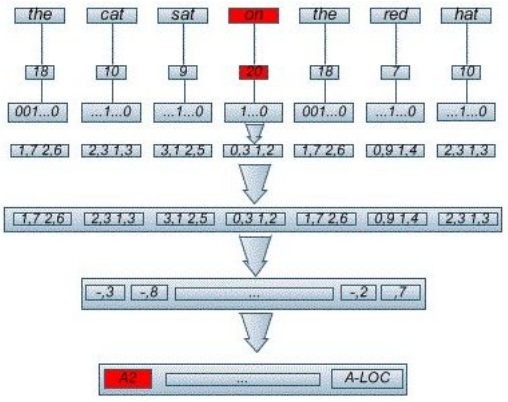

Figure 1: The arrows represents projections between layers. Current word is marked in red

3 Results

Because of limited hardware resources the neural networks could only be trained with smaller training sets. One megabyte and two megabyte of raw example data was used, this data was processed before training. Three megabyte files were tried but required too much memory resources and too long time to train, there were constant swapping between main memory and disk. Different window sizes were tried: a 7 word window, 9 word window and 11 word window. The 11 word window gave best results. The value for feature vector dimension and number of hidden units were chosen to 30 and 80 respectively. Since training a net with a one and two megabyte file takes considerable time (30 minutes to 2 hours) there wasn't enough time to experiment with the number of hidden units and feature vector size, but 80 and 30 respectively gave relatively good results. Having only the ability to train neural networks with one and two megabyte files resulted in networks that learned a classifying function for the set used during training with good accuracy, but with considerable loss in accuracy when tried on a test set disjunct from the training set. If trained and tested on the same set the accuracy was between 60-82 %, lower accuracy for smaller word windows and larger files. A disjuct test set gave an accuracy somewhere around the lower end of 40%. Even though 40% is low, as said above the correct role is often found among the top three or four most likely semantic roles so some more information than just the incorrect role exists.

The following table also gives the indication that the accuracy would be even lower with bigger files than 2 megabyte. This has to do with the model being too simple to handle sparse data from larger domains without other information than the word windows.

	1 megabyte	2 megabyte
7 word window	70% 42%	60% 41%
9 word window	78% 41%	60% 43%
11 word window	81% 41 %	66% 44%

Table 1: The element in the top row is the size of the training files before processing into the format the neural network takes, the left most

column is the window size. The first percentage in each cell is the accuracy when tested on the training set, the second the accuracy when tested on a real test set disjunct from the training set

4 Conclusions

It seems possible to learn a function that determines semantic roles using a neural network. The results for the test sets are low, but the model is also very simple. There are many extensions to the model presented here. Bigger files could possibly mean a little better or more stable results. But bigger files would also mean sparser data and that calls for some way to incorporate more information in the model than just word windows. Since every words semantic role is determined with respect to a predicate, adding position information about just this relationship would probably enhance the model.