Simulating Language: Lab 8 Annotations

Question 1:
We can study the effect of prior biases on the learning of a single individual in two different ways: either by creating a data set and a prior ourselves and feeding these into the learn() function (but remember that the learn() function gives a log probability as output, so we have to use exp() to turn this into a normal probability), or by using the iterate() function with just 1 generation. I will use the second alternative here.

What we want to do here is to see how, given enough data, learners with very different biases can still get to the same conclusion as to what the 'real' probability of word 1 is. As you should have noticed when plotting the priors, all values of alpha *below* 1 create a U-shaped curve (bias for regularity, favouring a very high or very low frequency of word 1), an alpha of *exactly* 1 creates a straight line (unbiased/uniform prior) and all values of alpha *above* 1 create a bell-shaped curve (bias for variability, favouring a frequency of word 1 close to that of word 0). For now let's explore the alphas 0.1 (regularity bias), 1 (uniform prior) and 5 (variability bias).

We can use the usage example given at the top of bayes1.py and change the input so that we run a simulation with a uniform prior, a data set of 10 words in which word 1 occurs 7 times, and just 1 generation:

```
In [1]: pW1_by_generation,data_by_generation = iterate(1,10,7,1)
```

The value for pW1 that the learner has inferred is now stored in the variable pW1_by_generation:

```
In [2]: pW1_by_generation
Out[2]: [nan, 0.725]
```

The first value in this list is 'nan' (Python code for 'not a number') because this is the pW1 for generation 0. The second value is the pW1 that our learner has inferred. As we can see, this estimate is not very accurate (since we know that the real probability of word 1 in our data is 0.7). If we run this same simulation a couple of times we will see that the outcome is always around 0.7, but still quite variable (probably varying between about 0.5 and 0.9). (Have a look at the learn() function and the log_roulette_wheel() function to see where this variability comes from.)

Now to explore the effect of the prior biases, let's do a little for loop to see what happens with an alpha of 0.1:

```
In [3]: for r in range(10):
            pW1_by_generation,data_by_generation = iterate(.1,10,7,1)
            print pW1_by_generation
```

This should give you numbers that push closer towards 1.0. This is what we expected given the bias for regularity. You can do the same for alpha=5, which should give you numbers that are closer to .5, as we expect based on the variability bias.

So when the learner learns from a small set of data, we can see the effects of the different biases. But now let's see what happens if we give the learner a very large dataset, for instance 1,000 words, with 700 occurrences of word 1.

```
In [4]: pW1_by_generation,data_by_generation = iterate(1,1000,700,1)
```

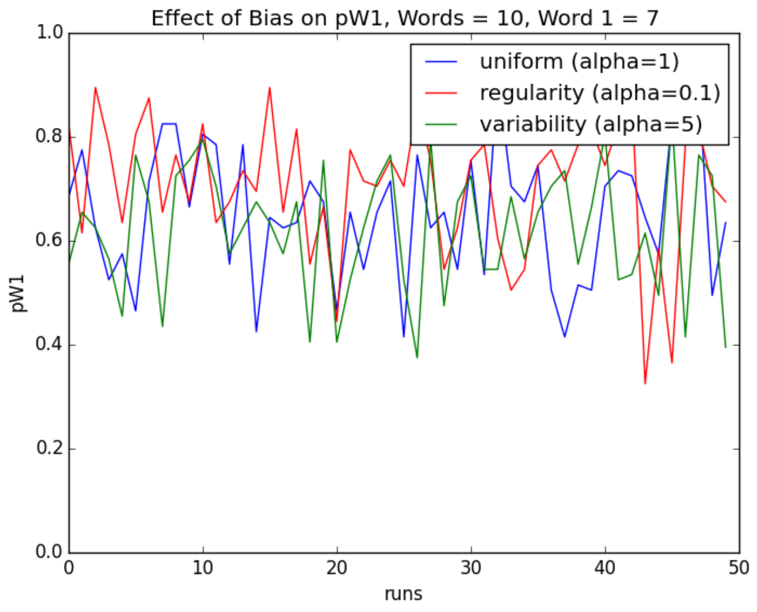
What you should find is that we this data set, the three different alphas we explored above come up with very similar results.
We can also put this into a nice plot by doing:

