EPISODE 47

[0:00:11.4] SC: Hello and welcome to another episode of TWiML Talk, the podcast where I interview interesting people doing interesting things in machine learning and artificial intelligence. I'm your host, Sam Charrington.

Before I introduce this week's guest, here is a quick reminder about the upcoming TWiML Online Meet Up which will be held tomorrow, September 12th at 3:00 Pacific Time. The discussion will be led by Nikola Kučerová who will be presenting Learning Long-Term Dependencies with Gradient Decent is Difficult by Yoshua Bengio and Company. Of course, it's best if you've taken a look at the paper in advance, but even if not, you're sure to learn something from the discussion. For more information or to register, visit twimlai.com/meetup. See you online tomorrow.

If there's any one topic that I've received a bunch of requests about covering on the show, is the subject of this week's discussion; evolutionary algorithms and machine learning. My guest this week is Risto Miikkulainen, professor of computer science at UT Austin and vice president of research at Sentient Technologies. During our talk Risto and I discuss some of the things Sentient is working on in the financial services and retail fields and we dig in to the technology behind them, evolutionary algorithms, which is also the focus of his research at University of Texas. I really enjoyed this week's interview and learned a ton and I'm sure you will too.

Before we get to the show, a word about this week's sponsor, Cloudera.

You probably think of Cloudera primarily as the Hadoop Company, and you're not wrong for that, but did you know that they also offer software for data science and deep learning? The idea here is pretty simple. If you work for a large enterprise you probably already have Hadoop in place and your Hadoop cluster is filled with lots of data that you want to use in building your models, but you still need to easily access that data, process it using the latest open source tools and harness burst of compute power to train your models. This is where Cloudera's data science workbench comes in. With the data science workbench, Cloudera can help you get up and running with deep learning without massive new investments by implementing an on-demand self-service deep learning platform on your existing CDH cluster.

To learn more about the data science workbench, visit twimlai.com/cloudera. For a limited time, Cloudera is offering a free drone to qualify participants who register for a demo. Again, the URL for that is twimlai.com/cloudera.

Now, on to the show.

[INTERVIEW]

[0:03:11.4] SC: All right everyone, I am on the line with Risto Miikkulainen, who is vice president of research at Sentient Technologies and a professor or computer science at University of Texas at Austin. Risto and I recently tried to connect at the O'Reilly AI Conference in New York. Unfortunately we weren't able to do an in-person interview, but we were able to get on the line together and I'm really looking forward to this conversation and I think you'll really enjoy it because we'll be talking about something we haven't talked about yet on the podcast, and that is evolutionary algorithms.

Risto, welcome to the podcast.

[0:03:52.7] RM: Thank you. Blessed to be here.

[0:03:54.0] SC: It's great to have you on. Why don't we get started by having you tell us a little bit about your background?

[0:04:00.7] RM: Okay. Yes. I got my Ph.D. at UCLA, 1990, and was working at the time on neural networks which I've continued doing whole my career, but also interested in evolutionary algorithms about the same time and have gradually drifted more in that direction and especially in the intersection of those two things. We've been evolving neural network since the early 90s and you can view that as one way of training neural networks in domains where you cannot really do other types for learning like deep learning, supervised learning.

That's been my research focus. Originally, I did a lot of work in natural language processing with neural networks and then understanding the visual cortex with neural networks, but most

recently this evolution-based work is focused on building robotic control, game characters and, more recently, solving problems in the real world using this technology. That's what I'm doing at Sentient. I'm on leave from UT Austin, have been for two years. Trying to take the technology to the real world, and it's been a lot of fun.

[0:05:04.4] SC: Tell us a little bit about what Sentient does and what the focus is there.

[0:05:08.0] RM: Sure. Sentient has been around actually for 10 years, although it came out of stealth only three years ago or two years ago.

[0:05:15.2] SC: That's a long time in stealth.

[0:05:16.0] RM: Well, it took a while to develop all the infrastructure. Initially, Sentient was doing stock trading and still is doing stock trading as one of the applications. It's a great first application, because you don't need to have much customer support and other things like that. You just run your algorithms and they trade directly and make money, but it took a long time to build those algorithms and also build the infrastructure, the computing infrastructure. Evolution is one of those technologies that requires a lot of computing power to excel. Here, at Sentient, we built that kind of computing power or distributed system across the internet using at that time two million CPUs to evolve stock traders. That was very successful and it was a lot of fun to build that. That was the first product.

Then we changed the course, so added another direction rather going to eCommerce building intelligent interfaces to eCommerce. There are two products there. One of them is SentientWare which is a visual interface to catalog a product and that understands what the user wants. It's more like a personalized assistant that you might find in Nordstrom and other fancy department store. You click on choices and their system will understand what it is that you're looking for and a couple of clicks will get you what you want.

The other one is Ascend, Sentient Ascend, which is a way of using evolutionary computation to design web interfaces so that they are most effective. For instance, optimizing conversion rate on an eCommerce website is one of the applications to that technology. Those are the three products.

[0:06:55.3] SC: Interesting.

[0:06:55.5] RM: Behind all that we've developed — We are a technology company, so we develop evolutionary computation methods, deep learning methods, combinations of those, as well as surrogate optimization, backpack optimization in various domains. That's our technology pace.

[0:07:10.8] SC: Okay. Actually, I'll get back to that. You mentioned 2 million CPU cores. Is that via the cloud? Is that distributed? Are those in your own data centers?

