Psychology 209 – 2017

Homework #2 – due Jan 24, 2017 (version 2, Wed, Jan 18, 10:27 am)

The purpose of this homework is two-fold:

1. To get you set up with the PDPyFlow software and some of the tools you will be able to use in this and later homeworks. The key elements here are

The PDPyFlow software background, philosophy and current status

The management of your copy of the PDPyFlow software

The minimal PDPyFlow user interface and graphical tools we have provided

2. To allow you to explore several key concepts relied on in our conceptual framework for understanding the neural basis of perception. The key concepts here are

Generative model of an environment and its instantiation in a neural network

How a neural network samples from the probability distribution of the generative model

How this approach allows us to:

(i) exploit conjunctions of constraints rather than simply their additive combination and  
 (ii) exhibit sensitivity to the implicit statistics of the training set

The first two pages below are preliminaries:

* Background on PDPyFlow
* Setting up to run your simulations

This is followed by the Homework handout itself, which builds on McClelland (2013), and then takes you into the actual simulation and questions you are to answer there

The last page is a User’s guide for the software tools used for this assignment

**PDPyFlow: Background, Philosophy and Current Status**

In the late 1980’s I developed the PDP software, implementing models in the PDP volumes. This software was written in C, ran on the simple PC’s available at that time, and used a 24 x 80 character window for both the user interface and for visualization. The software was re-implemented in Matlab in the late 1990’s. Today, we have moved to Python and Tensorflow for two reasons: First, these tools are open source resources while MATLAB is a proprietary product. Second, Tensorflow, which is built on top of Python, is the strongest tool for neural network research, allowing us to use GPU’s without thinking about them. Our approach is intended to allow you to begin to work with this cutting edge computational toolkit even if you have no prior experience or background with neural networks.

PDPyFlow is currently a work in progress. Alex Ten ([alexten@stanford.edu](mailto:alexten@stanford.edu)) a friendly and capable young man from Kazakhstan is doing the implementation. If you encounter bugs, contact Alex.

**Managing your copy of PDPyFlow**. Your account on the PDP lab computer will come pre-configured with a copy of the modules of the PDPyFlow software. Currently there are three application modules and some support software. The application modules are the MIA module, the FFBP module, and the RNN module. For the first homework, we will be working with the MIA module.

You will have your own personal copy of the software in your own directory on the lab’s computer. This will allow you to make modifications on top of the existing software and to save your changed versions. An advantage of this is that you have total control – and, if you are a great developer, you could even contribute to the enhancement of the software by creating or extending tools. Given that the software is still being developed, you may need to update your copy before you begin a homework assignment. In general, you should make sure you have the latest copy of everything by executing the ‘git pull’ command once you are inside the PDPyFlow directory (currently just called PDP). If you find yourself wishing to make extensions to the code, please consult with Alex.

**Minimal user interface with helpful graphical tools**. Many software systems rely on a GUI. The advantage of this is that you don’t have to remember long lists of parameter and command conventions. The disadvantage is that this approach invariably hides the inner structure of your software from you. With PDPyFlow we have opted to try to make everything accessible to you at the cost of requiring you to type commands at the command prompt. This is facilitated if you use a text editor to inspect the code so you can see how things are named. That way you can even make modifications to defaults used in the code so you will only need to type a minimum of commands at the prompt at run time. For the first homework, we provide a simple user’s guide at the end of this document, so editing files will not be necessary.

As a compromise to allow you to run exercises without having to learn arbitrary things, we have set things up to make basic use simple. For the first homework, you will need only a few simple commands. Extension then is open to you – though for that, you’ll have to do some digging for yourself. Detailed documentation of some things is unfortunately not yet available.

On the other hand, because there are particular quantitative simulation results and values that we want you to be able to track, we have created visualization tools that make the values we want you to understand accessible in a window that pops up and allows you to navigate around in your results. We’ll describe the version of this window that we will be working with below.

**Setting up to Run your own Simulations**

Psychology 209 uses a small server with GPUs owned by the PDP lab. Use these instructions to access the lab computer to do the exercises for the second and subsequent homework. Contact Steven (sshansen@stanford.edu) if you encounter difficulties.

