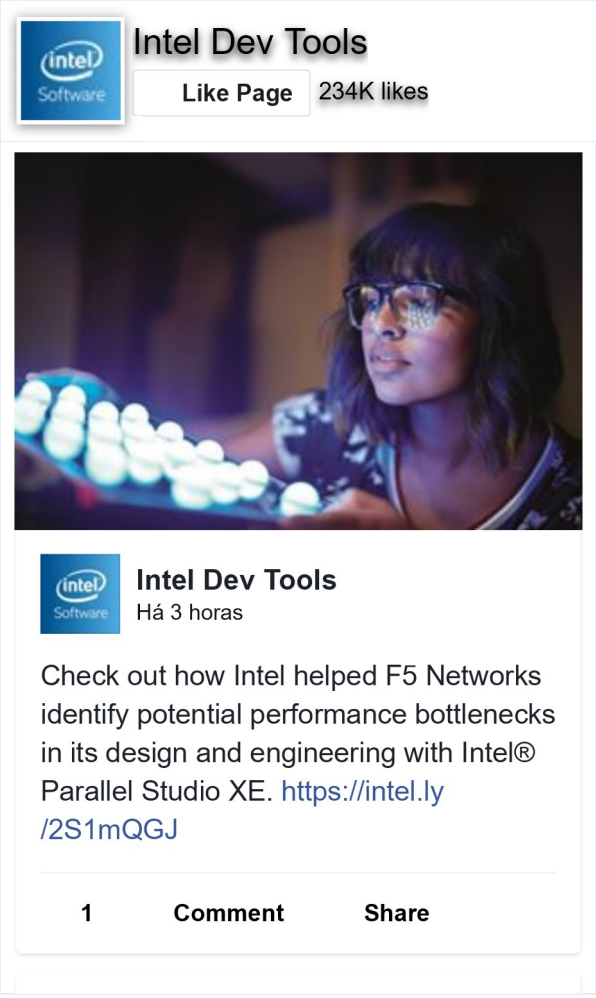
SPONSORED

When Your Network Fails to Converge



Despite our best efforts at designing and training neural networks, sometimes a particular network simply won't converge on a solution that is acceptable to the system requirements. It could get close, but not meet requirements. Or it could simply not even produce reliably consistent results, generating seemingly random outputs in response to the training data.

The classic example for this is probably 20Q - the online game of Twenty Questions.

20Q was trained by the players themselves - adding a question and category every time the game engine failed to guess the item in 20 yes or no questions. Because two to the power of twenty is a little over 1 million, the software should have generally been able to guess most of the most common English terms and expressions.

Imagine, however, that the people training the software hit a certain demographic — mid-30-something, affluent, educated professionals. Gathering the data online, the most common characters (a large percentage of the game) are names from "Seinfeld" or "Friends." The game maker takes this data and produces a handheld game, designed for kids eight to 12 years old. By the time it comes out, the most common names are characters from "Pokemon," and the app fails in play testing. These are real risks for machine learning, but they can be reduced.

How Network Convergence Fails

This outcome can happen because there aren't enough nodes to transform the input data into accurate outputs or because the architecture has to change drastically in order to better model the data. Perhaps you don't have enough training data, or the training data wasn't collected with data integrity in mind. 20Q, for instance, was simply a website that played a free game with no registration required, then used the players' answers as training data — anyone trying to "trick" the computer into giving strange answers would also be providing bogus training data. For a variety of reasons, you could have nodes in your network that never fire, which indicates that your initial design isn't a good model of the system.

Or we may not even have a good understanding of why we aren't able to produce good results in a particular network design. At this point in neural network engineering, what we do is just as much art as it is science. The choice of alternatives is driven less by a set of heuristics than it is by an intuitive feel for a technical approach that produces good results. Here are some things that architects have to consider if their results don't behave as they expect:

1. Keep the same fundamental design approach but change the network architecture. Add more hidden layers, or change some of the interconnects between layers. This might work, although it all depends on how the data correctly models the systems.

2. Look for other techniques to model the system, such as genetic algorithms. This

is the most intriguing possibility, because it involves an entirely different way of looking at the problem. Granted, the universe of problems that can be addressed through genetic algorithms is somewhat different than for many neural networks, but it could be a viable approach.

3. Reconsider the algorithms you are using to process data. If you start with nonlinear algorithms and fail to converge on an acceptable solution, consider using nonlinear algorithms such as hyperbolic tangent (Tanh) or other nonlinear functions.

Nonlinear functions can have problems of their own, depending on how you assign weights or prep your input data. Poor selection of weights or bad processing of inputs could leave your network vulnerable to not being able to find a solution. While changes in preprocessing data or changing the node algorithms and coefficients could provide a quick fix, it is more likely that a new network architecture is needed.

The key to building a good learning model is recognizing that it is an abstraction of the actual system. The model may not completely or accurately represent that system. In addition, that abstraction could well have flaws or leaks that prevent any solution from providing accurate results.

At this point, AI and network architects have to step back and look at what is working and what isn't. It likely won't involve minor adjustments to the existing network, but rather a complete rethinking of the approach and architecture.