[0:07:25.0] RM: Yeah. That's interesting. This is a great machine, sort of like folded or study at home. Yes, exactly. Same idea, utilizing freely available compute time — I mean not freely available, we pay for it, but available idle compute time around the world. This is, I think, a really interesting proposition and there's so much compute distributed now in people's laptops, PCs, cellphones. If you could harness that, we could do a lot of computing.

Of course, things change. The CPUs went perfectly good in the first several years when we're evolving stock traders. More recently we evolved deep learning neural networks and the need is a little different, because they are trained on GPUs. Now we have to have access to GPUs and the demographics has changed a bit. This is a very rapidly changing space. It's also become very economical to buy a bunch of GPUs, thousands of them perhaps and build your own center. We are continuously looking for opportunities to harness computation, whatever form it is, but we have expertise in those kinds of setups.

[0:08:31.6] SC: Okay. Is the company still using this highly distributed grid-like infrastructure or have you shifted to an owned infrastructure?

[0:08:42.2] RM: Yeah. We are still using it and we're also exploring other ways of getting the computation. They serve a little bit different purposes, one of them is very massively parallel and not necessarily that reliable, and that's good for some applications. In others, you need more reliability. The compute has to be accessible all the time and it has to finish. Therefore, you need different kinds of sources for compute and we are pursuing several sources as a result.

[0:09:09.7] SC: Can you give us examples of the kinds of applications that work well for each?

[0:09:15.1] RM: Yeah. For instance, the stock trading is very robust, because we are evolving traders, and evolution in general, evolution in computation in general is robust, because you have population of solutions and you are testing that population in a distributed fashion across the different compute sites. If something goes wrong and you don't get back an evaluation of one of your candidates, that's okay. You have hundreds of other candidates. It's okay to lose one candidate or be partially evaluated, because there's a lot of noise in this evolutionary search process and it actually thrives on diversity and multiple alternatives. It's not dependent on any single alternative or an accurate evaluation. It's based on evaluating a lot of candidates and many times. That you can do on a compute source that's very unreliable.

Now, if you are building a particular application based on, say, deep learning, you want that deep learning network to be very well built and very well trained and reliable. Sometimes that training will take several days. It has to be a reliable source so that you can get back that result of the training otherwise you waste a lot of time on it. That's where we need more reliable sources.

[0:10:35.3] SC: Okay. Is the company still working on financial services or did you migrate or pivot from financial services to the eCommerce solution?

[0:10:46.1] RM: We're definitely working on trading as well. It's more like an AI-based hedge fund than finding some services. There we have expanded to different markets. That part is growing and the algorithms have also evolved to some degree, but the base is still to run a hedge fund using AI. The traders are automatic. They actually are looking at the stock market and deciding what the trends are and what should be done and then make those trades.

Of course, in the end, there is a person also watching this with a hand on a red button in case something goes wrong, but that almost never happens. I don't think it actually has happened in that urgency. Sometimes, also, the stock market is interesting that there might be something happening in a stock market. You can tell that, now, this is not a usual situation. It's going

haywire and we'll stop the trading and let it come back and become more normal. There's a human in the loop, but it is interesting that AI can do much of this on its own.

[0:11:45.6] SC: Are you trading your own book or are you working with other hedge funds and helping them trade?

[0:11:53.7] RM: Yeah, currently trading our own funds, and there's a plan to open it up in the future for investors.

[0:12:00.3] SC: Interesting. I wonder why if you've got that working, it's kind of this perpetual money machine, right? Why even bother with retail?

[0:12:10.8] RM: That's a good question. In the AI world and in the startup world and in Silicon Valley it's always grow scale, get more, and this was a successful, is a successful technology. Now the question is what else can you do with it? Indeed, it's nice to have several different technologies and basis, and especially then because you start getting cross-pollination of them.

The second one we looked at was very different from evolutionary algorithms was deep learning, and we found that we could use it to encode human preferences, what humans perceive as similar and visually and identify that this would be something that the retail industry could really use, because retail is changing rapidly and fundamentally. From the brick and mortar stores, we are changing into internet-based commerce and this is a really big change, but internet commerce is quite clunky still today. It's hard to find what you want in those web interfaces.

The reason that was pretty obvious was that you have to know what you're looking for and you have to know the keywords for those ideas that you're looking for. If you do, you may be able to find to your satisfactory some product that you're interested in, but if you don't it can be very frustrating. As a matter of fact, we've done some tests and that typically in an eCommerce site, in a week, 15% of the catalog is actually seen by the users, because they have to go through these rigid categories and keywords and it's very hard to find different variability or diversity in a catalog.

We identified very early on that this is actually something that could be done differently. We could use human-like perception of similarity, visual similarity. If you're thinking of, say, a catalog of shoes or jackets or sunglasses, then a human can identify something that he likes, or he or she likes very quickly by being presented a bunch of alternatives, "This is the one I like the most." Then another set of candidates comes up, and these are now based on the first click. They assimilated the first click, but variation around that area, and now you can click again and it turns out in about 7 clicks you can find anything in catalog according to our evaluation.

In that process the users actually see about 70% of the catalog each week, and this is the situation where everybody wins. As a user, you have access to more variety and the eCommerce vendor will sell them all their catalog and the manufacturers get their products out there for people to see. It is just fundamentally a better access eCommerce catalogs, and it's based on understanding human perception and training machines telling neural networks to represent those similarities.