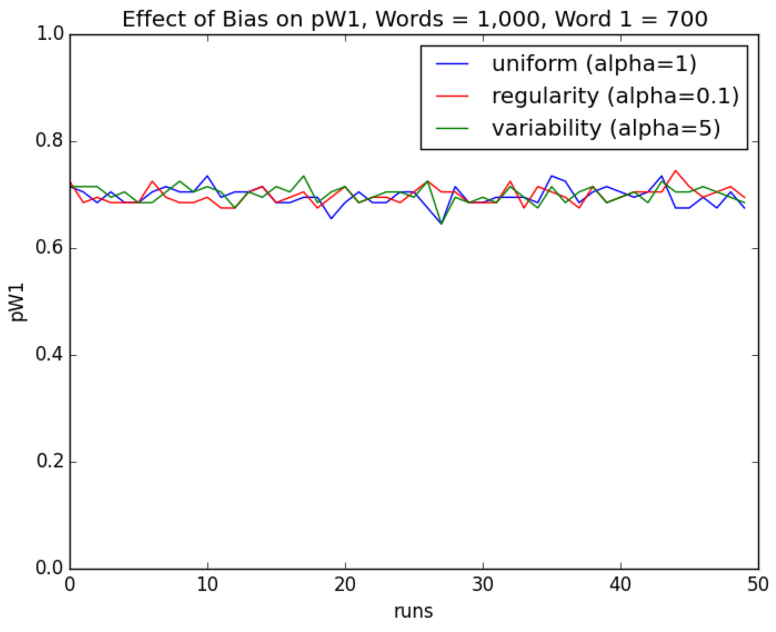
```
import matplotlib.pyplot as plt

## Doing several runs and saving the data:
uniform_accumulator = []
regularity_accumulator = []
variability_accumulator = []
for r in range(50):
      pW1_by_generation_uniform,data_by_generation = iterate(1,10,7,1)
      pW1_by_generation_regularity,data_by_generation =
      iterate(.1,10,7,1)
      pW1_by_generation_variability,data_by_generation =
      iterate(5,10,7,1)
      uniform_accumulator.append(pW1_by_generation_uniform[1])
      regularity_accumulator.append(pW1_by_generation_regularity[1])
      variability_accumulator.append(pW1_by_generation_variability[1])

## Plotting the results:
plt.plot(uniform_accumulator, color='b', label="uniform (alpha=1)")
plt.plot(regularity_accumulator, color='r', label="regularity
(alpha=0.1)")
plt.plot(variability_accumulator, color='g', label="variability
(alpha=5)")
plt.ylim((0,1))
plt.xlabel("runs")
plt.ylabel("pW1")
plt.title("Effect of Bias on pW1, Words = 10, Word 1 = 7")
plt.legend()
plt.show()
```

(The code above shows you how to do it directly in your code, but you can also do it in the interpreter by first calling figure() and then doing all the above but leaving out the 'plt.' prefixes.)
Now we can do this for our 10 word data set and for our 1,000 word data set, and see what happens:





Although the differences between the priors in the first plot are not super clear, we do see that the red line is generally a bit higher up than the green line, and that the blue line varies freely around 0.7. In the second plot we see no differences at all. With all three priors the learner gets to a pretty good estimate of pW1 every time.

Question 2:

To do this, we can use the usage example given at the top of the code:

```
In [11]: pW1_by_generation,data_by_generation = iterate(0.1,10,5,10)
```

And we can visualize how the inferred probability of word 1 changes over generations using:

```
In [12]: plot(pW1_by_generation)
```

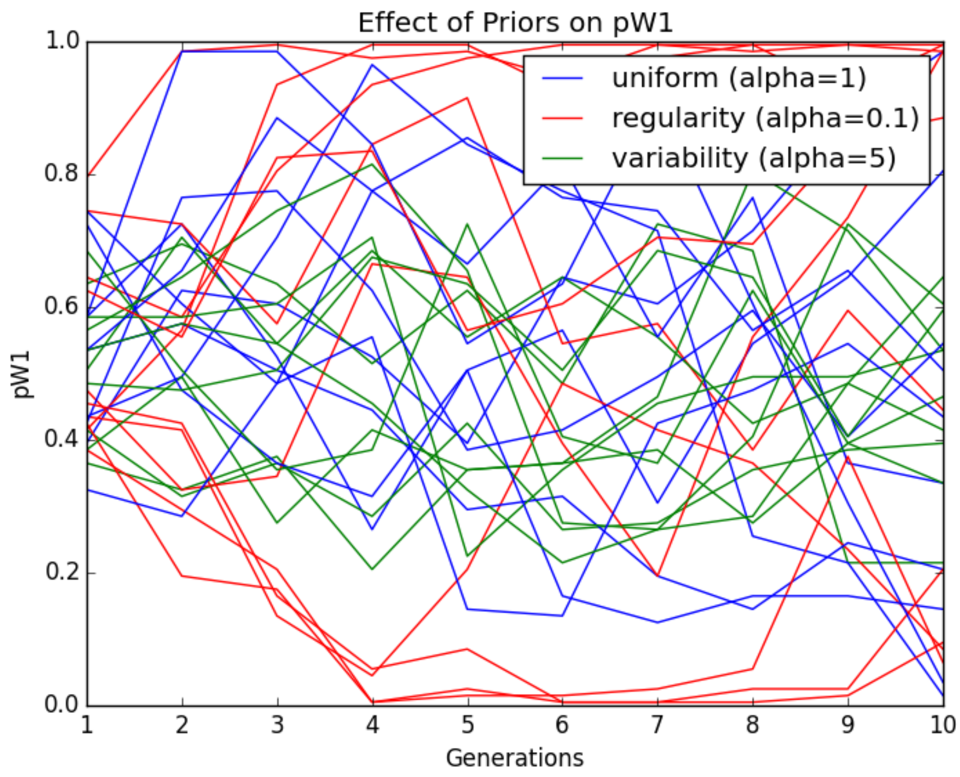
But we know that there will be variability in the outcome, so let's do a couple of runs with each different alpha:

```
import matplotlib.pyplot as plt

for r in range(10):
      pW1_by_generation_uniform,data_by_generation = iterate(1,10,5,10)
      pW1_by_generation_regularity,data_by_generation =
      iterate(.1,10,5,10)
      pW1_by_generation_variability,data_by_generation =
      iterate(5,10,5,10)
      plt.plot(pW1_by_generation_uniform, label="uniform" if r == 0
      else "", color='b')
      plt.plot(pW1_by_generation_regularity, label="regularity" if r ==
      0 else "", color='r')
      plt.plot(pW1_by_generation_variability, label="variability" if r
      == 0 else "", color='g')

plt.ylim((0,1))
plt.xlabel("Generations")
plt.ylabel("pW1")
plt.title("Effect of Priors on pW1")
plt.legend()
plt.show()
```

