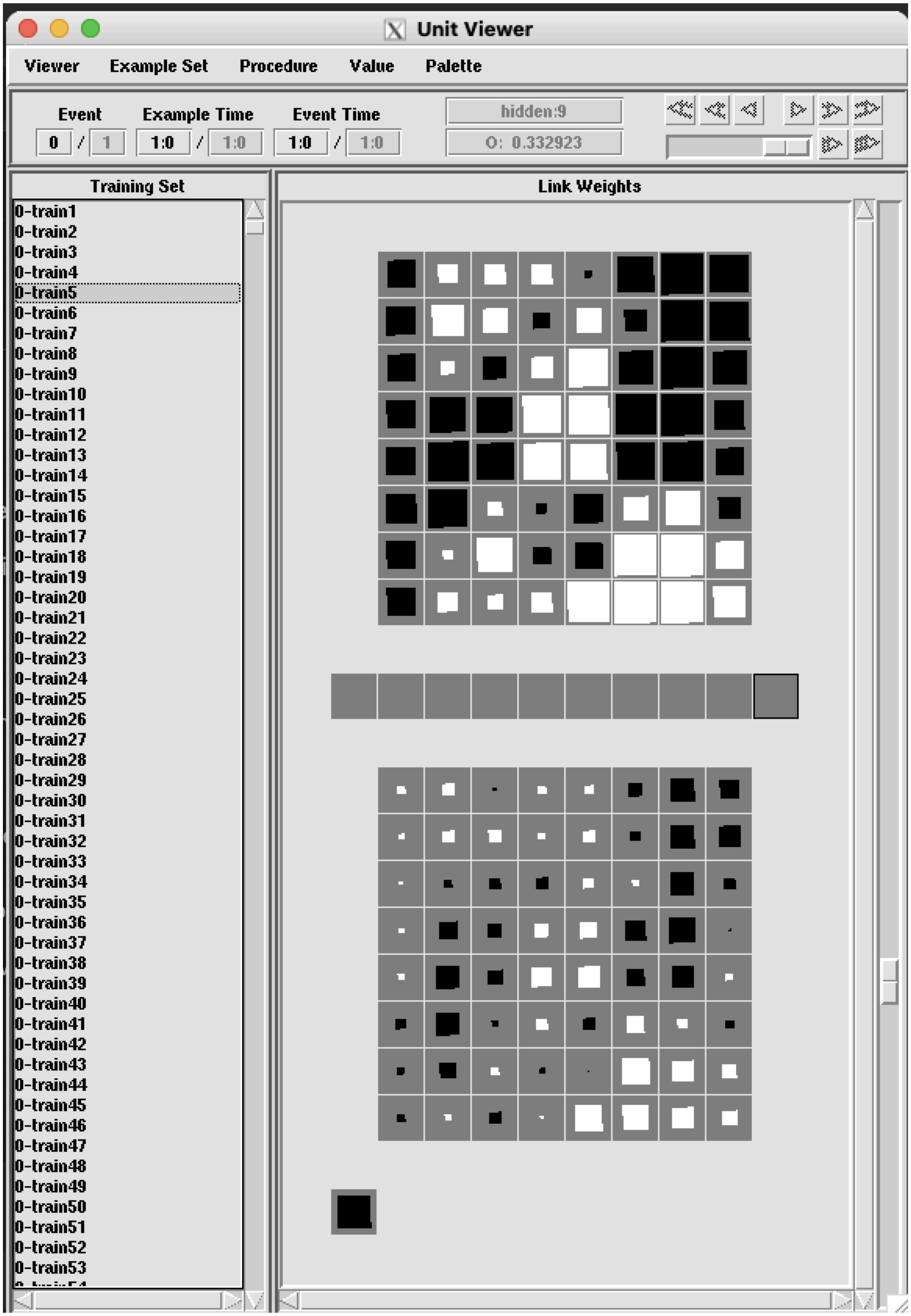
Q1 [15 pts.] Consider how the network processes two examples in particular: "0-train5" and "0-train24". For each of these examples, describe the way in which the reconstruction is inaccurate (i.e., the differences between the generated outputs and the targets=inputs), and why this make sense given the general properties of learning in auto-encoder networks.