[0:15:04.4] SC: That sounds like a really interesting application. Actually, you said the financial services you started with applying evolutionary algorithms and with this eCommerce application, the focus is more on deep learning? Is that the right way to think about it?

[0:15:23.6] RM: Yes. That's exactly right.

[0:15:25.3] SC: Maybe let's dive into evolutionary algorithms now. You gave us a little bit of a taste of it earlier, but what does that mean?

[0:15:35.7] RM: All right. Evolution — You can see this as form of simulative reinforcement learning where you do not get the correct answer to learn from. In deep leaning, and most of the machine learning applications that are out there are based on these large datasets where somebody has collected the data. This is a situation and this is the right categorization or action at that situation. For instance, images and their categorization, or speech and the transcription of the speech, and this is very doable and we have systems that have come up, Mechanical Turk and Mighty AI and others who are now collecting and forming such datasets. Then systems like deep learning can be invoked to build models of those kinds of tasks. Then it's very powerful.

But there's a lot of task in the world where those actual correct answers are not known. For instance, driving a car, mobile robot navigating, playing any kind of game, you don't know what the actual optimal answers are. In such domains the learning is based on exploration. You try out things and see how well they work.

In a big picture this might be called reinforcement learning, you get reinforcement to your actions. Now, reinforcement learning actually as a term refers to a category of class of algorithms that is a little different from evolution in sort of social or a traditional sense. Those reinforcement learning algorithms are mostly based on value function approaches, where you list all your actions and learn how good each action is in each state, and that's a different approach to solving a problem when you don't have gradients, when you don't have supervised targets.

Evolution is a different approach. The idea there is that you have a whole population of solutions and you evaluate each solution, and as in biology, those who evaluate well get reproduced more and generate offspring. The offspring is somehow generated in various ways from the parents. You take an encoding of the parent, an encoding of another parent and then you cross them over, just like in biology. There some kind of a linear encoding, like DNA, and you can take part of that DNA from one parent and another part from another parent to form an offspring. That means that you are combining properties of both parents into the offspring, and this is the fundamental idea of the evolutionary search. You are trying to recombine components or schemas or building blocks they are sometime called to find better combinations of them.

There's another component which is important as well, that's mutation.

[0:18:13.3] SC: Before you dive into that, I have a question about something you said about reinforcement learning, and that is you said that the reinforcement learning is characterized by these value functions and in order to evaluate these value functions, we apply gradient-based approaches. You said that in order to do that, we can only do that under supervision of some sort. Elaborate on that, because we're typically applying, as you said, reinforcement learning in what we think of unsupervised problems.

[0:18:48.4] RM: Right. Supervised learning is different from reinforcement learning. That supervised learning means that you have correct answers and we can calculate gradients. Reinforcement learning, we don't have those gradients. We don't have correct targets. It's based on exploration. In reinforcement learning, the agent will try out different actions and will eventually get an evaluation on how well those actions worked out and then incremental dynamic programming is typically used then to propagate that reward back to the earlier actions and changing their value in this representation of how good each action is.

It is not supervised. Things get a little confusing, because this, in its purest form, this kind of value function learning applies to a table of actions in a table of states, but that is very limiting. So we need to be able to approximate a large number of actions and a large number of states, and that means that a lot of times we use a function approximator, and then use gradient descend to build that function approximator using the propagated values as targets. We use gradient descent as a way of generalizing reinforcement learning, but the information for reinforcement learning comes from the outcomes of these exploration episodes.

[0:20:12.4] SC: Right. One of the challenges with reinforcement learning is that you are — What's the right way to articulate this? You're kind of weighing exploration and exploitation and your exploration tends to be focused around fixed paths and you don't really have a built in mechanism to combine different exploratory directions, and that sounds like that's one of the advantages of evolutionary algorithms?

[0:20:40.9] RM: Yes. This is absolutely right. The two main challenges for reinforcement learning, one of them is this lot search space. Like I said, it works really well if you have a smaller space where you can enumerate all your actions and states and it's difficult when those grow and become maybe continuous. Another problem is that it works well when your state contains all the information from the past. It's uniquely specified, or exactly specified.

If you have partially observable states, then it becomes very difficult to learn using these kind of value function approach. Both of those are actually addressed if you evolve, use evolution to construct neural networks in such tasks. Neural networks work very well when they are in continuous domains. You don't have to enumerate all the possibilities. They're already doing the function approximation.

Regarding the second problem, you can evolve recurrent neural networks. When you have recurrence, your entire sequence of actions that you've taken up to this date is taken into account in making the next decision. Therefore, it can disambiguate much of the state ambiguity and therefore work better in partially observable problems as well.

That is true. You have to have a decision making system such as a neural network as a base, but then you can evolve that neural network in order to get over these two challenges to reinforcement learning.

[0:22:12.5] SC: One challenge to reinforcement learning is the issue of reward attribution. Is that covered under the partially observable state issue or is that a separate category of challenge and how is that addressed by evolutionary algorithms?

[0:22:31.4] RM: They are related. Reward attribution, if you have a partially observable problem and you make a decision, part of the credit for the success for that decision depends on how you got to that state, those unobserved variables. If you have a system that take into account all the actions that got you to that state, you are doing the current assignment better and more accurately. Yeah, in that sense, they are related.

[0:22:57.3] SC: Okay. Is there a way to, granted that a podcast is limited in its bandwidth, not having a visual component here, but is there a way to kind of walk us through the setup for an evolutionary algorithm and how it's applied to training neural networks?

