Usage Notes:

Type

python XOR.py -h

in the terminal for additional usage instructions.

Typing

python XOR.py

defaults to expected behaviour (2-layer, lr=0.01, epochs=2000).

Screenshots:

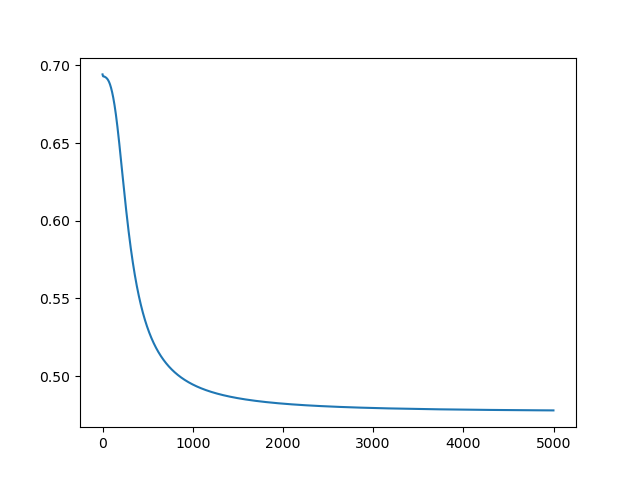
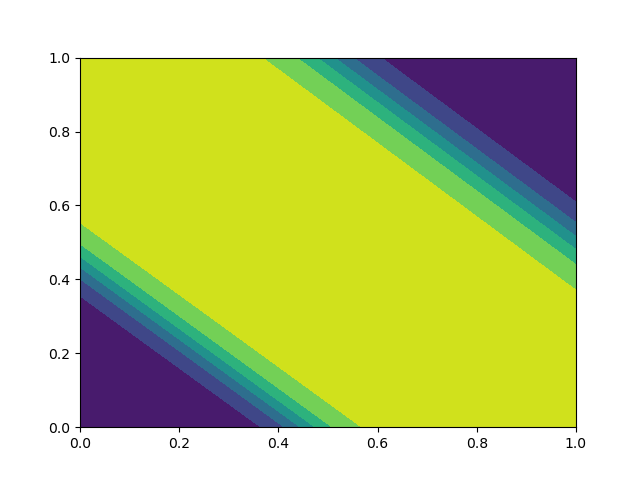


Fig. 1(a) and Fig. 1(b): 2-layer implementation with default configuration

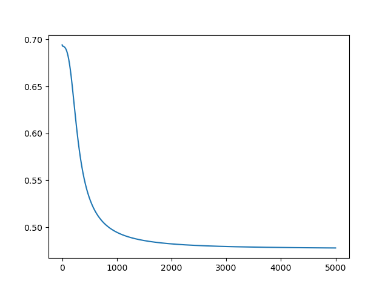
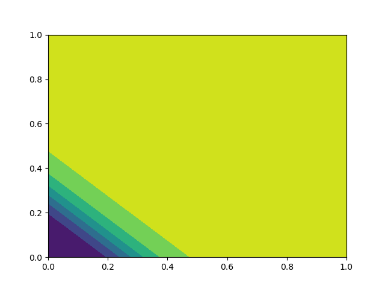
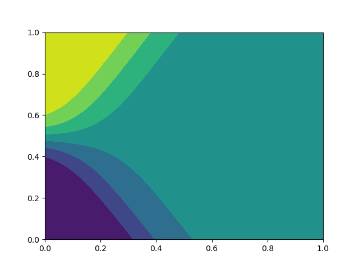


Fig. 2(a), Fig. 2(b) and Fig. 2(c): erroneous results from the 2-layer implementation with default configuration

The usage of Batch Gradient Descent (BGD), in particular, seems to produce undesirable results at times as shown in Figures 2(a), (b) and (c). Decreasing the learning rate, increasing the number of iterations and applying Stochastic Gradient Descent (SGD) leads to the network finding the correct result more often, as seen in Fig. 3(a) and 3(b).