Logging onto the server:

1. Must be connected to the Stanford network

-- if working from off campus, or from a laptop, you may need to use the [VPN](https://uit.stanford.edu/service/vpn)

1. Mac users:
   1. Open the terminal
   2. ssh YOUR\_SUNETID@171.64.40.39 -Y
2. Linux users:
   1. ssh YOUR\_SUNETID@171.64.40.39 -X
3. Windows users
   1. onetime setup:
      1. Download and install [Putty](http://www.putty.org/)
      2. Download and install [Xming](https://sourceforge.net/projects/xming/)
      3. Launch Xming
      4. Launch and Configure Putty
         1. Type YOUR\_SUNETID@171.64.40.39 into the hostname field
         2. In the “ssh” tab, find the X11 field and select the remote X11 authentication protocol with MIT magic cookie
         3. Type a name for this configuration (e.g psych209), in Saved Sessions and save for future use
         4. Load saved configuration and press open
   2. Regular use:
      1. Launch Xming
      2. Lauch Putty, Load saved configuration, press open
4. When it asks for the password, it is “password”
   1. Immediately change your password with something more secure.  
       Enter passwd at the command prompt

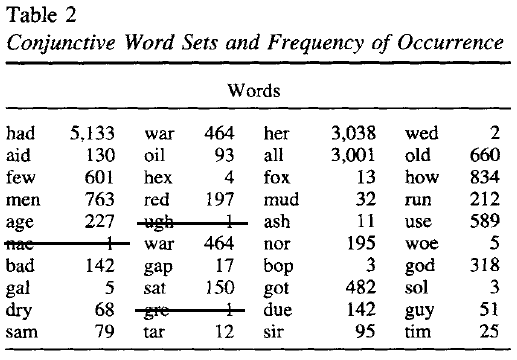
You will be prompted for your existing password, then for your new password

1. The PDP folder contains your copy of the code needed for the homework exercises
   1. Use the “cd” command (change directory) to enter this directory: cd PDP
   2. To ensure you have the most recent updates, use the git pull command from inside the PDP directory before you start your using the software
2. This system has Tensorflow installed, and has two GPUs that Tensorflow uses. Please don’t use Tensorflow until we’ve gone over it in class, as by default a single user will use an entire GPU (not good unless authorized!)

**Homework Background:   
Generative model of an environment and its relation to the state of a neural network**

We assume here that you have finished reading the McClelland (2013) paper that describes the relationship between neural network models of perception and Bayesian inference. Here we briefly summarize a couple of the key points and organize them to allow you to focus on the right concepts for the homework.

We envision a world of visual stimuli that are generated from a vocabulary of 36 different words. The words are shown in the table below, and come from an experiment by Movellan and McClelland (2001), and some of the materials were used as examples in McClelland (2013). Note that three of the words from M&M (01) were not used (the ones with a line through them), and the word *WAR* occurs in the table twice but is only considered to be a single word in the model. Although the words have different frequencies in the English language (and these are shown in the table), we do not incorporate this variable into the generative model, for simplicity.

In the generative model for this situation, stimuli are created by first selecting a word from the vocabulary (each word is selected with equal probability), then generating a set of three letters conditioned on the word and finally generating a set of features conditioned on the generated letters. For example, on a given trial we might select the word AGE, then select the three letters H, G, and E, and then select features based on the letters. A parameter of the model, called ‘OLgivenW’ determines the probability under the model that we select the correct letter in each position given the word that was chosen under the generative model. Let’s understand this since it can be confusing. Let’s say OLgivenW is 10. What this means is that the odds are ten to one that we will select the correct letter rather than a given incorrect letter. For example, we are 10 times more likely to select A in the first position than we are to select H, according to this generative model. 

With this value of OLgivenW, we will actually choose one of the incorrect letters more often than the correct letter, since there are 25 incorrect letters. The probability that we will chose the correct letter is OLgivenW/(OLgivenW + 25). So if OLgivenW is 10, the probability that we will select the correct letter is only 10/35, or about .286. You should be able to see that the probability that we would choose H in position 0 is .0286.