[0:23:21.2] RM: Sure. I've been vigorously waving my hands here, but you haven't been able to see it all along. Yeah, the big starting point that's different from reinforcement learning is the population. You have a collection of individuals, and typically it's 100, 200 maybe. In some cases, in different variations might be smaller.

[0:23:41.0] SC: What is an individual?

[0:23:43.0] RM: Yes. Each individuals represents a solution, a potential solution to the problem, like in this case, in the simplest case it could be a neural network and it means all the weights and all the nodes, weight values and all the nodes and the structure of the neural network.

[0:23:57.4] SC: Okay. It's pre-constrained by architecture. You've defined an architecture for this target neural network and you're using the evolutionary algorithm to figure out its various parameters as supposed to evolve the network architecture itself.

[0:24:14.5] RM: Well, that's the simplest way of doing it, but the most powerful evolutionary neural network systems actually evolve the architecture as well. You just have to have a representation that can be encoded into a string, typically, or a tree, but most often a string, like DNA. You have a DNA representation of that solution and it may include the weights on the connections, may include the hyper-parameters, like the slope of the sigmoid, something like that. It may include the topology, like how the graph is actually connected between nodes. It's just an issue of how you encode it.

The end result is that you will have some kind of a string representation or a tree in a more general sense for each of the individuals, for each of the neural network, the solutions. Now you have a population of them. Then on that population you run an evolutionary algorithm and there are many flavors of those. What I've been talking about is the generic algorithm flavor, John Holland's tradition. There are many others too.

That is the most closely associated is biology, and that you take each individual, you test them in the task. You evaluate them in a task, and that's where the parallel computing comes in because you have a population, you can send each individual to a different machine across the internet to be evaluated. Sometimes most of the time the evaluation is the most computationally expensive part of the algorithm, because you may be driving a robot or you may be doing stock trading or whatever it is that you're doing. It takes time to evaluate. That's really nice, because evolutionary algorithm is parallelized very well in that sense. Each individual can be evaluated separately.

Then you get back to your values, the fitness values. How well they perform in the task? Now, the entire population is — Each one individual is associated with a fitness value, and you can

find the best ones and you can find the worst ones. The worst ones you throw away, and the best ones you pair up so that you can do crossover, as I mentioned earlier. Also, the second component of evolution, the mutation.

Crossover gives you new combinations of building blocks or schemas, partial solutions. Mutation creates new ones, because it's not necessarily — You usually initialize the population randomly, like random weight neural networks, but it doesn't guarantee that you have the weight in a right place when you need it. Mutation is a mechanism for creating that. It's a random change in the weight value, a random change in the topology that happens on a low probability, about 2% or 4$ or so as part of that evolutionary step.

You take in your parents and you created their offspring, and that offspring then is used to replace those poor individuals that were thrown away, and that creates your new population, a new generation, and then this process repeats.

In this process, in essence, you are doing a parallel search in the solution space starting from your hundred initial random individuals who gradually reward those that perform the best and you perform more search in the areas where there's better solutions. It's in parallel, so you still not following just one potential kind of solution, but you are in parallel focusing on multiple potentially good solutions. Eventually, then you find some really good ones and the algorithm focuses on refining that area and finding the absolute best in a smaller area around the best solution. That's what ends up happening.

That's evolutionary algorithm in a nutshell. As I said, there are many variations, and they're also interesting to talk about, but that principle of parallel search in a solution space, which is directed by these periodic evaluations in the real world is the essence of the method.

[0:28:02.1] SC: Okay. It sounds like perhaps not if applied to a deep neural network, but if implied to a more simple type of machine learning model. Essentially, what we're doing is a search of the solution space that could be analogous to a grid search of hyper-parameters, but in the evolutionary world we're able to constrain our search and as well as parallelize it. I'm imagining that from like a big-O perspective, this is log in time as supposed to N or something like that with a grid search. Am I thinking about that the right way?

[0:28:46.8] RM: Yeah. Intuitively, I think that's correct. It's of course a different question of can we prove such results? That is in essence what's happening. It is not laying all your eggs in one basket. It is pursuing multiple alternatives at once and it's then gradually focusing the search where the most promise is. For us creatures, it's just brute force approach. You spread all your individuals as wide as possible and find something in there and we end up wasting a lot of time and also not necessarily finding the very best one, because you are limited by the grid.

In that sense, if you could think of it as more intelligence version of that, much more intelligent. It's actually sometimes amazing how efficient it can be. We've done, at Sentient, a couple of demonstrations in a multiplex of domain, which is a very easy domain to solve symbolically, but you can create it or turn it into a search problem. You have to find the right encoding of bits so that you get the right answers to any given address from your bit string.

Now, as a search base, it's huge in this case. We've solved 70 bit multiplexer, which the search base is 2 to the 2 to the 70. It's a very lot search space and, still, evolution can find solutions quite quickly in a hundreds of thousands of millions of trials. You'll find solution in that magnificently large space.

[0:30:14.6] SC: Can you take a step back and elaborate on that problem and how it's applied or how it's used in practice?

[0:30:20.6] RM: Right. Multiplexer, it's a benchmark problem. It's not something that you would solve using evolutionary algorithm. It's only used to illustrate or evaluate the different algorithms and how it works, how well they work. You have a number of address bits and then you have a number of data bits, so that is your individual. Then what you are learning is rules. How to map those address bits to a data bit? That's what the multiplexer does. Given an address, it gives you one of the database.

