Q1 [25 pts]: How is the recurrent network able to solve the task, when a feedforward network with the same number of hidden units between orthography and semantics fails? Be sure to discuss the role of the "cleanup" units in your response.

The recurrent network is able to solve the task when the feedforward network with the same number of hidden units between orthography and semantics fails due to its ability to repeatedly update unit activations synchronously (first nj , then aj), store entire activation history of each unit, attribute error to earlier activations, gradually update activations over time, and allowing for forward and backward connections and cleanup units to stabilize noisy inputs. From the collected data on error per example and % correct, the feedforward network (HS-ff_in.txt) is limited by high error rates (mean error around 6) and low accuracy (% correct around 27%), while the recurrent network (HS-rec_in.txt) achieves near-perfect accuracy (% correct around 97.5%) and a significantly lower error rate (mean error around 0.5). This difference is because the feedforward network processes the input in a single pass, so noisy orthographic inputs result in incorrect semantic activations with no mechanism for correction. However, as recurrent networks occur over several ticks (unit updates), cleanup units help the recurrent network by effectively adjusting activations by gradually reinforcing consistent semantic representations, allowing output features to stabilize. In the Unit Viewer, I examined the words DOG, BONE, HAM, and LAB by using the controls in the upper right to examine the temporal dynamics of the output and cleanup units during processing of the word and I did indeed notice that some semantic (output) features are activated sooner than others. The output units (semantic representations) gradually activate over multiple time steps, the cleanup units stabilized activations over time, and some of the words had more processing steps than others before getting to the correct activation pattern. So for example dog, there were strong early activations in the animal related semantic features and so on this comparison was able to be applied to the rest of the words as well and their respective categories. Ham and Lab were a bit slower and gradual with noisier activation. The cleanup units made it so the correct pattern stabilizes by steps three to four, especially for words with overlapping category properties. Due to this iterative process, the recurrent network is able to overcome noise and converge on accurate representations and cleanup units regulate competition between categories and refine and stabilize output and representation.

HS-ff_in.txt
Error per example, % correct
1. 5.62676, 27.5%
2. 5.84005, 22.5%
3. 6.22918, 32.5%
4. 6.61786, 27.5%
5. 5.84761, 25%

HS-rec_in.txt
Error per example, % correct
1. 0.51538, 97.5%
2. 0.59117, 97.5%

3. 0.54173, 97.5%
4. 0.42479, 97.5%
5. 0.45293, 100%

Q2 [10 pts]: Why does training error asymptote at such a high level?

The training error asymptotes at a high level because the Simple Recurrent Network applies targets to the output and generates correct activations at every time step during training, so errors from early time steps accumulate and cannot be fully corrected. Then during testing, only the final step is evaluated, allowing the network to converge correctly by the last time step, leading to 100% accuracy even with residual training error. From the collected data, the simple recurrent network consistently achieves 100% correct but has an error per example of 0.42629 so some earlier errors. Simple recurrent networks update feedforward networks for temporal tasks by incorporating context units that store hidden activations from previous time steps to guide future activations. Although, unlike fully recurrent networks, which update unit activities multiple times per input, SRNs are computationally efficient but functionally limited in their ability to refine activations over time, leading to residual training error. Also there is semantic overlap between words in similar categories like animals and food which causes competition by increasing training difficulty, requiring more iterations to fully separate representations. Intermediate activations may be unstable or incorrect with partial/noisy activations before stabilizing at the final step can't be corrected completely at intermediate step, showing why training error asymptotes at such a high level even with 100% correct.

HS-seq_in.txt
Error per example, % correct
1. 0.42629, 100%
2. 0.42629, 100%
3. 0.42629, 100%
4. 0.42629, 100%
5. 0.42629, 100%

Q3 [15 pts]: What determines when a given word is recognized? Why?

In the Unit Viewer I examined NUT, RUM, POP, and PORK, to get a sense of how the network is working. When all the semantic features reach the correct activation state determines when a given word is recognized, and it occurs at different time steps depending on the overlap in its semantic representation and its orthographic complexity. From my Unit Viewer examination of NUT, RUM, POP, and PORK, I observed that words with more different semantic features, different orthographic representations, and less semantic competition are recognized earlier (POP). Then words with overlapping and shared features cause stabilization to be delayed (PORK since it shares features with other food words). These are instances of strong and weak semantic activations respectively and determine how the simple recurring network stabilizes representations over time. The temporal processing of activation shows that words with more semantic competition require additional steps to fully separate from similar words as it requires

more processing time for complete recognition. Since the simple recurrent network processes inputs sequentially, small errors in earlier steps can spread forward, making training harder and shows why recognition time varies across words in the recurrent network.

