List of papers read in working with ESNs and causality:

1. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions - Socher 2011

2. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank - Socher 2013

3. Efficient Estimation of Word Representations in Vector Space - Mikolov 2013

4. Distributed Representations of Words and Phrases and their Compositionality - Mikolov 2013

5. Linguistic Regularities in Continuous Space Word Representations - Mikolov 2013

6. Causal Relation Extraction - Blanco 2008

7. Automatic extraction of cause-effect relations in Natural Language Text - Sorgente 2013

8. Learning grammatical structure with Echo State Networks - Tong 2007

9. Language modeling using augmented echo state networks - Rachez 2014

10. Probabilistic Language Modeling using Echo State Networks - Rachez 2012

11. On-line processing of grammatical structure using reservoir computing - Hinaut 2012

12. Real-time parallel processing of grammatical structure in the fronto-striatal system (reservoir computing) - Hinaut 2013

13. A practical guide to applying echo state networks - Lukosevicius 2012

14. Reservoir Computing Trends - Lukosevicius-Jaeger 2012

15. A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach - Jaeger 2002

16. Echo State Networks for Multi-dimensional Data Clustering - Koprinkova 2012

17. Echo State Networks in Dynamic Data Clustering - Koprinkova 2013

18. Effects of spectral radius and settling time in the performance of echo state networks - Shishir 2009

My notes about ESN implementation:

In ESNs the reservoir is a general computing space - the sparse random connections provide a rich representational space that allow processing to occur in a non-programmatic way. It is a means of representing the input signal in a novel way and an output layer can be trained to learn regularities in this new representational space. It is a general computing space since no training is done in the reservoir, it is simply a rich representational space that can be used for arbitrary signal processing.

The reservoir is also serving as a memory, which is why it is primarily useful for tasks of a temporal nature where there are dependencies existing over time. To my mind, this seems like a good reason to use it for natural language processing because language is inherently something that occurs over time and a word in one part of the sentence will have a dependency on other parts of the sentence. Though interestingly it is not necessarily the case the the dependency will occur linearly in one direction, i.e. the dependency may not necessarily be from a present word to a previous word but could be from a present word to a future word. Dependencies exist also over multiple sentences. This may be a reason consider a whole essay as an input sequence to find dependencies over sentence divisions, and for a single sentence, multiple passes might be required or inputting the sentence as a repeated signal in order to first let the entire signal be "known" to the reservoir, and then have it discern features of the sentence.

There is some indication that RNNs (particularly ESNs) do not handle high-dimensional data very well (e.g. Hinton's ESN video and also suggested by the Kaprinkova papers in their ESNs now being useful for multidimensional data, BUT they are using only 7 or so dimensions). This is unfortunate since I need 300 dimensional input! Although, this may just be a result of not being able to use a large enough reservoir. It seems feasible to me that larger dimensional data would only require much larger reservoirs in order to be of use in expanding the input in the representational space of the reservoir. Other than using lower dimensional word vectors, another approach might be to consider imposing a more structured architecture on the reservoir. That is, instead of just one monolithic reservoir with sparse random connections, maybe have many smaller reservoirs with sparse random connections, and then introduce connections between those smaller reservoirs. Or better yet, induce "small world" structure so that you have computational hubs. This would be more in the spirit of brain architectures, though this might make it much more difficult to get a linear readout of the reservoir.

It is possible that a sentence represented as a sequence of word vectors being fed into the reservoir one at a time will not be long enough to produce the desired dynamics in the reservoir, especially when a sentence is very short. Perhaps there can be a repeater function that will take a sentence sequence and repeat it N times so that you're sending pulses of the sentence into the reservoir, the idea being that after the reservoir has received the sentence pulse a number of times it will settle in the dynamics for that particular sentence and sort of resonate, and hopefully that resonating will allow for a targeted classification of the desired aspects of the sentence, whatever it was trained to recognize. A one-off pulse may not be enough, might need to consider pulse trains.

For all intents and purposes, I am treating a time step as a training instance and the dimensionality of the word vectors will be the number of features for an instance. Even though it is really more proper to treat a single sentence as a training example, Ridge Regression is operating on a matrix of concatenated values over time and the acquisition of the correct readout weight matrix relies on computing those weights in relation to the full time-series, not based on each sentence presented sequentially. This still allows for pulsing of sentences if needed, just introduce the pulse sequence of a sentence concatenated on to a chain of pulse-sequences.