You can evolve it to solve that in various ways. You evolve rules, and the rules express that, "Okay. If we have this, this and this bit in the addresses, then this is the output." Now, this space that you're searching is the set of all rules sets and that space is humungous, and we can calculate how large it is.

There's a work done by John [inaudible 0:31:14.3] a long time ago just to demonstrate how effective learning algorithms can be, evolutionary learning algorithms can be. At that time, I think we've been considering 11 multiplexer, which has a search base of 10 to the — I think it's 616. You can solve it by searching this set of rules and in a rule you have a certain way of identifying the inputs bits and then and'ing and or'ing them and so on, performing logical operations. So that in the end you get the data bit.

You can calculate how large that search space is. How many ways there are to combine these input bits, the address bits and the logical operations on them? Therefore, you can estimate how large the search base is and how hard the problem is.

In 11th case, it's 10 to the 616th, and we expanded it to 2 to the 2 to the 70th, and it still works. That is, I think to me, is very amazing that evolution can very quickly identify what actually works. Those component solutions, those building blocks, and then put them together into a solution, and the search base is huge. That is power.

Not everything is of course amenable to evolutionary computation. This happens —

[0:32:30.5] SC: Before we jump in to that, you mentioned provability. Can you elaborate on the work that's been done there? For example, you mentioned in describing evolutionary algorithm that you cannot throw away the worst performing models or parameter sets, and then you mate, if that's the right word, the best performing ones. Is there formal proof that that's not susceptible to some local and minimum maxima, and you might get some better result by mating non-performers and performers?

[0:33:07.9] RM: Yeah. There's quite a bit of theory in evolutionary algorithms. It started already in the 70s with the schema theorem, and that says the schema theorem states that in this process, the shorter the schemas are, the combination of elements in the solution in that bit string, in that generic string, the shorter they are and the more powerful they are, the more strongly they actually effect the fitness of the organism, then more prominent they will be in future generations. They will become more prevalent in the population. That is a theoretical result. It's a scheme theorem. It shows that this happens in this mechanism.

[0:33:46.9] SC: How does a short schema apply to a practical problem? What does that mean?

[0:33:53.7] RM: Schema length means that you have — Let's think of a generic encoding. For instance, string of weights on a neural network. The schema means that a good neural network has this weight on this connection, this weight on this connection, and this weight on this connection. That's a schema of length three. It may be that indeed you have these kinds of interactions between weights in a neural network and you have to set them right in order for the network to perform well. The schema theorem just says that if they are short, small segments of weights, or segments of the neural network that are powerful, they are easier to find.

Now, if the schema covers a large number of weights, that's more difficult to find obviously. It makes sense, right? You have to set more numbers correctly in order to see the benefit, and the schema theorem just says that if you have short schemas, easy ways of gaining the benefit, finding this three weights of right values, that's going to be very easy to find and it will very quickly become prominent in a population.

From the point of view of biology, that makes sense. If there are some genes that are very short, I mean just the single genes instead of interaction with multiple genes, that's going to be very prominent and very quickly propagate through the population. The mathematics of schema theorem just expresses that idea.

[0:35:17.7] SC: When we're training with evolutionary algorithms, is the schema length constant or does this method kind of zoom in and zoom out to different levels of resolution?

[0:35:31.8] RM: Yes, exactly. It will do that. We don't know ahead of time what the schemas are like. We define the encoding, and you try to put in as much insight into that encoding as possible. You try to define a search base where you believe, first of all, that the solutions lie in that search space and also that it's easy to move around in that search space.

That requires human thinking and creativity, but evolution will then find you the best combinations of those elements in that search space, but encoding matters. In some cases it is obvious and its given by the domain very much, like in neural networks it's an obvious encoding

to have weights put together into a string, but then when you think about it some more you realize that, actually, the topology of the neural network matters as well, then you want to have some way of encoding topology and letting evolution discover different topologies. Then you discover things like, “Well, if in order to make it easier to search this space,” which now became much bigger because it’s all the topologies in addition to just all the weight values.

A clever idea is that let’s start with a very small neural network, a small search space. Initially, we just started with neural networks that connects inputs directly to the outputs. No hidden nodes at all, and let evolution run in that space for a while. Find good simple networks. Then we’ll add complexity. We had a hidden node, and another hidden node, and then we add recurrent connections and gradually discover complexity as we go. This principle, instead of starting with all kinds of topologies of neural networks as the initial population, if it starts small, if it starts simple and gradually complexify, evolution is much more powerful in finding these complex solutions.

That’s what I meant about being smart about how you encode it and how you let evolution search the space. It makes a big difference. This one principle has turned out over and over again to be very useful.

[0:37:32.0] SC: How complex can you get with this technique? If you come up with an encoding that can represent convolution layers and rectifying layers and the kinds of things that we do and CNNs for computer vision, can this thing come up with a very deep model, the types of models that we’re using for objective recognition nowadays, or is it more limited?

[0:38:02.1] RM: Yes. It’s a good question. There’s really been two approaches to doing this, and the first one I just described was that you have an encoding that in principle could encode anything, any kind of connectivity, and you build up from that. That is actually very good on task like the reinforcing learning problem where you have to define a custom kind of recurrency that retains just the right information overtime in order for you to disambiguate the possibly observable problem.