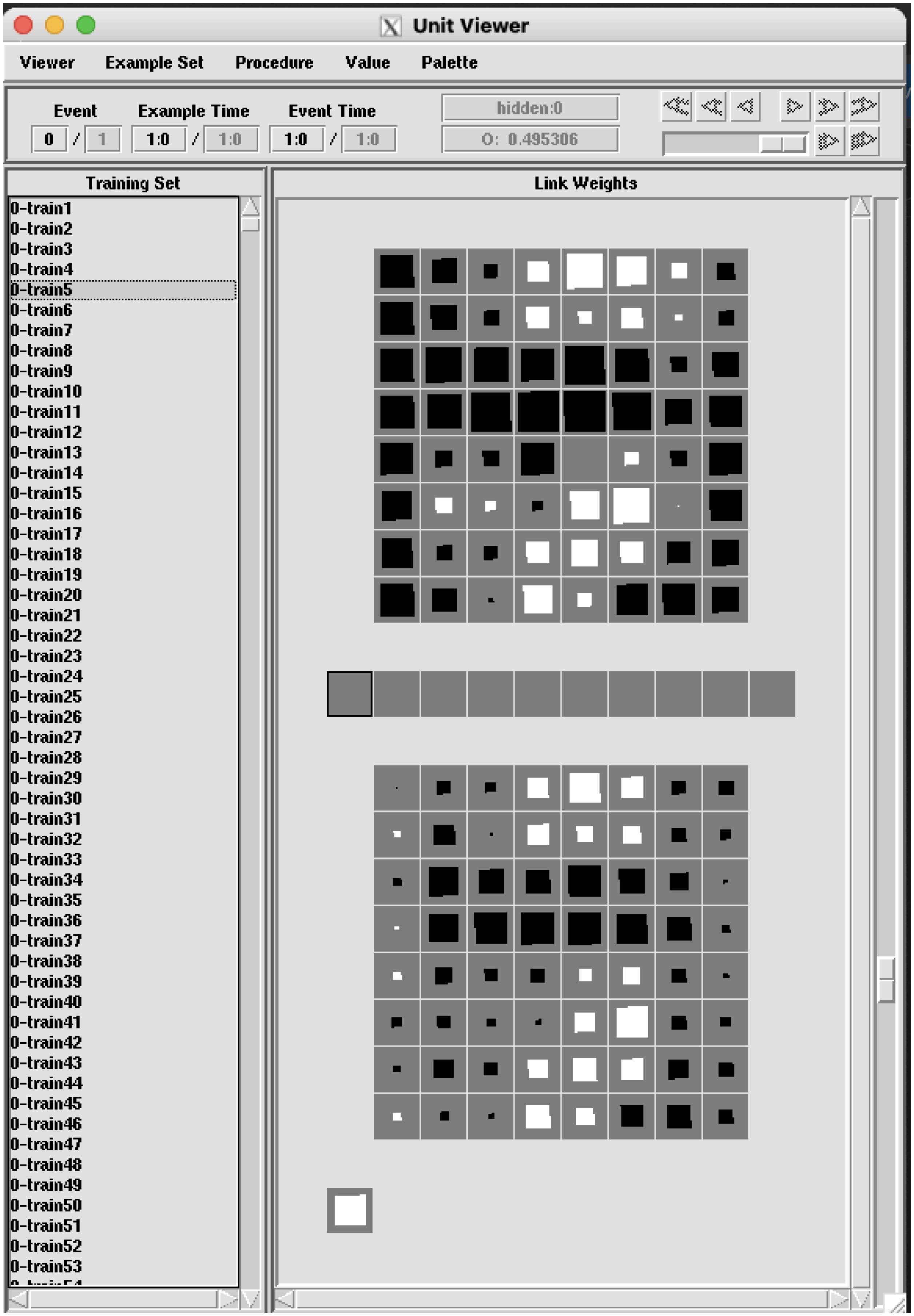
From comparing the Unit Viewer reconstructions of “0-train5" and "0-train24", I can see that each is accurate to an extent because auto-encoders prioritize general structural features over minor detail specifically when dimensionality challenges arise. "0-train5" forms a reconstruction that mostly follows a similar oval shape of zero but deviates with inaccuracies in the upper left and lower right corners where some expected activated target pixels are weakly activated with white pixels and some pixels that were meant to be black are instead activated as gray which I didn’t expect. After auto-encoder observation, the compressed structure generates a hidden representation that deactivates the more idiosyncratic appeal while activating what common and average features as well. Then, "0-train24", creates a clearer reconstruction with a more continuous and symmetric outer rim and hole in the center clearly separated, meaning that this zero is a more consistent representation of zeros from the training set and would be displayed more accurately by the internal code. The difference is caused by training presentation errors that change timestamps, for instance, the smallest detail is only activated by epoch 300. This means that for strong activation to occur initially and be maintained, it would have only supported larger concepts than smaller distinctions. The limitations of representing high dimensional input using a compressed hidden layer are demonstrated by the fact that both images have some blurriness and a softer grayscale contrast. This is similar to the Elman net challenges as well as Munakata's findings regarding graded internal representations in cognitive development.