This should give you a plot that looks something like this:

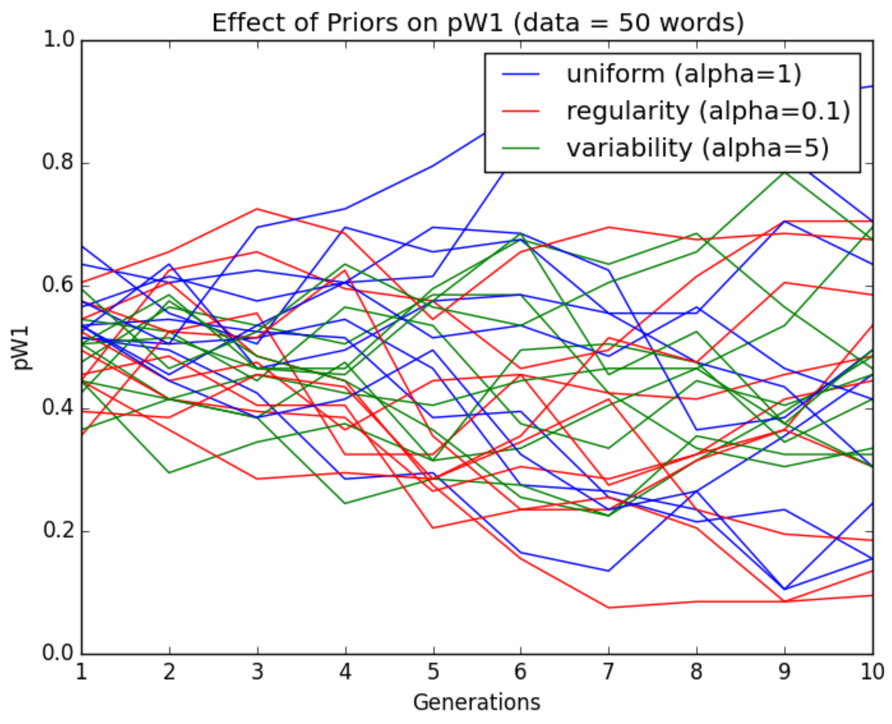


This is a very messy plot. (Have a think about why that is. What is it we see here? What is the *real* value of pW1? What do we expect to happen to pW1 under different priors? What would happen if we would have more data or more generations?) We do however see something different happening under the different priors. With a uniform prior there is quite a lot of variability in what happens to pW1. With the regularity bias pW1 gets pushed towards the extremities (either p = 0.0 or p = 1.0), whereas with the variability bias pW1 remains in the middle (around p = 0.5). These differences are hardly visible at generation 1, but become more apparent over a couple of iterations.