Very recently, in the last two years or so, maybe — Yeah, two years or so, another approaches emerged, and that’s specifically to evolve deep learning architectures. There’s an interesting

difference and that those architectures are composed of specific components. You mentioned different convolutional neural networks, LSTM networks, dropout as a parameter, max pool layers, all kinds of structures now exist that people compose these deep learning architectures from. It now makes sense to do this a little differently, because we know there are some components that are useful. Well, let's give those components as a source material, raw material for evolution.

Now you can come up with a mechanism that maybe operates in a couple of different levels. You could evolve the weights, you could evolve the components, and then you could evolve the overall topology that's based on those components. This is currently where the deep learning neural evolution is evolving deep learning networks. There are multiple approaches, but by enlarge they do this. They evolve their hyper-parameters of those networks and maybe the topology of the networks, but then the weights are trained using a supervised training set, like image recognition, you mentioned.

There's still a big benefit from evolving those deep learning networks even if you are applying it eventually to a supervised problem, because it's very hard to construct the right topology for your problem. Let evolution do that, and then you use the training to set the weights. There are millions and billions of weights, but potentially it's very hard for evolution to get every single one right if it needs to do a mutation to get the weight value right.

Deep learning is much better to doing that, but the topology matters as well and the architecture of the components matters as well and hyper-parameters matter, and that we can optimize using evolution.

[0:40:31.6] SC: Help us understand — What you just said was the evolution is better for the architectural, identifying the architectural solution, the connectivity and the types of layers and things like and traditional deep learning training techniques are better for the weights. Why exactly is that? You mentioned because there are a lot of weights, but I thought that was an advantage of the evolutionary approach.

[0:41:00.9] RM: It's actually very simple point that if you are evolving the entire network, including the weights, then evolutionary operators need to set the weight values, and that

means crossover or mutation, and mutation means that you are changing each weight randomly. If you have million weights, that's a very slow process.

In contrast, something like deep learning, or back propagation, stochastic gradient descent, every time you show an example, you can change every single weight. There's a lot more parallelism. It's a lot more efficient way of changing the weight.

[0:41:37.2] SC: What we're saying here is if you've got training data, use traditional training techniques, gradient descent, which uses that training data to accelerate training.

[0:41:47.7] RM: Yeah, exactly.

[0:41:49.1] SC: Where you don't have training data is kind of at this higher level. It seems like you kind of — If you have the training data you should also be able to use that for the architectural stuff, but I guess we don't have techniques for doing that, like gradient descent. That's where evolutionary comes in.

[0:42:09.4] RM: Exactly. That's exactly the point.

[0:42:11.0] SC: Interesting.

[0:42:11.4] RM: Yeah. Of course, people do research and try to break free of these restrictions and you could for instance use evolution for the weights as well even in deep learning networks, but there's an interesting approach that allows you to do it, and that is that you don't evolve every single weight separately, but you evolve, say, patterns of weights.

[0:42:33.8] SC: Evolution?

[0:42:34.1] RM: Well, it's a different level of evolution. We call it co-evolution of different level, co-evolution of topology, co-evolution with components, and then weights. The trick here is that instead of having to set each value independently using mutation crossover, you have some kind of a pattern that you are evolving, and that pattern, you use that pattern to derive the weight values. In extreme, that pattern could be given by a different neural network, a separate

neural network. You're evolving a neural network who's outputs then give you the weight values. That technique is called compositional pattern producing neural net and it's being used to evolve these deep learning architectures. This is an example of how you can still use evolution even if you don't have supervised training data necessarily. We'd still use a deep learning network with millions of weights, but you have to use this kind of an indirect encoding of its weights, perhaps through another neural network.

[0:43:33.8] SC: When you're using evolutionary algorithms to evolve the architecture and the connectivity of deep neural network or anything for that matter, are you applying evolutionary algorithms hierarchically or that as an approach hierarchically? First, evolve the connectivity and separately evolve the layers, or is it all done at once?

[0:44:03.8] RM: All of that is possible and all of that is under research right now.

[0:44:08.9] SC: I guess I should have anticipated that.

[0:44:10.6] RM: Yes, exactly. You mentioned co-evolution. That's a powerful approach in evolution and that means that you have two evolutionary processes, two or more going on at once and they interact. In this case it would be that you evolve the module, that might be a convolutional layer or LSTM type of a module, and then you evolve a topology, how you connect them together. That means that you have two population, one of them encodes different LSTM node and the other one encodes how they are connected. Then you can evolve them at the same time and evaluate them together as a single architecture and then the individuals inherit the fitness of the entire architecture. That's a very interesting approach.

You could also do it differently if you have a way of assigning a fitness to say a single LSTM node somehow, like maybe its memory capacity or something, then you could evolve that first and evolve a bunch of different, maybe LSTM nodes for different kinds of fitness functions. Use those are raw material at a higher level evolution and do it sequentially. That's also possible.

[0:45:16.3] SC: It almost suggests that there's probably some analog for GANs, generative adversarial networks, like generative adversarial evolutionary algorithms or something like that. Does that mean anything in this world?

[0:45:32.8] RM: Yes, exactly. That's how actually how GANs kind of got started and motivated.

[0:45:36.9] SC: Oh, really?

[0:45:37.6] RM: Yeah. It was possible to evolve input examples that broke the deep learning network that have been trained very well in a training set. Then Jeff Clune evolved these patterns that were pretty much just noise, but the networks still confident to claim that, "Oh! That's a dog."