Q2 [15 pts.] Characterize, in general terms, the nature of the features that the hidden units have learned and how they combine to reconstruct the digits. [Do not exhaustively describe each unit; rather, illustrate your general points with details from a few specific units and digits.]

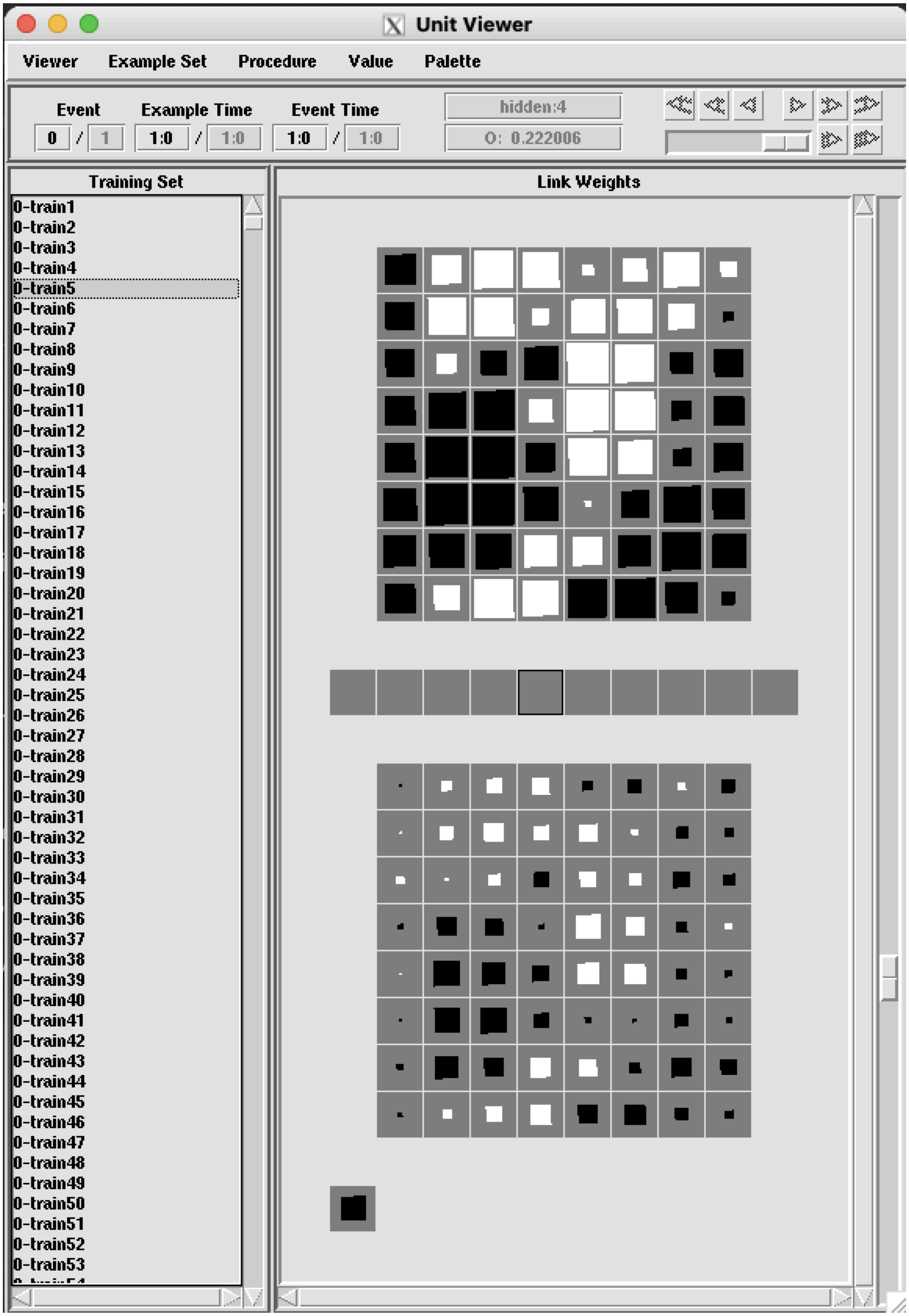
To characterize in general terms the nature of the features that the hidden units have learned and how they combine to reconstruct the digits, I examined the hidden units in 0-train5, In dimensionality reduction, each hidden unit in dimensionality reduction learns to recognize a specific portion or pattern of the digit, such as a curve or a line rather than trying to represent the entire digit solely. This happens because of the network's constraint layer structure, limiting the amount of hidden units it can use and requires effective input compression. It learns to neglect irrelevant or redundant information while recognizing the more significant features that differ between digits rather than learning the whole digit. For example, hidden unit 0 (screenshot 2) learns strong incoming weight for the top and bottom edges in relation to a top-bottom border or symmetry via the horizontal axis which was common for most 0/1s. Hidden unit 4 (screenshot 3) is representative of a weighted vertical line section as it receives most of its weights from the middle column line because one of the characteristics of 0s is symmetry via the vertical axis. Hidden unit 9 (screenshot 1) learned curved edge segments or semi-circles since its strongest incoming weights are along the bottom right edge and center left. Since each digit represents an array of hidden units targeting fractional features, no single hidden unit can code for complete numbers. To regenerate the initial reconstruction, the decoding learns how to combine each outgoing weight. It learns that by exposing its fractional learned subcomponents to other types that are comparable without a specific, consistent identity supporting noise resistance and generalization. So the hidden layer uses only their connections to generate complex input representations, functioning as a collection of consistent fragments from which it can construct them repeatedly.