Q4 [20 pts]: Suppose we had added an additional word to the vocabulary, LIM, which had a target of 0 for the first semantic feature (output:0). Assuming that the model was trained successfully to minimize error, what would the activation of this unit be for each step in processing LIM, and why?

If we had added an additional word to the vocabulary, LIM, which had a target of 0 for the first semantic feature (output : 0) and assuming that the model was trained successfully to minimize error, the activation of this unit will gradually decrease over time toward 0 in the last step of processing. At Step 1, the network could partially activate output: 0 because of orthographic similarity with other words in the training set with comparable features (LOG or LIP, same letter "L" → early activations with overlapping representations, possible similar semantic categories, etc.). Eventually with more letters/category context, the hidden and cleanup units solidify reduced activations, which suppress the incorrectly activated outputs. By Step 3/4, the network becomes stabilized, output: 0 approaches 0 due to training target. This is because simple recurrent networks rely on previous outputs and hidden states to adjust activations and allows the network to gradually correct errors from times it wasn't entirely correct before. Clean up units attempt to suppress and fix competing representations so the final output pattern accurately reflects proper semantics learned association with LIM. So simple recurrent network minimizes error with some intermediate fluctuations until stabilized to final representation.

Q5 [10 pts]: How do the hidden representations in the DBM differ (qualitatively) from those in the equivalent BP network, and why? How might this contribute to why the DBM needs more hidden units?

The hidden representations in the DBM differ qualitatively from those in the equivalent BP network because of the feedback and symmetric connections of hidden and output units. In BP, hidden representations are created by activation in a feedforward way and then adjusted by weights based on error backpropagation leading to an effective, steady input to output mapping. The DBM is trained with Contrastive Hebbian Learning, with a positive phase, where both inputs and outputs reinforce correct representations, and the negative phase, only the input is used and the network can reach a stable state. It is an iterative process occurring over time but hidden representations are more distributed and redundant than in BP. Regarding BP, it manually reduces error through gradient descent leading to more efficient hidden representations. From the Unit Viewer analysis, hidden unit activations in DBM are more spread out and distributed and less clear than in BP, showing that DBM requires more hidden units (50 in DBM, 10 in BP) to account for weaker, more overlapping feature activations. The error graphs shows that BP quickly converges to almost 0 error with minimal variation, DBM has spikes because its hidden units are activated more inconsistently, and the network tries to account for representations. The inconsistency and need for bidirectional compensation means it requires more hidden layer capacity to retain a larger variety of intermediate activations to train

effectively. The DBM needs more hidden units because its representations need to support gradually refining outputs over multiple updates and not from input during a single forward pass so it can represent accurate representations and correct errors.

Q6 [20 pts]: What comprehension errors (or near errors) occur in the network? Considering them together, what factors seem to influence the kinds of errors that occur, and why? How might the properties of the DBM representations (as considered in the previous question) lead to the "spikes" in training error caused by these comprehension errors?

The comprehension errors in the DBM happen when the network activates the correct ident unit and some incorrect ident units, due to semantic overlap and word similarities. From the Unit Viewer final activations, when looking at the ident units, some of the significant comprehension errors that occurred in the network were at HAM (hock), LIP (hip), COW (ram), CAT (dog), and BUG (pig). The () shows the other partially activated unit. This is because they may have similar orthographic or semantic features, such as if they are in the same category and synchronized well together (ham and hack both start with “H”, lip and hip rhyme both body parts, cow and ram are similar animals, dogs/cats are in the same kind animal group, bug/pig both end in “g” and both 3 letter animals). The factors that seem to influence the kinds of error that occur have to do with misrecognition because the DBM's distributed representations store overlapping feature activations not independently localized representations. In BP, forward mapping has specific entries within certain units, so the DBM's forward and backward learning process can have hidden units to encode multiple related concepts like for example maybe “bear” and “bare”, resulting in partial activations of incorrect ident units . As seen in the unit viewer, words share semantic features (animals, body parts, meat, rhyming, alliteration, etc.), and their representations can increase interference. The error spikes in the training graph with 310 epochs are when the network is adjusting weights to fix classification of activations to minimize the other generalized activations. During contrastive Hebbian learning, the positive phase reinforces correct connections and the negative phase lets the network go into stable states (average the activations, if a unit is active for 2 different contexts can increase stability and increase convergences), but if multiple ident units are partially active, it increases instability and slows convergence increasing errors before stabilizing especially with similar semantic features. So the spikes in training error because the network is unable to make similar words distinct over multiple updates as it is attempting to get accuracy and generalizability simultaneously in the representations.