We can already calculate the probability under the generative model that we would have selected the word AGE and then generated the letters H, G, and E. We assume conditional independence, that is, that the letters are chosen conditional on the word AGE but independently for each position, so their choice is conditionally independent given the word AGE. The calculated probability is 1/36 (the probability of selecting AGE) times .0286 (the probability of generating h in position 0 given age), times .286 (the probability of generating g in position 1 given age) times .286 (the probability of generating e in position 2 given age. This turns out to be a pretty small number: 6.4788e-05, or .000064788. What is the probability that we would have selected AGE and then generated all three letters correctly? What is the probability that we would have selected ‘age’ and then generated the letters H, X, and K? You can determine these additional probabilities using OLgivenW and using the assumption of conditional independence. (Answers are 6.4788e-04 and 6.4788e-07. Be sure you see why).

Next, we assume that the features that a participant sees are also probabilistic, with the features in each position chosen independently conditional on the letter. Another parameter governs this: it is called OFgivenL, and by default we also chose a value of 10 for this parameter. That means that under the generative model, we are 10 times more likely to choose the value of a feature that is correct for a given letter than the value that is incorrect for that letter. For example, if the letter we have generated for position 0 is H, then we are 10 times more likely to specify that the vertical segment at the lower left of the first letter position is present (since this feature is present in the letter H) than to specify that it is absent. We are also 10 times more likely to specify that a horizontal feature at the top is absent than to specify that it is present.

With these given assumptions, let’s consider the probability that all of the features of a given letter will be generated correctly given the letter under the generative model. First, we note that the probability that a given feature is generated correctly is OFgivenL/(OFgivenL + 1), since there’s only one incorrect value for each feature. If OFgivenL is 10, this number is 10/11, or .9091. There are 14 of these features, and each is generated conditionally independently, so that the probability that all the features would be generated correctly is only moderate– it is (10/11)14, or .2633. What is the probability that all the values of the features will be correct, except for the bar at the top? It is (10/11)13\*(1/11), or .02633 – the ratio of the probabilities of these cases is 10 to 1.

Under all of the assumptions of our generative model, and the given values of OLgivenW and OFgivenL, the probability that all the letters and all the features of our selected world would be generated correctly is actually quite low. But, on the other hand, correct letters are far more likely than incorrect, and the same is also true of features. Thus if we believe this sort of generative model underlies our experiences of the world, this will lead us to use context to interpret the inputs that we receive.

**Our neural network.** We now consider the neural network that instantiates the above generative model. In this neural network, we have a set of 36 word units, and we can set the bias weight for each equal to the log(1/36), since each word is equally probable. In addition, we have weights in the network from words to letters, and we can set these weights as follows: The weights from the word level to the letter level could be set equal to log(OLgivenW/(OLgivenW+25)) for the correct letter in each position and to log(1/(OLgivenW+25)) for all the incorrect letters. However, we can simplify here because when we compute we will be using the softmax function. Under the softmax function, the only things that matter are the ratios of corresponding probabilities, rather than their absolute values. For example, we can ignore the bias weights at the word level, because their ratios are all equal to 1, and log(1) is 0. Similarly, we can set the weight to the correct Letter from a given word to log(OLgivenW) and the weight to each incorrect letter from a given word to be equal to log(1) – ie we can let the incorrect weights all simply be equal to 0. Also, at the letter level, the weights between the letters and their correct feature values are given by log(OFgivenL), and the weights between letters and their incorrect feature values are again log(1), and thus are just left equal to 0. ***Our network then simplifies to one in which there are positive weights between mutually consistent units across levels and there is competition via the softmax function between units within a level.*** While these weights are defined by the generative model (i.e., top down), they are used in both directions in our computations. Between layer influences are exclusively excitatory (as in the brain, where connections between different regions in the brain are only excitatory), while inhibition is needed only to resolve competition among neurons within layer (again, in the brain, inhibition is local within a small region of cortex, similar to the way it is in the model).