[0:45:56.1] SC: Is this like the school bus draft kind of example, that kind of thing?

[0:46:00.4] RM: I'm not familiar with that example, school bus draft. What is that?

[0:46:04.4] SC: I'm maybe mixing up my objects and animals here, but there is some set of famous examples where for whatever reason if you give one of the same famous object detection, CNNs, picture of a zebra or a giraffe or something like that, it confidently proclaims it to be a school bus.

[0:46:25.0] RM: Oh! I see. I see. Yeah, there are many demonstrations of these same problem and that sounds like it's one of them. Yes. You can mix the categories, but the most impressive demonstration to me is that there's an image that look pretty much just noise. You cannot see anything in it, but still the deep learning system declares that to be — I don't know, a school bus or something else.

The origin of that was that it was possible to evolve those images. It wasn't just that we look at bad mistakes in a training set, but evolve these images that we had no constraints of what they had to be and they turned out to be pretty much just noise looking images, noisy images. Evolution discovered that this is a way of getting the deep learning network to perform very poorly, classifying its confidently something else.

[0:47:11.6] SC: That's a really interesting background. I didn't realize that.

[0:47:14.6] RM: From there on, the whole field of generative adversarial network started, like how could we actually do this? How could we have an adversarial training, trainer, or supervisor so that the network could actually perform better, because you could use those images to train as well as just break it. This is a very close relation, and I think it's still being explored. It's possible to evolve training sets and it's possible to evolve also these systems that are learning from them so that they become more robust and also they develop internal representations that are more representative. We can use evolution to bias these learning systems in a way we want. They can make it more general, make them interpretable and more robust.

[0:47:58.2] SC: Interesting. I interrupted you earlier when you're about to talk about classes of problems for which evolutionary algorithms are particularly suited and not suited.

[0:48:08.5] RM: Right. There's, of course, a lot of work in trying to expand a space of possible problems. Now, we were comparing reinforcement learning with evolution. I would like to point out that reinforcement learning has a little bit different perspective and goal. The idea there is to model a learning of an individual more or less during its lifetime.

As its living its life and it's performing, every step counts. In essence, it's like an online method. The typical application of evolution is offline engineering type of application, that you do have a simulator, for instance, of the systems — Or the environment for the system you're trying to evolve, and you can fail miserably in some of these candidates. The only thing that counts is that in the end you have a very well-engineered system. That is a different kind of a perspective. You don't get penalized for your exploration in evolutionary algorithms, you only get evaluated in the final result. Versus in reinforcement learning, it's a continual lifelong learning system perhaps.

That makes sense. Evolution is an engineering approach, it can be. We can evolve, for instance, controllers for finless rockets is one thing that we did. You couldn't possibly use it in a physical system because you would have to explode hundreds of thousands of rockets, but if you have a simulator for that system you can do anything you want of it. You can explore very wide solutions. As a matter of fact, some of those wide solutions develop into really good solutions in the end, and in the end we have a controller for a finless rockets that keeps it stable in a reliable way. That's the kind of typical application for an evolutionary algorithm system.

Now, on the other hand if you have an online system that needs to learn online while it's performing, that's not as easy to use evolution for that. You have to build extra machinery for it, but you can. That's the kind of a reinforcement learning system that reinforcement learning initially came from, but it has also been confused everybody. It has been used to do engineering design as well, just like evolution is used to do online learning. That's opposite to the origin, and that's sort of the first application would be engineering versus online learning.

[0:50:22.7] SC: You mentioned simulation and that reminded me of a question that I had earlier. Is there a relationship particularly in the context of the trading work that you've done between Monte Carlo types of approaches, the evolutionary approaches?

[0:50:38.3] RM: Yes. Monte Carlo's simulations means just that you randomize your domain and generate new situations that way. Now, when you'd use it as a solution mechanism, you're banking on the idea that even randomized solutions are likely to be successful sometimes. You could think of evolution as a 2.0 of that, that you're actually trying to learn from your mistakes. You're trying to learn from those trials and that is using crossover on a good candidates and mutation and a good candidates and focus the search more. That is, I think, a good way to formulate the relationship.

You could even look at some of the evolutionary algorithm methods. We've been talk about generic algorithms, which is crossover mutation-based. There are other evolution methods that are closer to something, like Monte Carlo, and that they will build a statistical model of the domain. Which individual components, schemas, how reliable they are in predicting the fitness. You form that probabilistic model of your search space, and then when you construct new individuals, when you construct the offspring, you don't do it based on crossover mutation. You do it statistically. You sample from that model. It's still a population-based approach, in that sense, 4th under the evolutionary algorithms, but it's closer to probabilistic reasoning and Monte Carlo methods and other statistical methods, and they perform quite well too

[0:52:07.3] SC: Oh. Interesting. Interesting. Is there a simple way to kind of characterize the various types of evolutionary algorithms or approaches or the various tweaks that have evolved to the basic generic approach?

[0:52:31.5] RM: Well, we can attempt to do it. Researchers are very creative. Whenever you come up with a category, then they will cross the categories. That's part of how research works too, that you will combine ideas across the different approaches and categories, but generic algorithms is one where the close connection to biology is obvious, crossover mutation. Then there are these statistics-based approaches like I mentioned. Sometimes called estimation or distribution algorithms where you estimate the probabilistic model of what works and then sample from that.

