Contents

1 Bug machine **1**

2 KinPix **3**

Two basic open problems that will get you thinking in new ways

1. One is about the benefits of diversity, group learning, and certain algorithms that employ electro shocks

2. The other is about a GAN like game that your grandmother would like to play

1 Bug machine

We want to classify tomatoes: they're either rotten or not rotten (±1). Tomato bugs are experts who have the capability of knowing from a distance whether tomatoes are good.

Put one tomato bug in each box. They can only see out the front of the box; they don't see each other.

You want to train the bugs to classify tomatoes. Train as follows. Put an electroshock on either the front or the back depending on whether the tomato is rotten; the bugs will try to avoid the shock. After a while they learn to classify.

On a new tomato, look at where the majority of bugs are.

If I want to do this for a long time, how do I prevent the tomato bugs from being lazy and just staying where they are?

Use the minority bashing algorithm: if the large majority is in the front, put the electroshock to the back and vice versa.

I claim this is stable.

This only works if you have variety. If all bugs are identical, whether they do the right or wrong thing, they agree. The "oompf" comes from variety. They coax each other. This is some type of "cotraining".

What are the benefits of variety for stability? My philosophy about the world centers around that (evolution, multiplicative weights). Machine learning algorithms have tricks (ex. sleeping) to preserve variety. Variety makes it stable/interesting.

If one guy decides to fall asleep, it will get shocked.

Ex. "bugs" are neural networks trained up to a certain start with a subset of examples.

If the concept drifts, the whole system might adapt. The bugs coax each other.

What could be a danger is that they become similar but not smarter. There is a general tradeoff (cf. politicians) that they cause competition, pick the best one, or they can step back and create subclusters. There's a tradeoff between short-term (harvest best) and long-term (develop variety, so when society changes, you're ready).

We want to rule out concepts which are far away from the target. Replace by vote of guys that survive.

What conditions can you hope for?

What is a measure of variety? How do you prevent things from becoming too homogeneous?

As a superorganism, the collection of bugs hopefully is smarter than each individual bug. Cf. boosting; the bugs are weak learner. You want lots of weak learners; that is more beneficial than the perfect learner.

This is true for evolution in general. You do not want one thing that is very good. That's monoculture, the one that gives the highest yield right now.

Imagine you have lots of linear learners, with different views of the plane. They're somewhat close. Minority bashing exhauses the variety. Can you diversify again?

If you overtrain, you always go to 0. If you overfit there is no adjustment.

How do you diversify? In neural network you can throw in noise. How do you measure variety? Initially there is huge variety. Stop at the right point.

The only way the bugs can communicate is via the learning task; they can't see each other.

SD: Each time someone is right, reward by the fraction who got it wrong?

RG: Game theoretically, each still want to get it right.

If there are several tasks, and they are not able to get it all right, they will be incentivized to get right what others who get it right.

There may be mixed strategies... (randomly say the wrong thing $\frac{1}{k}$ of the time...)

What types of algorithms produce diverse ensembles? Evolution uses multiplicative updates (concentratino is multiplied by survival factor). Exponentiated gradient, etc. correspond to algorithms in nature.

What happens quickly: hone in to best one. If you have survival vector of 0.9, it's quickly better than the 0.8. This is the curse of the multiplicative update; it exhausts variety. This is a deep topic in a societal context.

Metaphor: species separated by mountains. Break the mountain range. They exchange. After competition and dieout. The mixing has the disadvantage of losing variety. In the short term it's also good to mix everything. In the long term, you want separate populations that have variety. In genetic algorithms people use localization.

Throw bacteria in solution: three species survive in three niches: on bottom, middle, on surface. After mixing, only 1 survives. Nature has many ways to preserve variety. Physical boundaries is one of them.

Preserve variety by lower-bounding weights. The best algorithm is the **Winnow algorithm**:[1] learn junctions, take product. If you are above the threshold and correct, you don't update. This is crucial! Otherwise, one guy wins. The key thing is to make the update conservative.

If you have 1 constant task, it's best to train everyone on all examples. If you have lots of training examples, it is not smart if environment changes to overtrain them.

AR: How long is the horizon? Cf. follow the regularized leader.

One way to ensure variety is to regularize.

HL: Measure variety by how much disagree when distribution changes.

Ideally, all decisions should be made slightly $> \frac{1}{2}$ but correct.

[1]`https://en.wikipedia.org/wiki/Winnow_(algorithm)`

One nifty trick in nature: linkage. Genes do not freely float in nucleus. They link in genes. This *preserves* variety. If you let genes compete separately, one would win. If they link, they have to cooperate.

Restricted class in learning: if you get something right, you get something else wrong?

Suppose you have 30%A, 60%B, 10%C. You don't know the mixture, and want to make more of it. Use PCR: Take your strands, double everyone. This is classical multiplicative update; each has a slightly different factor! How do you make more of the soup? You need a trick!

Take these strands. Make a very long strand where you have roughly the same proportion of A, B, C. Take one of long strands and copy that all together. Make any amount you want. Split apart again at gluing. Use linkage to prevent multiplicative update. This is why we have genes: they are linked together so the variety is not lost.

2 KinPix

I want to design a game my mother would play. As a good mother she would look at family albums and recognize kin. Let's design a game around that.

Put a bunch of parent picture and one child picture; you have to find a match. Mother's task is to find the match. System's task is to pick a typical pair plus guys that look alike. (Cf. GAN: two players against each other.) Find decoys that look similar.

Biologists have theories: it's easier to recognize fatherhood than motherhood.

If we had this game at large we can test this theory.

It's not important that it's kin. You can put people and pets, women and boyfriend, etc. It's GAN-like.

You can only model relationships between pairs. What information do you get? Do you want to learn from the game?

Suppose A, R match.

If match is found, the decoys were bad. Move the decoys away, A from R. If person made a mistake Q, can move closer.

Doing GANs where discriminator is humans. When a new guy comes in, you want to do pretraining. NN being discriminator is still important.

If R is certain ethnicity (red hair), then match with decoys that also have red hair.

Do humans have a chance? Then weed the database.

Cf. game Dixit. Give a description of card; others try to put decoys.

HL: Give confusing multiple choice test.