In summary, the neural network model we will use to instantiate the generative model described above consists of a set of 36 word units – one for each of the words in the above table – and three pools of 26 letter units. Each pool of letter units has connections to its own set of 14 sets of mini-pools of feature level units, although in the code these are just 28 values of an input vector, since they are specified as inputs by the modeler. The letter pools are indexed 0, 1, and 2, because in Python and Tensorflow we count from 0. There are no bias terms in the network, but there are weights that link word units to the letters that are actually in these words, and weights that link letter units to the values of features that correctly characterize a letter (thus there’s a weight from A to ‘crossbar at the top present’ and a weight from H to ‘crossbar at the top absent’) There are only two free parameters that determine the values of the weights: Weights between words and their letters are equal to log(OLgivenW) and weights between letters and their features are equal to log(OFgivenL).

**Processing in the network**. Our neural network is thought of as sampling from the probability distribution of possible states than might have produced a given pattern of experienced features. For concreteness, we continue with the example of the input ‘hge’. What might have generated this under our generative model? Well, there are two particularly likely possibilities. One is that the word was AGE, but the first letter was mis-generated as an H rather than an A, and then the correct features of H were all generated. The other is that the word was AGE, all three of the correct letters were generated, and then the feature at the top of position 0 was incorrectly generated. If OLgivenW and OFgivenL are equal to each other, these two possibilities are actually equally likely. Either way, we should end up thinking that the underlying word was likely to be AGE, although the letter that gave rise to the features might have been either A or H.

Processing takes place as follows. We first set the features to the values specified in the input, and we make sure no units are already active at either the letter or the word level. Then, processing takes place over a series of time steps (default: 20). In each time step, we first update the units at the letter level in each position. We think of this as a parallel computation: For each letter in each position we calculate its net input, which is the number of features of each letter that are specified correctly in the input, scaled by log(OFgivenL). On timestep 0, there is no word level activity to affect this computation, so that’s all there is to it. In our example, the number of such features is 13 for A in the first position and 14 for H in the first position. The resulting net inputs are put through the softmax function and one letter is selected to be active with the probability specified by the output of the softmax. In the model, this is a discrete event. However, we actually run a large batch of independent samples (default: 1000) of such computations and we report the proportion of times each letter is selected across the batch. This is a random process, so that the value should only approximate the true value (If the true probability is *p*, and *B* is the batchsize, the sampled proportion will fall within the range  
   
95% of the time. For B = 1000, the range is about ± .03 if p is around .5, ± .02 if p is around .1 or .9, ± .01 if p is near .98 or .02, and less than ±.01 if p less than .01 or greater than .99 ).

What happens next in each sample is that we compute net inputs to word units based on the sampled letters. If we had selected the letters A, G, and E at the letter level, then we would be likely to select the word AGE, but if we had sampled a different pattern of letters we would be less likely to select AGE, and indeed we will sometimes select the incorrect word, even if the correct letters were selected.

Processing in subsequent time steps proceeds in the same way, first updating the letter level and then the word level, but now when the letter level is updated, the active word is taken into account. For example, if AGE was selected at the word level, there will be ‘top-down’ input supporting the possibility that the first letter was A rather than H.

**Homework Simulations and Questions to Answer**

**Running the model**. Log into the lab computer making sure X-11 forwarding is set up, enter cd PDP, then enter

python3.5 MIA/MIA\_model.py

Python will start to run, and you will find yourself looking at a command prompt. At this point you can enter the following command to run the simulation we have been considering:

mia.run(‘hge’)

This sets the input to the network to be the correct features of the letters ‘h’, ‘g’, and ‘e’, and runs a batch of 1000 parallel copies of our network for 20 timesteps with the default values of OLgivenW and OFgivenL (both equal to 10) used to set the values of the connection weights. If all is well, a display appears summarizing the results of your simulation. You will be looking at the state of the network at time step 0, that is after one forward pass of activation from features to letters and then to words, but before any word-level input has affected the letter level. The display indicates the batchsize, the W to L scale factor, which is the natural log of OLgivenW (default = 10, the log of which is about 2.3), and the L to F scale factor, which is the natural log of OLgivenW (also 10).

You should see that the presented letters are more often selected than any other letters in each of the three letter positions. This is indicated in two ways: First by the length of the bars shown next to each letter in each position, and second, but the actual numerical value displayed in the second column to the right of the set of bars. For each letter position, and for the word level, there are two columns of numbers but for now we are only concerned with the ***rightmost*** of these columns, ie the numbers shown without [] around them.