There are methods based on evolutionary strategy where you don't have crossover, but you have a small population and you're mostly using mutation to do the search, but you make the mutations intelligent. If you have a small population, you form, for instance, a covariance matrix of how your mutations co-vary. Try to find these interactions specifically, and then use that model of the interactions to construct new individuals.

There's differential evolution, there's different buzzwords. I can give you a lot of these, but there's a large number of them and they are based on sometimes letting go the biological analogy and instead focusing on the idea that what if you do have a population, how could you utilize the information in that population to construct new individuals better? That is kind of the general umbrella of evolutionary algorithms.

You can even think of an interesting direction trying to go towards the biology, going the opposite direction. Talking about abstracting to probabilistic reasoning, probabilistic and stochastic search, but you could also go towards biology and there's an interesting idea. Try to take some ideas that are more biological, for instance, indirect encoding. The fact that in biology DNA, the actual [inaudible 0:54:30.7] does not really specify the full individual. There's a large network of interactions that come after generating the proteins from the DNA and RNA.

Genetic regulatory networks are a huge part of biological construction from the DNA to an individual, and currently we are pretty much missing that in these algorithms. There's a lot of complexity in this kind of indirect encoding. There are approaches that try to invoke generic regulatory networks. There are approaches that are trying to include and develop mental phase, which is also big in biology.

After the individual is constructed, after it's born, there's usually a period of learning that happens, interaction with the environment, and that then constructs the final individual. For instance, human brain, we have 30,000 genes maybe in the genome. There's no way the brain can't be specified except at a core kind of instructive level, or pattern level.

Most of the brain structure is actually learned in an interaction with the environment. Evolutions produces a starting point for that learning. Combining learning and evolution that way is fundamentally biological and it's also something that we should be looking into and we are looking into it, people are looking into it.

[0:55:49.2] SC: This is really a fascinating area. For folks that want to dig in a little bit deeper or learn more, where is the best place to start? Are there canonical papers they should start at? Is there a resource that you'd like to recommend people take a look at?

[0:56:06.9] RM: Sure. There are some classic books that are about evolutionary algorithms and can find generic algorithm evolution computation books that are textbooks. Holland, Mitchel, Goldberg for instance, they tend to be a little bit old right now because the field is developing very rapidly. It's been exploding growth. But the typical sources on Scholarpedia for instance, and then tutorials that are appear in the main conferences I think would be a great source. There will be actually an Evolution Computation Conference, GECKO, starts tomorrow as a matter of fact in Berlin.

The first two days are tutorials, and lots of those tutorials are online, even some of the videos about the tutorials online. I think that those are really a great way to get started. It is a big field and a very diverse field. It's kind of interesting, this evolution drive and diversity, the algorithm's drive and diversity, but the field is also tremendously diverse.

That is a bit of a challenge that you may get lost in all these terminology and all the different approaches, and that's why I'm recommending maybe starting from one of the textbooks even if it's a little bit older is a good idea that gives you the perspective and then looking at the tutorials and maybe there you should have access to a lot of literature on the internet.

[0:57:29.7] SC: Are there tools and frameworks or do the deep learning frameworks, the TensorFlow of the world and alike, do they support — Have any kind of support for evolutionary algorithms or are folks rolling their own?

[0:57:46.0] RM: Yeah. There's support. TensorFlow I don't think currently has evolution component, but it is likely that it will in the future. It's an open source project and people are contributing to it, so it's very likely to happen. There's, for instance, ECJ, evolution computation in Java, that's a big effort, George Mason University. That software includes many different evolutionary algorithms and that I think is currently the best source to get started with, software system to get started with.

[0:58:14.1] SC: Great. Well, to wrap things up can you let folks know what's the best way to check in on you or to get in touch with you.

[0:58:24.1] RM: I have a name that's very easy to find if you spell it correctly.

[0:58:26.8] SC: If you can figure out the spelling.

[0:58:29.4] RM: Yeah. I have a website at UT Austin where much my research from my whole career is always there. I'll try to keep that up to speed. Of course, Sentient itself has a set of blog posts and webpages describing the technology we are developing here, which is evolutionary computation and deep learning. Those sites are usually quite well up to date. Then we have pointers to material and you could use those as starting point.

[0:58:56.7] SC: Okay. Great. We'll link to both of those in the show notes.

[0:58:59.9] RM: Very good. Thank you.

[0:59:01.0] SC: All right, Risto, thank you very much. This was amazing.

[0:59:03.7] RM: It was a lot of fun.

[END OF INTERVIEW]

[0:59:10.4] SC: All right everyone. That's our show for today. Thanks so much for listening and, of course, for your ongoing feedback and support. For the notes for this episode head on over to twimlai.com/talk/47.

I've got a quick favor to ask, if you enjoyed the podcast and especially if you like this episode, please take a moment to jump on over to iTunes and leave us a five-star review. We love to read these and it lets others know that the podcast is worth tuning into.

Another thanks to this week's sponsor, Cloudera. For more information on their data science workbench or to schedule your demo and get a drone, visit twimlai.com/cloudera.

I'd also like to send a huge shout out to a friend of the show, Hilary Mason, whose company, Fast Forward Labs, was acquired by Cloudera just last week. For more on Hilary and Fast Forward Labs, check out my interview with her at twimlai.com/talk/11 and my interview with the former president of that company, Kathryn Hume at twimlai.com/talk/20.

Thanks again for listening, and catch you next time.

[END]

