Milestone 3 Partner Report — Neural Architecture Search

Overview

We have now applied DARTS to a number of data sets across different problem domains, including classification and regression on both standard ML data sets and novel scientific questions. The dream of NAS is to find a single tool that works well out of the box with minimal human intervention. While DARTS can indeed be a useful tool in a NAS, it is not a "silver bullet" that reliably achieves good results with default parameters for most of the problems we tried. It has proven to be a tool that can achieve solid results with some parameter tuning, especially once you get the hang of it.

We have seen that DARTS can fail in various ways:

1. Traditional hyper-parameters (e.g. learning rate) are not chosen correctly: MNIST
2. Dataset too simple and DARTS gives a model that is way too large: Graphene
3. Problem limited by training data & generalizability instead of model capacity, where DARTS itself is not enough to get top score: data augmentation for Galaxy Zoo

DARTS should be particularly useful if the model capacity is the major bottleneck in the problem, as in the case of ImageNet. But this is not the case for many scientific datasets we explored.

When evaluating the usefulness of DARTS against existing models, it is also important to remember that DARTS already incorporates prior human knowledge learned from popular models like ResNet: identical cells are stacked sequentially, and each cell is a DAG with skip connections inside. Those structures are already in the DARTS meta-model before running the architecture search for find out the optimal architecture parameters.

Current Stage

The hyper-parameter problem with DARTS on MNIST

Training a model with DARTS consists of two stages, an architecture search and training a full model with the learned architecture. The architecture search requires the user to specify a number of important hyperparameters including the set of primitive operations considered; the sizes of the cells; the number of layers in the model; the number of channels in the first layer; and the learning rates for the regular model parameters and the edge weights (architecture learning rate). The result of the architecture search is a set of edge weights for the normal and reduction cells. A discrete architecture is then selected by choosing the maximum weight from

each row of the edge weights matrix. The user selects a whole new batch of hyperparameters to train a model with these cells (architecture). Hyperparameters include the number of layers (typically deeper than during the search phase); the number of channels in the first layer (typically larger); and the learning rate (may be the same, may be smaller if the network gets bigger).

We discovered early that the default parameters had been tuned specifically to the example CIFAR-10 problem and did not generalize well to other problems, including the canonical benchmark, MNIST, for which training diverged because the learning rate was too high, as shown by Fig. 1. We changed the default optimizer from SGD to Adam and set the default learning rates to what we considered sensible defaults, 0.001 for parameters and 0.01 for architecture weights. The other parameters still need to be set using intuition and experience. The larger the data set, the more layers and channels you can use.

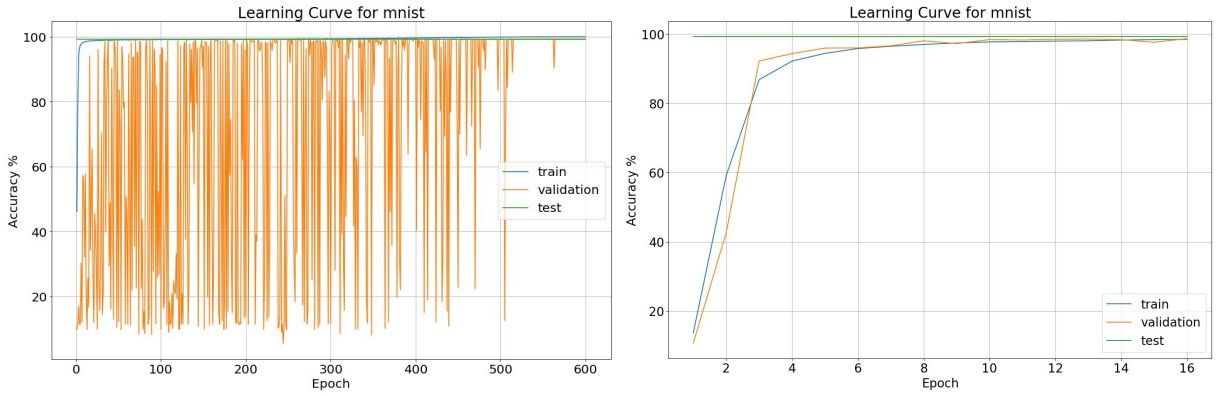

Figure 1. (left) DARTS on MNIST with the default learning rate used for CIFAR-10; (right) with reduced learning rate.