May want to try RidgeCV as it will go through a selection of the best regularization parameter with GridSearch

Important to note that the reservoir will need to time to respond appropriately to signals. It seems that I would also have to introduce "cleansing" periods between input signals so that the network has appropriate time to adjust to the current input and process it appropriately. Maybe consider sending a signal in as pulses (i.e. repeating same signal one after other as an input train) so that the reservoir begins to resonate for that signal and will settle into an oscillatory state that is representative of that input that can then be read out.

There are a lot of variations on how the network is constructed, depending on how interconnected the layers should be. The reservoir only can be connected to the output nodes, the inputs can also have connections going straight to the output, the output nodes can have direct feedback connections on themselves, there can be feedback connections going back into the reservoir. It seems like the out-reservoir feedback connections are a separate entity (and randomly assigned) while the in-out, reservoir-out, and out-out connections are all thrown into the output weight matrix that is trained. For simplicity I will just start out with reservoir-out and out-reservoir and then can complicate the connections later. Both Lukosevicius and Jaeger also include in-out.

There is a slight peculiarity when scaling for the reservoir weights to the desired spectral radius. In the scipy.sparse.linalg package, the function eigs can only reliably generate eigenvalues when they deviate sufficiently from zero, namely when the computed spectral radius (max(abs(eigs(w)))) is greater than or equal to 0.01. The eigs function works iteratively, so it seems that when all of the values are close to zero it cannot reliable compute them, and the result then is that when you scale the weights to a target spectral radius, it is unlikely to actually be scaled to that value since the computed spectral radius jumps around in value from one execution to another. -- I have resolved this in the initialization of the reservoir weights by simply having a loop that re-initializes the weights until it gets a set with spectral radius above 0.01. Having a SR below that threshold is reasonably rare so the loop will likely only execute a couple iterations at most.

It seems wise to reset the the reservoir somehow between each run on a sentence (unless classifications are being made for an entire corpus, in which case it might be able to pick up on structure across sentences), otherwise the network might erroneously "believe" there exist dependencies across sentences when in fact the sentences have nothing to do with one another. How that resetting should occur I am not sure: either set the reservoir activations to 0 or random values; I can try both and see if there is any appreciable difference. Can also consider a repeating signal for each sentence, in case the dynamics don't settle in time for the single sentence, but again that might introduce dependencies that shouldn't exist.

Will have to make decision about how to determine final classification of a single sentence. The papers distinguish between two types: online learning and end-of-sentence classification; the online learning involves outputting a value over the duration of the sentence so as to serve as a prediction for the classification that could change over time, while the end-of-sentence classification is output only once after the entire sentence has been seen. I think the latter seems most sensible since the sentences are reasonably short and what matters is the classification of the sentence as a whole - it should be that the entire sentence must be seen as key relationships might exist only at the very end of the sentence, and in the end it is that final classification that we care about, not its ability to make premature predictions. Still, even in the latter case there are decisions to be made.

I am now structuring the data as one list, that list containing training (or testing) instances, each of which is a list containing two numpy arrays, the first being the inputs over time for that instance and the second being the targets over time for that instance. I decided to use lists because that seems most appropriate for Python and when trying numpy arrays, strange indexing things started occurring. The logic behind doing things this way is that even though ESN networks are primarily for continuous signals, my data is coming in as chunks of data corresponding to individual sentences. I would therefore like the network to process it as such (signal pulses) instead of one continuous stream. The plan is to drive the network with each sentence, each time resetting the reservoir initially to either zero or random activity, and then collect the reservoir states in an ongoing collection array. This collection array is what will be used in ridge regression and will have all activity except the initial zero point between each sentence. The target states will also be collected and this can save on space as I then only have to pass around a single vector for a sentence classification instead of a redundant signal stream. This allows each sentence to determine its own reservoir activations and resulting output without being biased by activations from previous sentences, but the regression can still be performed on the continuous stream without being explicitly broken up by a bunch of zero point activations. This approach seems to be consistent with both Jaeger (who has state collection matrices for reservoir activations and targets after a certain time, but he has continuous signals) and Hinaut (who resets the reservoir after each sentence application).

There is motivation for me writing my own ESN as opposed to using some of the out-of-the box implementations from Oger and other toolboxes. Using the ESN for sentence classification instead of a traditional signal requires specific structuring of the incoming data and also a specific way of processing the data in terms of how to compute and collect network states. It is not as simple as just concatenating a bunch of signals and feeding them to a standard ESN, decisions have to be made that are not typical for an ESN application.