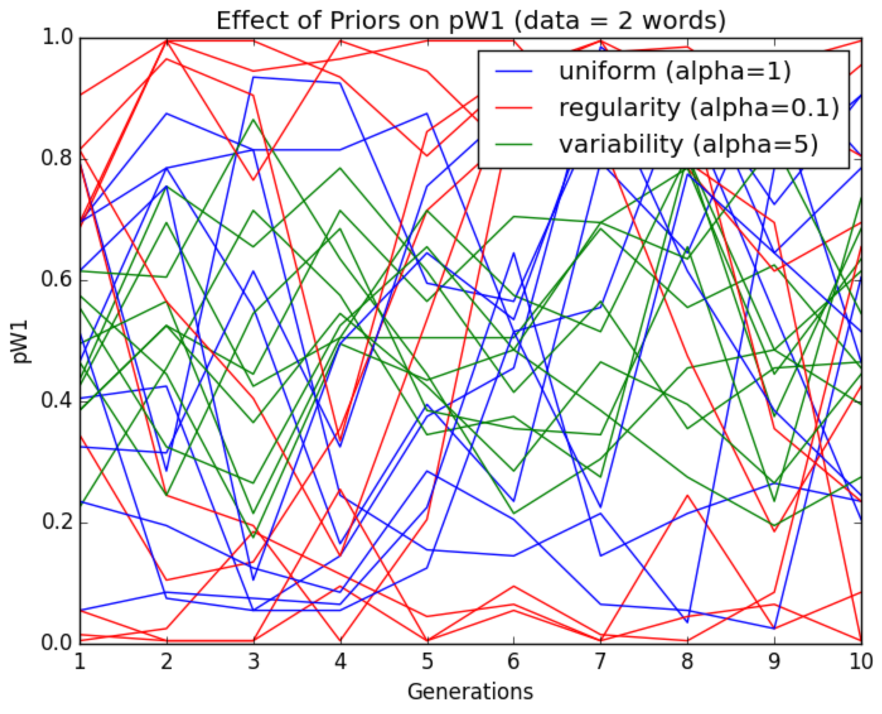
To further explore the outcomes of different learning biases you can make some even nicer plots using the bonus code that Kenny has kindly provided us with (download bayes_histograms.py from the website; the new code for plotting is at the bottom).

Question 3:

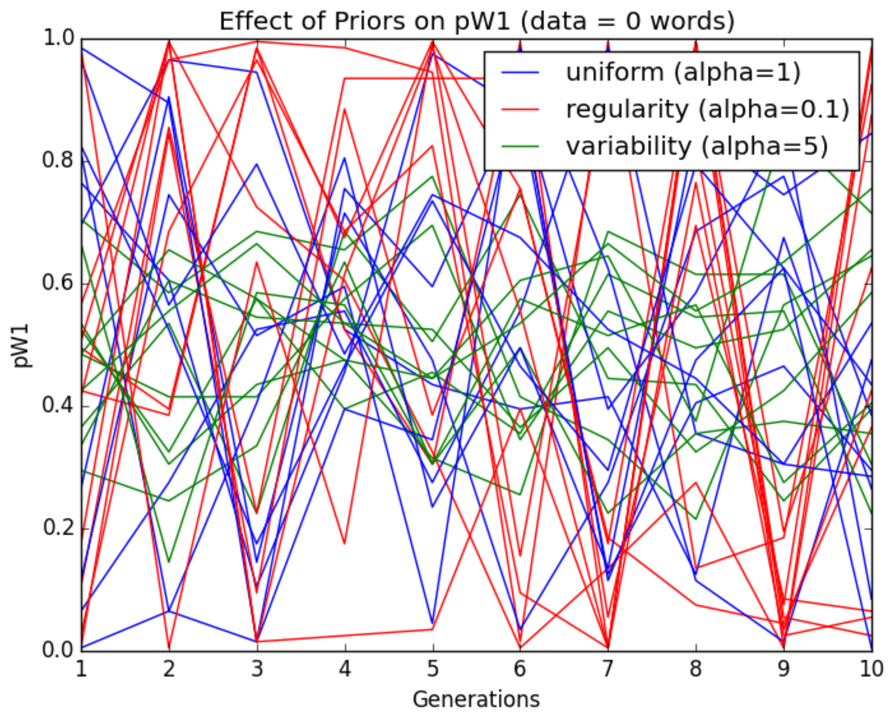
Let's make the same plot as above and see what happens with different amounts of data. We can do this by changing the value of 'n_productions' that we give as input to the iterate() function. But note that if we do this we also have to change the value of 'starting_count_w1' to make sure that the input for generation 1 still consists of 50% word 1 and 50% word 0. With 50 productions per generation (and 25 occurrences of word 1 in the initial data set), we get the following graph:



With only 2 words being transmitted between generations we get something like:



And with no data being transmitted at all we get something like:



So we can conclude that the more data being transmitted from generation to generation, the more gradual the changes in pW1. The last simulation where no data is being transmitted at all corresponds to a situation where the signalling behaviour of the agents (i.e. how often they produce word 1) is somehow innate and doesn't depend on the behaviour they observe in the previous generation.