DARTS vs ResNet baseline on Galaxy Zoo data

After we found that the Graphene problem was too simple for DARTS style architecture search to help, we chose a second scientific data set that we thought would play better to its strengths: classifying galaxy morphology from images in the Kaggle Galaxy Zoo competition (https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge). This seemed like a good cross between an image classification problem, where DARTS has proven to work well, and a novel scientific problem. The results were interesting and not what we expected. The winner of the competition achieved an RMSE loss of 0.075, and during architecture search DARTS achieved a loss of 0.094. We thought this was quite respectable for a fully automated process. It's similar to the loss of 0.095 achieved by ResNet (the product of huge effort), and served as a solid baseline; if this had been a competition entry, it would have been in the top quintile of the leaderboard.

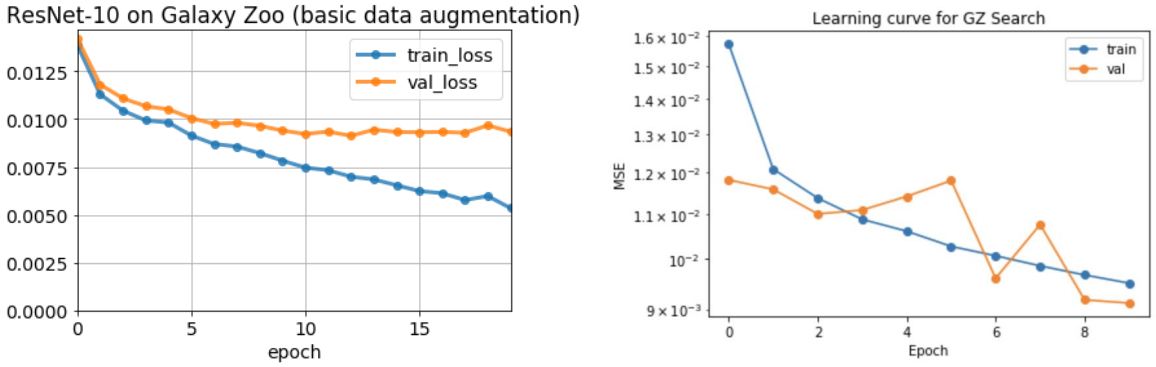

Figure 2. (left) ResNet-10 (halved the layers of ResNet-18) trained on galaxy data. (right) DARTS trained on galaxy data.

The gap between DARTS/ResNet performance (~RMSE 0.095) and the Kaggle winner solution (RMSE 0.075) is mainly due to the excessive data augmentation used by the winner solution (http://benanne.github.io/2014/04/05/galaxy-zoo.html), which fully exploits the rotation invariance of the galaxy image. In comparison, our DARTS and ResNet only have basic data augmentation (random rotation and flipping).

The "architecture discretization" problem in DARTS

The surprise came when we took the best architecture discovered and attempted to train a new model from scratch, using the discretization of the cell as proposed by the authors of the original DARTS paper. The results of this training were *worse* than what was achieved during the architecture search. We tried various settings of the number of layers and channels for the discretized-cell model, but were unable to achieve better results than the non-discretized-cell model. The summary of our results are below:

Models[1]	**RMSE**
DARTS (Search)	0.094
DARTS (Discretized, Same)	0.114
DARTS (Discretized, Small)	0.101
Random Search	0.098
ResNet	0.095

[1] None of these models use the Galaxy Zoo decision tree proposed by http://benanne.github.io/2014/04/05/galaxy-zoo.html

Table 1: Comparison of model performances in terms of root mean squared error (RMSE) on the validation set. “Same” denotes that the model uses the same number of layers and channels as the DARTS Search model (non-discretized). “Small” denotes a model that uses half the number of layers and half the number of channels.

A second interesting result on Galaxy Zoo was the competitiveness of the Random Search baseline. With a comparable amount of GPU time, random search achieved an RMSE of 0.098. We are slightly suspicious that the published comparisons of DARTS to Random Search might have been flattering to DARTS insofar as the authors did extensive hyperparameter tuning of DARTS before they did a single clean architecture search and training run. A more in-depth visualization of the random search for this dataset can be found in figure 3.