0-train1
0-train2
0-train3
0-train4
0-train5
0-train6
0-train7
0-train8
0-train9
0-train10
0-train11
0-train12
0-train13
0-train14
0-train15
0-train16
0-train17
0-train18
0-train19
0-train20
0-train21
0-train22
0-train23
0-train24
0-train25
0-train26
0-train27
0-train28
0-train29
0-train30
0-train31
0-train32
0-train33
0-train34
0-train35
0-train36
0-train37
0-train38
0-train39
0-train40
0-train41
0-train42
0-train43
0-train44
0-train45
0-train46
0-train47
0-train48
0-train49
0-train50
0-train51
0-train52
0-train53



0-train1
0-train2
0-train3
0-train4
0-train5
0-train6
0-train7
0-train8
0-train9
0-train10
0-train11
0-train12
0-train13
0-train14
0-train15
0-train16
0-train17
0-train18
0-train19
0-train20
0-train21
0-train22
0-train23
0-train24
0-train25
0-train26
0-train27
0-train28
0-train29
0-train30
0-train31
0-train32
0-train33
0-train34
0-train35
0-train36
0-train37
0-train38
0-train39
0-train40
0-train41
0-train42
0-train43
0-train44
0-train45
0-train46
0-train47
0-train48
0-train49
0-train50
0-train51
0-train52
0-train53



0-train1
0-train2
0-train3
0-train4
0-train5
0-train6
0-train7
0-train8
0-train9
0-train10
0-train11
0-train12
0-train13
0-train14
0-train15
0-train16
0-train17
0-train18
0-train19
0-train20
0-train21
0-train22
0-train23
0-train24
0-train25
0-train26
0-train27
0-train28
0-train29
0-train30
0-train31
0-train32
0-train33
0-train34
0-train35
0-train36
0-train37
0-train38
0-train39
0-train40
0-train41
0-train42
0-train43
0-train44
0-train45
0-train46
0-train47
0-train48
0-train49
0-train50
0-train51
0-train52
0-train53

Q3 [10 pts.] Describe the patterns of weights you observe and contrast them with the types of features learned by the hidden units in the auto-encoder network. Why do the two learning frameworks develop such different patterns of weights?