You should see that H has been selected most frequently on time step 0 in position 0, while G has been selected most frequently in position 1, and E most frequently in position 2. This is due to the fact that their features all exactly match the features present in the input to the corresponding position.

**Question 1**. This is your first homework question. **(Q1.1)** Report the selection probabilities for the letters A, H, and P in position 0. These three numbers should come close to adding up to 1.

Now, let’s consider the ratios of these numbers. **(Q1.2)** Calculate and report p(A)/p(H) and p(P)/p(H). What do you notice? **(Q1.3)** Now determine, report, and explain these observations based on the value of the OFgivenL parameter (100 words max).

[HINT: As we have discussed, the computation that has been performed here is sampling from a softmax across the letter units in the first position based on the net inputs computed from the bottom up feature information. This computation is equivalent to calculating the posterior probability that it was the letter A, H, or P that was actually responsible for generating the observed features in this position, independent of any contextual influence since word-level activity is not yet in play. Given that OFgivenL is equal to 10, and given that H matches all of the presented features while A matches all but one of them, what do we expect the ratio p(A)/P(H) to be equal to? What do we expect for the ratio of p(P)/p(H)? You will probably see that the ratios calculated from sampling do not have exactly the correct values. The numbers we calculate from our knowledge of OFgivenL are true underlying expected values, and your sampled values should be similar to them.]

Now, consider the word level probabilities that were sampled during the first forward pass through your network (still timestep 0!). **(Q1.4)** Write down the probabilities you obtained for the words AGE, HAD, HER, HEX and HOW (again, the rightmost column of numbers). **(Q1.5)** Give an approximate explanation of the relative values of these probabilities in relation to the letter probabilities and the OLgivenW parameter (100 words max).

Next, let’s consider the letter and word probabilities after timestep 1 (the second time step!). Click/drag the slider slightly to the right of its initial position so that the value of ‘Timestep’ that you see is equal to 1. **(Q1.6)** What do you observe in this situation? Describe differences from timestep 0 in all three letter positions. Also describe differences that you see at the word level. Describe in words what seems to be happening (150 words max). [HINT: Recall that the letter probabilities are based on the word from the previous step, while the word probabilities on the current step are based on the letters from this same step. *Be sure to recall that the numbers you are seeing are proportions of sampled cases rather than continuous activation values.* For example, in a given sample, only one of the word units is active at the end of each time step.]

Finally, let’s consider the state at the end of 20 timesteps (timestep 19). Slide the slider all the way over to the right so that the timestep shows as 19. **(Q1.7)** Discuss how the values of the letter and word probabilities have changed further as the network has continued iterating. Write down the most important ones – A and H in position 0, G and E in positions 1 and 2, and the same words you were tracking previously. Describe in words any changes from timestep 1 (100 words max).

**How a neural network samples from the probability distribution of the generative model**

As discussed in McClelland (2013) the MIA model samples from the joint posterior of the generative model underlying the neural network, after it reaches equilibrium. This part of the homework emphasizes the joint posterior and the concept of equilibrium.

What does this statement mean? It means that after a ‘burn in period’ of a few time steps, the probability that the network will be in a given state should be equal to the posterior probability of that state under the generative model. As discussed in the paper, the fact that the underlying probability of being in a state does not change does not mean that there is no change going on in the network. In fact, different samples from the batch are moving in and out of different states between time steps (see the discussion of detailed balance in McClelland, 2013). Considering a ‘state’ to be a configuration in which one word is active and one letter is active in each position, there are 36\*263 such states, or 632,736 possible states. The two most likely states we have been considering are only two out of this large number of possibilities. Each state has a path probability of being generated under the model, equal to the probability of the word, the probability of the letters given the word, and the probability of the features given the letter. The posterior probability of each state, *given the features that have been observed,* is the path probability just described, divided by the sum of the path probabilities of all of the possible states of the network. Unfortunately, there are too many of these numbers to display. Instead we display the marginal posterior probabilities for each word and for each letter as well as the summed path probability (bottom middle of the display). The marginal posterior probability (MPP) for a word is the sum over all of the states in which that word was the active word (there are 263 such states!) of the path probability of each state divided by the summed path probability. For the word AGE, two of these states are contributing most of the total value of the posterior marginal probability: [AGE:A,G,E] and [AGE:H,G,E]. There are many other contributing states as well. The MPPs are the numbers displayed in [] just to the right of each letter or word histogram. You can see that AGE has the highest value, and several words including HAD have small but still noticeable values. Similarly, in each letter position, we are reporting MPPs for each letter. In all positions other than the position 0, the marginal posterior probability is very high for the letter actually shown, but in position 0, A has an appreciable probability, even though H best matches the input. It is difficult to perfectly understand the marginal posterior probabilities, since many states contribute to them, but we can work out the actual contribution of particular states. Most of the state probability is concentrated on a few states, for which the corresponding words and letters will show some marginal posterior probability. For example, A in the first position is primarily associated with the state [AGE:A,G,E], whereas H is associated with many states, including [AGE:H,G,E], [HAD:H,G,E] and [HOW:H,G,E] as examples.

