# NEURON VECTORS OBTAINED WITH WORD2VEC

I transformed data having 20ms intervals into 200ms sentences as follows. I used neuron numbers (ids) as 'words' and list of neuron ids that spiked in a 200ms interval as a 'sentence'. Each word represented one spike, and if a neuron spiked more than once, I included it several times in the sentence. If there were no spikes in a 20ms interval, I included "_" in the sentence to mark a pause in spikes. Later, I removed some of those pauses so that there are no more than two consecutive "_".

Example of a sentence:

```
['_', '_', '51', '51', '_', '_', '55', '49', '_', '35', '4', '4', '56', '35']
```

Next, I built a word2vec model using the 'sentences' as input. The output were 63 'word vectors' (numeric vectors of length 200), each representing one neuron. In order to check if the neuron vectors were any good, I used word2vec's most_similar() method, which outputs a ranked list of n most similar neurons together with their similarity values.

Example of most_similar():

```
In [371]: model.wv.most_similar(['3'], topn = 5)
Out[371]:
[('21', 0.8916540741920471),
 ('22', 0.5335803031921387),
 ('14', 0.5274465084075928),
 ('46', 0.4985032379627228),
 ('17', 0.4570053517818451)]
```

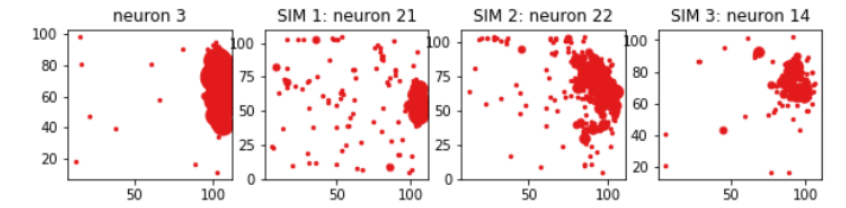
Here, we can see that neurons most similar to neuron 3 are 21, 22, etc., with similarity of neuron 21 being considerably higher than others.

In order to see if the receptive fields of the 'most similar' neurons are indeed similar, I plotted their receptive fields.

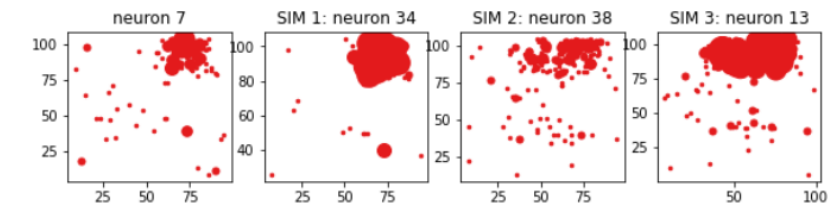
# PLOTTING RECEPTIVE FIELDS OF MOST SIMILAR NEURONS

Below are some example plots of receptive fields of some neurons that were considered as similar by word2vec's most_similar() method . Here, the leftmost plot is the neuron given as input. The rest ("SIM 1", "SIM 2" and "SIM 3") are three neurons whose word vector representations are closest to that of the input neuron, with "SIM 1" being most similar. It seems that especially "SIM 1" indeed does look similar and sometimes also the others (although to lesser degree), which seems to indicate that my word2vec model had indeed learnt something.

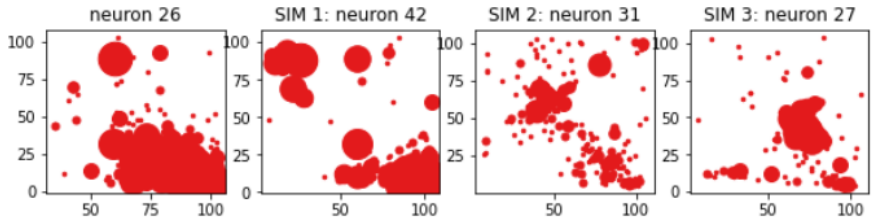
```
plot_most_similar(3, similar_df)
```



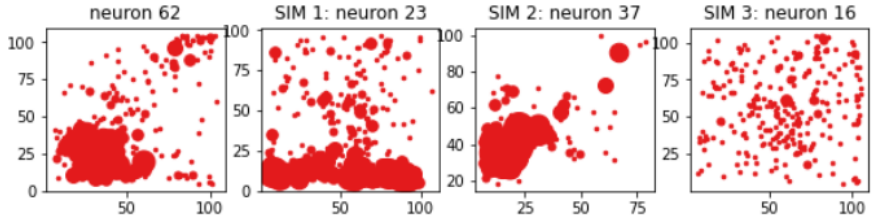

```
plot_most_similar(7, similar_df)
```