In the Self Organizing Map, all of the map units become like a prototype digit after a while so map unit 3 responds mostly for the digits 0, 2, 9, and its weights are adjusted to show what an entire version of 0, 2, 9 would look like, the hidden units of an autoencoder learn something more dispersed and fragmented. This is because an idealized weight pattern would comprise entirely shapes so weights that could look like 0 or 8 or 6 instead of parts. SOMs operate through competitive learning, so when the digit is the most for a map unit, that map unit and its corresponding learned neighborhoods learn to adjust weights toward that digit. Over time, as weights adjust and the area of effect which is the neighborhood becomes smaller, each map unit is developed for a specific purpose, and topographical ordering occurs so all map units with similar digits are located next to one another. Through activations, patterns will then overlap or be actually close enough to activate adjacent or identical map units. Reconstruction error determines how all of the system's units will adjust, while an autoencoder allows connection weights to be changed through backpropagation. Hidden units learn to combine fragments instead of representational fragments since this network has a limited constraint layer that needs to learn effective codes for compression before any decoding can occur. So as the autoencoder deals with dimensionality reduction and reconstruction accuracy by compositional characteristics, SOM deals with classification and topological similarity.

Q4 [15 pts.] Characterize, in general terms, how the learned features in the recognition network (i.e., the patterns of incoming weights to hidden units) differ from those in the auto-encoder, and why these differences make sense given the nature of the two tasks (generation vs. recognition).

Compared to the auto-encoder's more general, overlapping features, hidden units in the recognition network are learning more specific features in general and to the categories/class. This is because auto-encoders are attempting to reconstruct inputs by compressing and expanding patterns, while recognition networks try to classify inputs by extracting features which is maximizing class separability. In a recognition network, every hidden unit prioritizes highlighting spatial areas and stroke patterns which are highly representative of particular digits. For example, within different digit examples from 0-9-train examples, I right clicked on random hidden units 0-9, Hidden unit 4 shows strong response to bottom loops of curved digits such as 0 and 6, while hidden unit 6 picks out vertical strokes in the right half common to 2 and 3. Additionally, the weight patterns are more localized and fewer than the auto-encoder's distributed features, such as in hidden unit 9 focused significantly on the bottom middle region probably to distinguish between bottom heavy digits such as 4, 5, or 9. This is because discriminative representation is required for the recognition task to support class boundaries in the feature space. The auto-encoder learns smoother, overlapping encoding features more effectively suited for general pattern regeneration instead of categorization by attempting to reduce reconstruction error. While the auto-encoder learns features that fit the overall reconstruction, the recognition network learns as many characteristics that distinguish between them as possible.

Q5 [20 pts.] Report the training and testing performance of the network at both 150 and 1150 epochs, and explain the pattern of results.

At 150 epochs, the recognition network successfully trained on training and test sets training error per example 0.21804, and 97.75% accuracy on examples, and test error per example 0.40054 with 94.32% accuracy. With 1150 epochs of training, the network had attained 100% training set accuracy and 0.00018 error but poor generalization to test set: error per example was 1.52883 and criterion performance was 91.82%. The pattern is overfitting, with more training, network memorized training data reducing training error to near zero, but with performance decreased on new inputs. Classification accuracy is measured by the output unit criteria, which is the correct output unit with the maximum activity. The decrease on the test set shows that the features learned are too specialized for training examples. Since too much training results in weak representation, where internal representation fails to detect underlying patterns in a variety of examples, recognition networks generalize best when internal representation reaches a balance between generality and specificity.

```
150 epoch
Train:
Test results on 3823 examples, 3823 ticks:
Network
Error total:        833.576
Error per example:   0.21804
Error per tick:      0.21804
Unit cost per tick:  0.00000
Output unit criterion reached on 3737 examples (97.75%)

Test:
Test results on 1797 examples, 1797 ticks:
Network
Error total:        719.768
Error per example:   0.40054
Error per tick:      0.40054
Unit cost per tick:  0.00000
Output unit criterion reached on 1695 examples (94.32%)


1000 epoch
Train:
Test results on 3823 examples, 3823 ticks:
Network
Error total:        0.67828
Error per example:   0.00018
Error per tick:      0.00018
Unit cost per tick:  0.00000
Output unit criterion reached on 3823 examples (100.00%)

Test:
Test results on 1797 examples, 1797 ticks:
Network
Error total:         2747.30
Error per example:   1.52883
Error per tick:      1.52883
Unit cost per tick:  0.00000
Output unit criterion reached on 1650 examples (91.82%)
```

Q6 [10 pts.] Report the results for the deep network. Why is performance of the deep network so much worse than that of the standard network?

