Neural networks, final project report

The task I chose for the final project was to implement the project's Part II, i.e. the ETAM system as described here, with its training and retraining algorithm. More precisely, the original formulation of the problem was, as follows:

Based on the training patterns, create a network such that every training pattern is in a stable state of this network (meaning the application of the network to the pattern results in the same pattern). This network will then be applied for recognizing corrupted patterns. While recognizing pattern q, we apply the network to it, and then, until we reach a stable state or a cycle, we apply the network to the result and so on. If we reach a stable state, we check if any training pattern p was identical to that state. If so, we recognize q as the same class as p. If we reach a stable state with no training pattern, or fall into a cycle, we do not recognize q at all.

The dataset used for the task was, again, the MNIST handwritten digit dataset. The network was a single layer of 784 neurons, because the input images were composed of $28 \cdot 28 = 784$ pixels. The patterns were preprocessed so that all grey pixels were set to value 1 (active pixels), and all white pixels - to value -1 (inactive pixels).

After training, the i-th neuron was supposed to be activated for all patterns whose i-th pixel was active, and only such patterns. The procedure, as described in the ETAM algorithm, was to initialize the coefficients vector of the i-th neuron to the sum of all such patterns, minus the sum of all the other patterns, and then normalize the vector to length 1. The bias was initialized to 0. Afterwards, a series of changes (retraining phase) was made to the coefficients vector and bias, representing translations and rotations of the separating hyperplane of this neuron. This was supposed to increase the margin of error for this neuron, i.e. make it more likely that the result of applying the network to a corrupted pattern will be the same as the result of applying it to the original, non-corrupted pattern.

To understand the impact of the retraining phase, I first ran the ETAM algorithm limited to the initialization phase. This did not yield good results. Most test patterns (9188 out of 10000) were not recognized at all, meaning the repeated application of the network to them either resulted in a cycle, or transformed them into a stable state that was not identical to any pattern from the training set. Because of this, I conducted some experiments to see if I can improve this result.

The main problem of the algorithm was that the initialization phase did not, in fact, make most of the training patterns stable. Therefore, while recognizing a corrupted pattern, the fact that it was first stabilized by repeatedly applying the network, and then we checked whether the obtained stable state was identical a training pattern, meant that it was unlikely to be recognized, because that stable state would be very unlikely to actually be identical to a training pattern. Because of this, I tried to improve performance by stabilizing the training patterns themselves before assigning them to the stable state they ended up in after stabilization. Fortunately, it turned out that after just 3 applications of the network, all of the training patterns were stabilized. With this improvement, 2099 test patterns were recognized correctly, 7901 incorrectly, and none were unrecognized. This is interesting for two reasons. First, it provides us with a good baseline to test the effects of the retraining phase - not particularly good, but considerably better than random guessing (21% accuracy versus 10% for random guessing, given that there are 10 labels). Second, it is curious that patterns were often recognized incorrectly, but never not recognized at all. That means the network was still quite likely to transform images representing different digits into the same stable state.

After including retraining with learning rate set to $l = \frac{1}{128}$, with the number of applications of the network to the training patterns $a = 3$, the number of test patterns which did not end up in stable states after a iterations increased to 9088, while among the patterns that were stabilized, 407 were correct and 505 incorrect. This would suggest that increasing a to decrease the number of unstabilized patterns, the ratio of correctly and incorrectly recognized patterns would be preserved, and the obtained accuracy would

exceed 40%. While the number of unstabilized patterns did decrease to 0 with increasing a, after setting $a = 12$, the results were, as follows: correct 2719, incorrect 7279, unrecognized 2. Therefore the accuracy of 27% was obtained, 7 percent points better than without retraining.

I think that what decreased the stability of training patterns after retraining was an assumption of the algorithm that was not fulfilled. The step in which the hyperplane is translated to halfway between patterns p_n and p_p is supposed to serve to maximize the margin of error between the patterns above and below the hyperplane. Patterns whose i-th pixel is active (let us call them active patterns) were supposed to lie above the hyperplane, and the others (inactive patterns) below it. Intuitively, such a translation looks sensible if p_p lies above the hyperplane, and p_n - below it. However, p_n is defined as the active pattern with the smallest signed distance to the hyperplane. It is not guaranteed that this distance is positive. A symmetrical problem, of course, exists for p_n. In fact, after initialization, p_p does tend to lie below the hyperplane, while p_n - above it. Thus, while the translations of the i-th neuron's hyperplane would preserve the stability of the patterns (with respect to the i-th pixel) if they were stable to begin with, they were not stable and there was no guarantee that the translations would not further decrease their stability.

It can be noted that even if that was not a problem, the step in which the hyperplane is rotated is not guaranteed to preserve stability even in the case of all patterns being initially stable.

I tried chaning the heuristic and choosing p_p as the pattern with the smallest positive distance, and p_n as the pattern with the highest negative distance. However, this yielded results lower than the baseline: correct 973, incorrect 9027, unrecognized 0.

Finally I experimented a bit with the learning rate. With $l = \frac{1}{512}$, the results were: correct 3125, incorrect 6874, unrecognized 1. With $l = \frac{1}{1024}$, they were: correct 3130, incorrect 6869, unrecognized 1. Given the small difference between those results, I did not experiment with any lower values.

Thus, the final accuracy obtained on the test set was 31.3%.