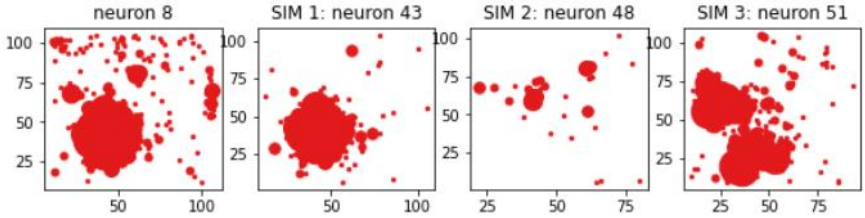
```
plot_most_similar(26, similar_df)
```



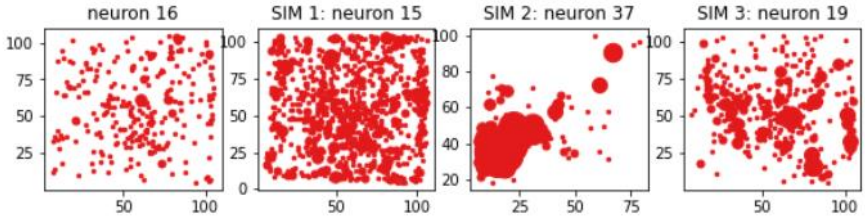

```
plot_most_similar(62, similar_df)
```



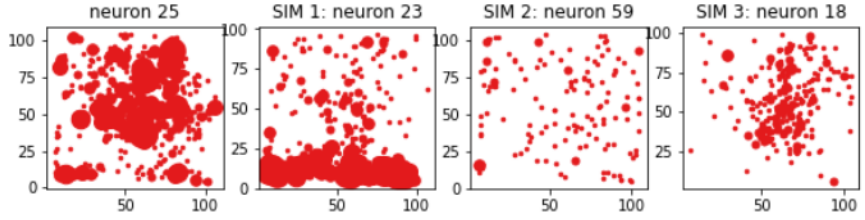

```
plot_most_similar(8, similar_df)
```




```
plot_most_similar(16, similar_df)
```




```
plot_most_similar(25, similar_df)
```



## A REGRESSOR ON TOP OF NEURON VECTORS TO PREDICT LOCATION:

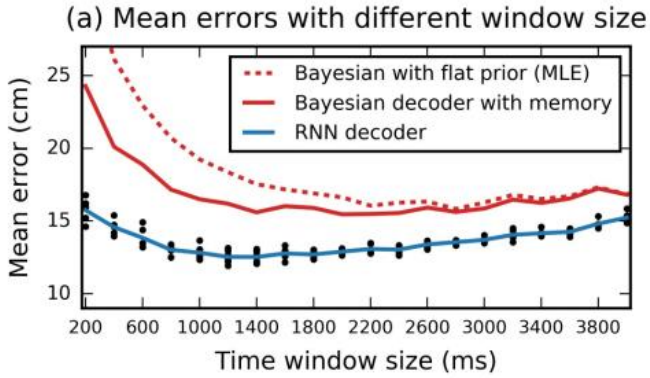
Next, I averaged neuron vectors of each sentence to get sentence vectors. I used first 4400 timesteps for training and last 1010 timesteps for testing. I only used the sentence vectors as input, i.e. I did not include location, direction, speed nor anything else in training data.

Then I used the sentence vectors as input to a Random Forest Regressor model (with multioutput, in order to get predictions for both coordinates), using min_samples_leaf=5 and otherwise default parameter values.

It must be noted that this model includes no memory. Each prediction was made solely based on spiking data in one 200ms interval, without looking at previous intervals. The 200ms windows were not overlapping. I only trained one model, without cross-validation nor parameter tuning, so the results are preliminary. Still, they serve as some indication. The perfomance of the model was:

**`RandomForestRegressor: mean error (in cm):`** `26.96`

For comparison: results form your article:




## CONCLUSIONS:

When comparing results obtained with 200ms time window in the article, it is clear that Random Forest performed considerably worse than the best model (RNN) in the article. However, it performed better than the worst model ("Bayesian with flat prior") in the article, and only slightly worse than "Bayesian decoder with memory". From this it can be concluded that using word2vec in neuroscience may not be an entirely bad idea and is worth exploring further.

## NEXT STEPS:

The article showed that 200ms was not a good time window when using spike counts as input. It is quite likely that also with word2vec vectors as input, actually a larger time window could give better results. Notably, ca 10% of test data in my Random Forest model consisted of 200ms intervals were *no spikes occurred at all*, so predictions were basically generated without any input information. Using larger time window is one way to avoid this situation. Also, of course a lot of parameter tuning could be done - both for regression model, word2vec model as well as in the way the sentences were formed in the first place (I tried to make reasonable choices, but there are also other ways this can be done).

But most importantly, it might be interesting to try to use the word2vec vectors as input to some more advanced models that also use memory of past activity, like LSTM (and maybe also Bayesian). It might be interesting to compare spike counts and word2vec vectors by building the exact same models with both types of input, using same time windows, train/test splits, hyperparameter values etc. At the same time, even if word2vectors turn out to be not so good with place cells data and location prediction problem, it is possible that they may be better suited to some classification task and input from some other types of neurons.