By 150 epochs of training, the performance of the deep network is so much worse than that of the standard network. On training data, deep network had an error per example of 1.22618 with 79.18% accuracy, while shallow network had significantly better error of 0.21804 and 97.75% accuracy. On test data, the deep network had an error of 1.35291 with 73.85% accuracy, while the shallow network had an error of 0.40054 and 94.32% accuracy. This performance disparity is because of the difficulty and challenge with training deep feedforward network with sigmoid units due to the vanishing gradient problem; earlier layers of deeper networks see much weaker weight updates, slowing down or even possibly stopping learning entirely. This makes training features inefficient except under special conditions, so it performs worse because it saturates and sees weaker gradients. Since this deeper network still uses sigmoidal units, it cannot successfully train hierarchical representation within 150 epochs and thus performed significantly worse than the standard network trained under identical conditions.

Q6
150 epoch

```
Train:
Test results on 3823 examples, 3823 ticks:
Network
Error total:         4687.70
Error per example:   1.22618
Error per tick:      1.22618
Unit cost per tick:  0.00000
Output unit criterion reached on 3027 examples (79.18%)

Test:
Test results on 1797 examples, 1797 ticks:
Network
Error total:         2431.18
Error per example:   1.35291
Error per tick:      1.35291
Unit cost per tick:  0.00000
Output unit criterion reached on 1327 examples (73.85%)
```

Q5:
150 epoch
Train:

```
Test results on 3823 examples, 3823 ticks:
Network
Error total:         833.576
Error per example:   0.21804
Error per tick:      0.21804
Unit cost per tick:  0.00000
Output unit criterion reached on 3737 examples (97.75%)
```

Test:

```
Test results on 1797 examples, 1797 ticks:
Network
Error total:         719.768
Error per example:   0.40054
Error per tick:      0.40054
Unit cost per tick:  0.00000
Output unit criterion reached on 1695 examples (94.32%)
```

Q7 [15 pts.] Report the results for the deep RELU network. Why is performance of the deep RELU network better than that of the standard sigmoid network?

After training for 150 epochs, the performance of the deep RELU network was significantly better than that of the standard sigmoid network. On training data, RELU network converged to 0.07427 error on each example and reached its output unit criterion in 99.37% of examples, while sigmoid network converged to 0.21804 error and 97.75% accuracy rate. On test data, RELU model converged to 0.36195 error on each example and 95.44% accuracy rate, while sigmoid network converged to 0.40054 error and 94.32% accuracy rate. This is because in the RELU network, while sigmoid units saturate and are negatively impacted by vanishing gradients specifically in deeper layers, RELU units preserve larger gradients in training so more efficient error backpropagation and faster convergence are achieved. RELUs also introduce sparsity in representation by discarding large activations, supporting more conservative representation of discriminative features throughout the network. Although RELUs are not limited between 0 and 1 as are sigmoids, addition of weak penalty on large activations prevents too large values without compromising their gradient benefit. Overall, even after the same amount of training time, deep RELU networks exhibit higher generalization because of their deeper structure and enhanced learning dynamics.

Q7:

Train:

```
Test results on 3823 examples, 3823 ticks:
                Network
Error total:        283.947
Error per example:   0.07427
Error per tick:      0.07427
Unit cost per tick:  49.9153
Output unit criterion reached on 3799 examples (99.37%)
```

Test:

```
Test results on 1797 examples, 1797 ticks:
                Network
Error total:        650.431
Error per example:   0.36195
Error per tick:      0.36195
Unit cost per tick:  49.5805
Output unit criterion reached on 1715 examples (95.44%)
```

Q5:

150 epoch

Train:

```
Test results on 3823 examples, 3823 ticks:
Network
Error total:        833.576
Error per example:   0.21804
Error per tick:      0.21804
Unit cost per tick:  0.00000
Output unit criterion reached on 3737 examples (97.75%)
```

Test:

```
Test results on 1797 examples, 1797 ticks:
Network
Error total:        719.768
Error per example:   0.40054
Error per tick:      0.40054
Unit cost per tick:  0.00000
Output unit criterion reached on 1695 examples (94.32%)
```