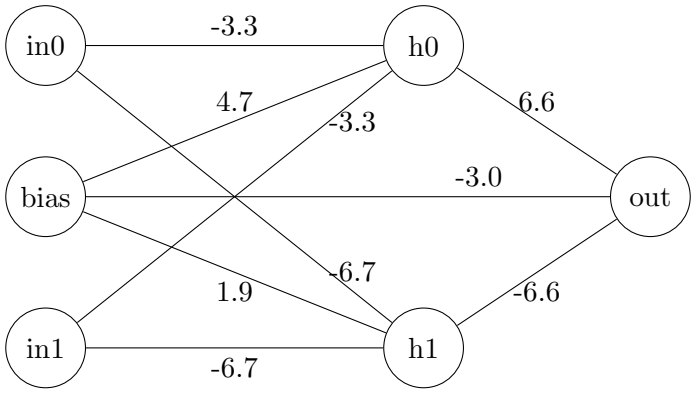
CS63 Spring 2017
Lab 7: Neural Networks

Cindy Li, Grace Zhang

April 10, 2018

1 XOR

With random seed 24 our network achieved 100% accuracy on the XOR data set after 10000 training epochs. The resulting neural network is shown in the following diagram.



Looking at our truth tables:

in0	in1	h0	h1
1	1	0	0
0	1	1	0
1	0	1	0
0	0	1	1,

we can say that h0 represents the NAND gate while h1 represents the NOR gate.

Together, they are equivalent to the boolean function h0 AND (NOT h1):

h0	h1	out
1	1	0
0	1	0
1	0	1
0	0	0.

2 Keras

Here is how we achieved 97.88% accuracy on the MNIST handwritten digit data set.

2.1 Parameters

- hidden layer activation: `relu`
- output layer activation: `softmax`
- optimizer=`Adam`
- loss=`mean_squared_error`

2.2 Process

First, we tried changing the number of epochs. Changing the value from 10 to 12 led to a 10% increase in accuracy, but a change from 12 to 20 epochs showed negligible results, so we decided to keep this number at 12.
Next, we played around with the number of hidden layers. Adding a hidden layer drastically decreased the performance so we stuck to two hidden layers. Afterwards, we played around with the other available activations. We found that changing one of the sigmoid activations to relu resulted in an accuracy of 87.73%. We changed the other sigmoid to a softmax activation according to Bryce's suggestion in the lab write-up, resulting in an 89.47% accuracy.
Finally, we changed our optimizer to Adam and kept our loss as mean_squared_error after playing around with a few. Some of the other optimizers we tried included the default SGD, RMSprop, and Adagrad. Some of the other loss functions we tried included squared_hinge and poisson.
`https://keras.io/losses/` `https://keras.io/optimizers/`

2.3 Explanation

When using the ReLU activation function and the input is positive, the derivative is just 1, so there isn't the sqeezing effect you get on backpropagated errors like from the sigmoid function.

Softmax acts similarly to the sigmoid function, but normalizes the outputs so they are equivalent to a probability distribution, which tells you the probability that any of the classes are true.
`https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Func`

Adam maintains a per-parameter learning rate instead of a single learning rate (alpha) for all weight updates. This improves performance on data sets with sparse gradients. These per-parameter learning rates are adapted based on the average of recent gradients, so this algorithm can do well on online and noisy problems.
`https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`