**Question 2.** First we consider how you can determine the exact posterior probability of a particular state, as well as what is left over for other less likely paths. Let’s consider the state [AGE:A,G,E] under our input ‘hge’, where position 0 contains all the features of H. This path probability is (1/36)\*(10/35)3\*(10/11)41\*(1/11), or 1.830e-06. The summed path probability that we see in the display is 4.0487e-06. The ratio of these two numbers is the posterior probability of [AGE:A,G,E], and you should see that the value of this number is 0.2922. **(Q.2.1)** For the first part of your submitted answer, write down the expression for the path probability for the other probable path from AGE, [AGE:***H***,G,E] and report this path’s posterior probability. Then, considering the marginal posterior probability for all paths involving the word AGE, determine the summed posterior probability of all states involving AGE but excluding [AGE:A,G,E] and [AGE:H,G,E]. (Hint: you get this by a little addition and subtraction of the numbers you already have, and the result should be less than .04 but greater than .01. Your precision will only be around 3 decimal places because some of the numbers you have have been truncated, but this is good enough for our purposes.)

Now, let’s turn to observing how the sampled values of the MPPs at the letter and word levels approach the correct values. In this way, we can now ask about the time course of reaching equilibrium. You can assess this by comparing the numbers in the two columns next to each histogram.

**(Q.2.2)** Present the sampled numbers at timestep 0 for A and H in position 0, for G in position 1; for C, E, and F in position 2; and for AGE, HAD, HER, HEX and HOW and note how they differ from the MPPs. **(Q.2.3)** Explain why the numbers differ from the MPPs at this point, giving a short succinct summary (~100 words). **(Q.2.4)** Now examine the convergence of the network toward equilibrium. Does it reach equilibrium after one more step (timestep 1), or does it take a bit longer? You may find that it is a bit difficult to tell exactly when equilibrium is reached with a batchsize of 1000. Note that, even at equilibrium, the state of the network can change from timestep to timestep. A given sample can switch back and forth between states (this is what causes the sampled state probabilities to bounce around a little). **(Q.2.5)** Based on the letter and word level probabilities at equilibrium, what two states do you thing the network is most likely to alternate between? Explain your answer briefly.

If you like, you can run the simulation again with a batchsize of 10000. This allows a more exact sense of how closely the sampled values match the correct values under the generative model.

**Question 3: Exploring characteristics of the MIA model**

As the final part of this homework, we offer you the chance in Q.3.1 to explore the MIA model (and describe your exploration briefly) and then in Q.3.2 to write a brief evaluative comment. For your explorations in Q.3.1, the inputs can have some features masked from view. There are four pre-defined characters that you can use to create an incomplete display: ‘\_’ blanks all the features for a letter position; ‘#’ blanks the feature that distinguishes A from H; ‘@’ blanks the feature that distinguishes B from D; and ‘?’ blanks the features that distinguish E from O. You can also mask any feature in any position, as described on the MIA Program guide below.

**(Q.3.1)** Choose exploration A or B below or make up another of your own choosing. Limit yourself to 250 words in describing and discussing your exploration. Use some of the calculations you have learned above to consider the posterior probabilities of selected specific states, not just the marginal posterior probabilities of words or letters.