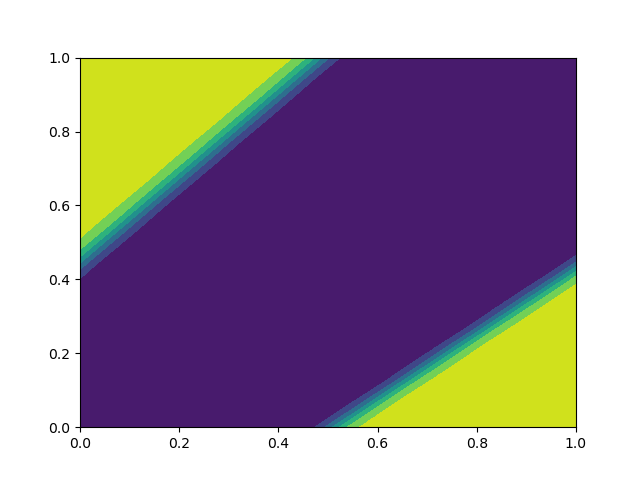
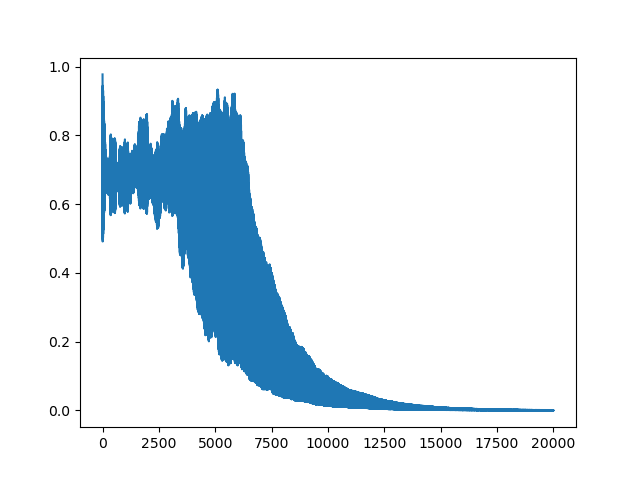


Fig. 3(a) and 3(b): 20000 iterations of SGD, lr=0.005 (20000 / 4 = 5000 epochs).

Note that different “answers” with different directions/signs of the central band exist.

The other two networks were largely ineffective at finding the correct solution, as shown below.

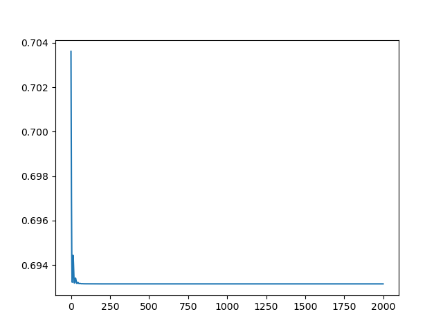
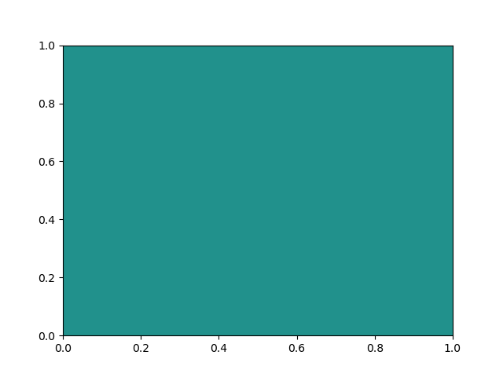


Fig. 4(a) and Fig. 4(b): results of the 4-layer network

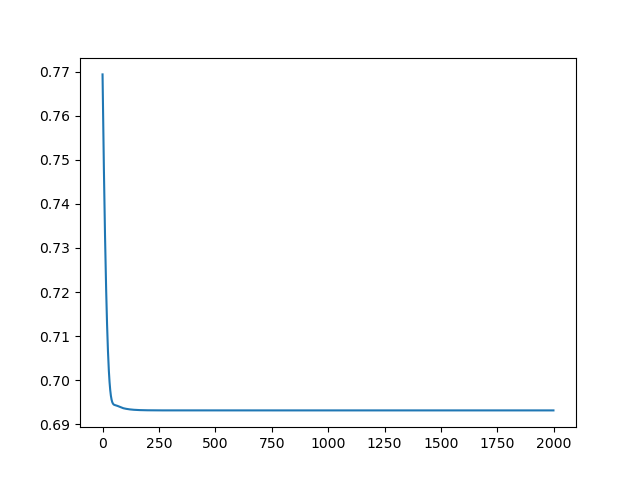
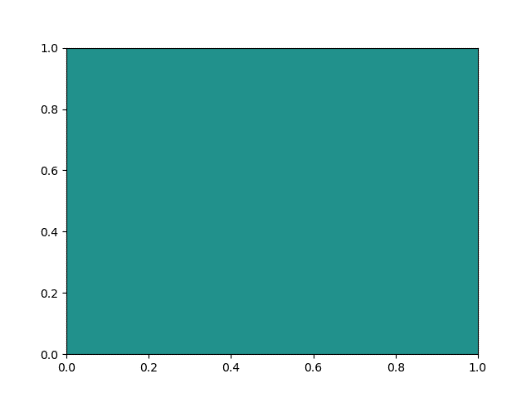


Fig. 5(a) and 5(b): results of the 1-layer network

Note that the 4-layer network is effectively identical to a 1-layer network since the first few layers introduce no nonlinearities and a sigmoid function is applied to an essentially linear transformation of the input at the end.

Both the 1-layer and 4-layer networks achieve an accuracy of 0.5, which means the network performs about as well as randomly guessing the result.