VIRTUAL NEURAL NETWORKS

Author : Tofara Moyo

Company : Mazusa email: tofaramoyo@gmail.com

Traditional approaches to NLP use word vectors such as those from word2vec as the inputs into a network. It is known that randomly initialized word vectors make for a poor performing NLP system, and is the reason for approaches like word2vec embedding system that first discover contexts for words from a corpus and places them in relation to each other in the vector space to reflect this. This approach uses an n-gram of n positions to the left and n positions to the right of the word being vectorized as a window to calculate its context and it is known that the greater the size of n the better value the are the word vectors that are generated to NLP tasks. These word vectors are then fed into a neural network for classification or translation or as the particular task may be. We know that if we used an arbitrarily large value for the value n in the n gram model, our neural network would have an easier task in finding a mapping between the words. Imagine if instead of using backpropagation to alter the weights of a neural network in an NLP system, we used the backpropagation algorithm to alter the word vectors. We know that better word vectors make for better learning, but what if the actual learning was the change in word vectors? Then we could place the whole of the complexity of the system into the design of the word vectors through backpropagation (which takes the word vectors as inputs rather than the weights of the network) and rely on a minimal actual neural network infrastructure. Another step forward could be to actually map individual words onto individual neurons permanently and then use a Drop out procedure to represent different training cases in a neural network. Dropping out whole sentences if need be or altering the position of words and adding others. Then we could assign two values to each neuron. As inputs to a word's neuron we would take the second value of the other neurons in the sentence and feed them into that word's neuron weighted by its own first value. Then each word in a phrase would take in as inputs all the other words second values and weight them by their first value. In the diagram below, we name the first value "a" while the second "b". The "b" values in a particular sentence act as inputs to the all the words/neurons while the "a" values act as the weights to this input. Note a difference with the traditional approach is that we have only one weight, "a" for all of the individual inputs into a particular neuron, instead of having each "b" input being associated with its own weight. This apparently reduces the flexibility of the system but as we shall see we shall work around it. But the main reason for this approach is to avoid the problem that some sentences have more words than others and so more b values, together with the fact that we want a word to drop out and in with its "a" and "b" values. This system causes during training for the "a" value of the neuron/word to learn its context from the nature of its interaction with the particular "b" values it associates with, in the training examples. This is the Hebbian process at work and I believe is far more contextual than the current n gram method of establishing context. To clarify we will have as a network the neurons associated with the words in the sentence used in the training example. And if we want a binary classification we take the outputs of all these word neurons after passing them through a suitable activation function such as Relu, we put them through a single output neuron which will also have only one weight for all inputs for similar reasoning as above and we will use a suitable loss function using the output from that neuron. This output neuron will be the only permanent feature of the network while the other neurons (which are mapped to words) are dropped out and dropped in. As you can see this is a shallow network and as

such should be easy to train. We then backpropagate the loss in two phases. We first hold all the “b” values constant while learning values for the “a’s”. then, once we notice no more improvement in the system, we hold these learnt values of the “a’s” as constant while backpropagating the loss to only the “b” values. Then we iterate till there is no longer any improvement. This is similar to coordinate gradient descent. The question then arises whether we can guarantee that we can have a solution. First, we know that if we have a system of equations where the number of equations is less than the number of unknowns then we have an infinite number of solutions. Now imagine that every subset in the powerset of the English language was to be classified with a binary label. Each of those functions represents an equation, while the values of the individual words represent the variables. Since a words value is the output of its associated neuron, and that value changes with the value of the other word’s “b” values, then in every set in the powerset a particular word is found in, it will have a unique value. Since there are multiple powersets with more than one element then there must be more variables associated with the set of all mappings of the sets in this power set with a particular binary value. Meaning that the system possesses an infinite number of solutions with multiple degrees of freedom. A note is that each training example will be a different network in its own right. Note that in a phrase, the period “.” Will also be a token in the system and associated with its own neuron. This sets us up to make this more than a bag of words model. We will add one more input to each word in a phrase along with the “b” values. This will be a constant and the number of its position in the phrase. Note we take the position of the word in the overall passage and not simply in its sentence. This enables us to represent the full context of each word in any passage made of multiple sentences. This network can be extended to a graphical system, for example a cancer image scan detection system. Instead of words mapped to neurons, this time we have particular color ranges of pixels associated with their own neuron. Then we could connect all these neurons to a single output neuron whose output will be used to classify the image scan as having cancer or not. To keep the context of a particular instance of a neuron in the image we only need to enter a number associated with any method, such as listing, we use to order the pixels as an additional input to be fed in with the “b” values of all the neurons in that particular image.