Forward-Feeding Neural Network (FFNN): Takeaways from Learning Poker Hands

STUFF TO KNOW BY HEART - EVEN WHEN DRUNK!

1. Skewed classes poses a serious problem for machines to learn to classify them well; methods to mitigate this problem all tend to be computationally expensive

2. With a big data set, it is often computationally infeasible to let the machine look at the whole data set all at once, and it is better to give the machine "mini-batches" of a manageable number of cases each time

There are 2,598,960 five-card poker hands (and, if the order of the cards is taken into account, there are 311,875,200 poker hands!!!), each of one of the following 10 types:

1. nothing to shout about (50.1%)
2. 1 Pair (42.3%)
3. 2 Pairs (4.75%)
4. 3 of a Kind (2.11%)
5. Straight (0.392%)
6. Flush (0.198%)
7. Full House (0.144%)
8. 4 of a Kind (0.024%)
9. Straight Flush (0.00139%)
10. Royal Flush (0.000154%)

The task of learning to classify these 10 types of poker hand involves dealing with some tough problems, as discussed below.

1. SKEWED CLASSES

The 10 types of poker hand differ from the 10 numeric digits in one crucial aspects. While the 10 digits occur with equal statistical probabilities (they form 10 **balanced** classes) the 10 types of poker hand occur with exponentially different probabilities (they form 10 very **skewed** classes). In a typical poker data set, the mundane classes such as Nothing or 1 Pair make up over 90% of the cases, while there are very few cases of the rare classes of poker hand such as 4 of a Kind and Straight Flush.

This poses a problem because the the **classic Cross Entropy cost function for classification machines treats all cases equally**, making the machine care about **overall accuracy** over an entire data set. If a machine can learn to classify just Nothings and 1 Pairs well, it is already over 90% accurate overall, and it can start resting on its laurels and not having the "motivation" to classify the rarer classes. (In technical terms, at 90%+ accuracy, the gradients of the weights become quite flat because of the limited room for improvement, so additional learning will be very slow.)

Our goal is actually **NOT maximizing overall accuracy**, but rather **maximizing the machine's ability to distinguish** among 10 classes. If we need to classify the rare classes well, we must either let the machine see the rare cases more often, or require it to place more importance on such cases. There are 2 corresponding mitigating solutions:

1. Instead of training the machine using the typical skewed data set, train it machine with an **adjusted data set with the rare cases repeated** so as to make the classes (more) balanced in that big data set. The more skewed the original classes are, the multiplicatively bigger the adjusted data set is than the original data set, and consequently the more training time is required.
2. Modify the cost function to **weigh the cost of different classes differently, depending on their prevalance** (**rarer** classes must be assigned relatively **higher importance**). However this leads to mathematical formulas that are not as neat as in the balanced-classes case, and this leads to slower training too.

2. MINI-BATCH LEARNING FOR BIG DATA SETS

Training a machine to classify poker hands well require a data set of about 1,000,000 cases.

It is computationally infeasible, and wasteful, to let the machine to look at all 1,000,000 cases each time it adjusts its weights a little bit. The practical approach (which also has certain theoretical bases) is to divide the large number of data cases into many **mini batches** of a relatively small number of cases, say 100 or 1,000, and let the machine look at one mini batch and adjust to that mini batch each time.