*A. Conjunctive sensitivity to combinations of constraints.* Note that many of the words come in quadruples like FOX FEW HEX and HOW. Given this situation, if you present F\_X, what happens in the network? Describe the initial and equilibrium states of the network, concentrating on the word level and the letter position of the blank, and consider whether the outcome seems reasonable for a model of perception of letters in words. Compare what you see to what happens when you present F\_\_ or \_\_X. How does the combination of F and X constrain the results in ways different from the separate effects of F and X? Try an additional case of your choosing to explore further (consult the list of words earlier in this document for another example case to try). Mention what you tried and how the results support or alter your conclusion about whether you think the model is doing something reasonable.

*B. Exploiting regularities implicit in the language*. What happens when we present a non-word that is similar to several words known to the model? Consider the string FEX, which is not a word. How do the sampled probabilities change from timestep 0 to equilibrium? Although the changes are subtle, they are systematic. Even though the stimulus is not a word, the expected marginal probabilities for F and E (and even X though the effect here is very slight) are greater than the initial independent probabilities that are calculated bottom up on timestep 0. Present a few other nonwords and describe what happens with them. What might such results tell us about the mechanism responsible for the fact that people see letters in non-words that are similar to many existing words more accurately than letters in random nonsense strings?

*Other possible explorations.* You can explore the effects of changing the odds of generating correct letters or correct features. The command mia.setl2f(V) sets OFgivenL to the value V and mia.setw2l(V) sets OLgivenW to V. You could explore whether this speeds up or slows down reaching equilibrium, if you increase or decrease either of these parameters from their default values of 0. Don’t feel restricted to this suggestion.

**(Q.3.2)** Evaluate the MIA model. State one thing that you find attractive about the model and one aspect of it that could be changed to make it a better model (in your opinion). How would you change the model to improve it. Your comment can focus on computational, psychological, or neuroscience issues as you prefer. Don’t try to do too much. Limit yourself to 250 words.

**MIA program usage guide**

To run the program from the PDP directory, type this at the linux prompt:

python3.5 MIA/MIA\_model.py

This produces a python process with a command prompt ‘>>>’

Before running a simulation you can set the parameters of the model with these commands:

mia.batchsize = 1000 (positive integer) [Numbers are default values]  
mia.timesteps = 20 (positive integer)  
mia.setl2f(10) (positive real > 1)  
mia.setw2l(10) (positive real > 1)  
mynet.setd(1) (0 turns top down input off – 1 turns it back on again)

To run the model use the following command. You may change parameters before each net run.  
  
 mia.run(‘str’,maskP=[N,…],…)



Details on use of run command:  
 ‘str’ is a string consisting of three letters or special chars. There are 4 special chars:  
 ‘\_’ all features hidden  
 ‘#’ A or H with top feature hidden so A and H are equally likely  
 ‘@’ B or D with the bar at the right missing so B and D are equally likely  
 ‘?’ E or O with distinguishing features hidden so E and O are equally likely  
 maskP=[N] lets you hide feature N in position P. (P = 0, 1, or 2; N in range 0 to 13)  
 For example, if ‘str’ is ‘age’, mask0=[2] hides the top feature of position 0 is hidden.  
 This is equivalent to the string ‘#ge’  
 There can be more than one maskP=[N,…] and a list of N’s separated by commas  
 See figure at right for feature numbering information. The features   
 around the edge clockwise from lower left are 0-5, then the horizontal  
 and vertical spokes from left clockwise are 6-9, then the diagonal spokes from  
 the lower left are 10-13.  
 Examples: 5 = bottom, 6 = short left middle horizontal, 11 upper left diagonal.

Each simulation launches a new copy of the viewer window which can be left open and saves a log file in the MIA/logs directory. The names of the log files are MIAlog\_N.pkl where N ranges from 0 upward with each new run. The logs contain all relevant parameters of the run but you may want to keep track of which one you ran with a given set of inputs or parameters.

Exit the simulation with ‘exit()’ or EOF (Control-D) . This closes viewer windows

To view a saved log, run this command from the PDP directory:

python3.5 MIA/MIA\_viewer.py

You will be prompted for a log. Enter the log name (MIAlog\_N.pkl) or the index N. After a log is plotted you will be prompted to plot another. If you answer ‘n’ the program closes its windows and exits.