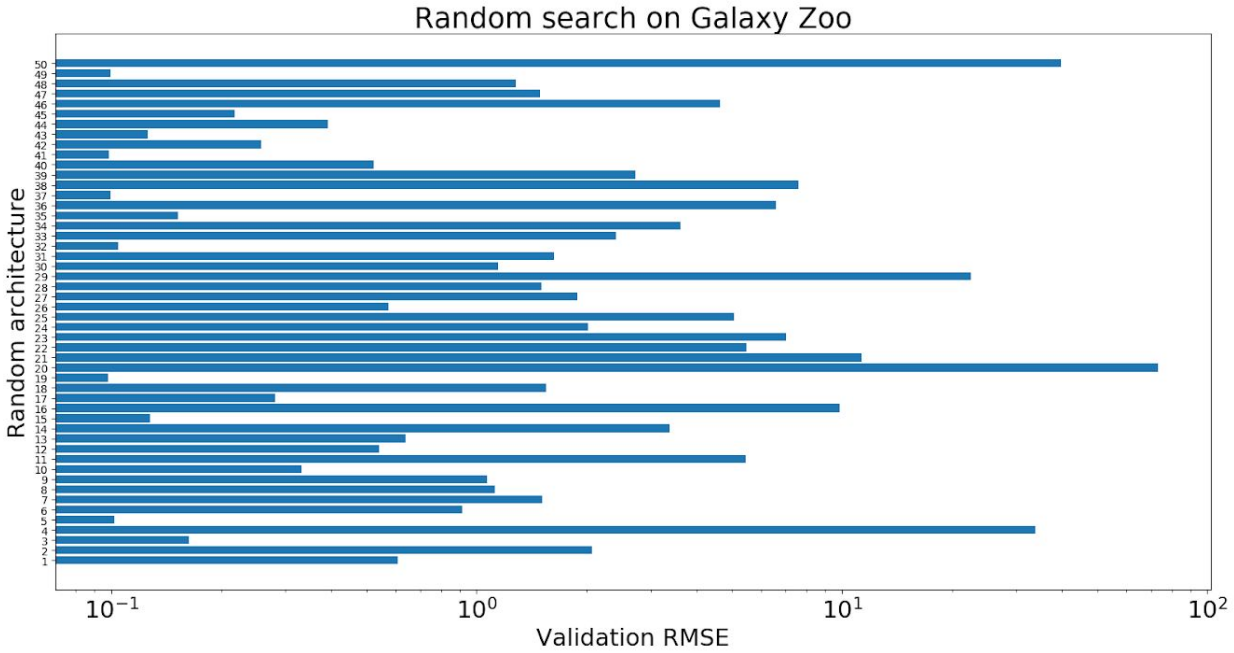

Figure. 3 Validation RMSE obtained after 20 epochs on 50 different random architectures. The mean of all this RMSE is 5.388 with a standard deviation of 12.428.

When the DARTS paper was submitted, a number of reviewers posed the question of why the heuristic of selecting the maximum of the edge weights as a discrete architecture should be expected to work in general. Clearly, if certain edge weights have been reduced to near zero during the architecture search, removing them will not hurt the results; but what about the cells where the second and third highest weights that might be far from zero? On the problems tested by the authors, the “argmax heuristic” worked well, but the GalaxyZoo data set is an example of where it doesn’t work. To our knowledge, this may be the first concrete example demonstrating this behavior. The intuition is that the network is really using the linear combination of features, and the highest weighted one is not dominant. While this is not the obvious “win” we might have hoped for, it is arguably more informative about how DARTS and similar styles of NAS behave. It also points the way to future plans as explained below.

Current Plans

Our experience with the failure of the argmax heuristic in the GalaxyZoo data set has motivated us to modify DARTS to make it more likely to succeed. Our first idea is to encourage sparsity in the learned architecture weights by adding an L1 penalty and experimenting with initializations that are less uniform. Another idea is to further tune the architecture learning rate. We have already increased it a full order of magnitude compared to the parameter learning rate, and still find that many weights have not been pushed to 0 or 1 (Fig. 4). We plan to further increase this and test if the architecture training maintains stability. We hope to develop a variant of DARTS that generates architectures that are sparse enough for the argmax heuristic to consistently succeed.

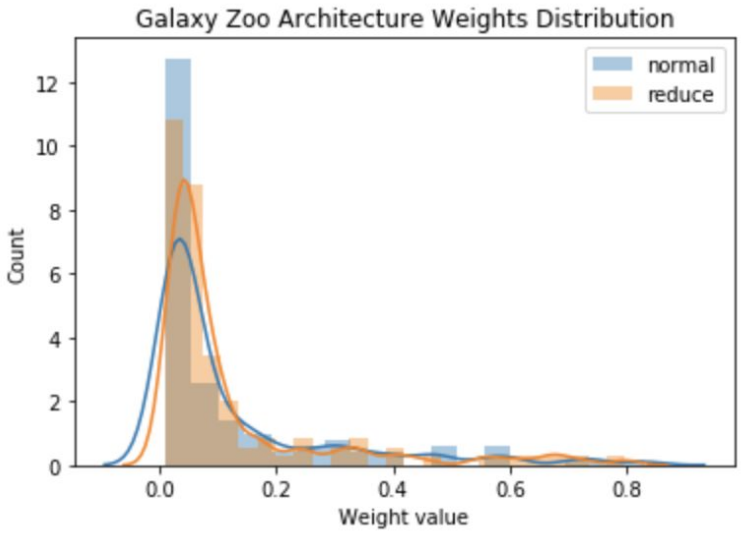

Figure. 4 The learned architecture weights by DARTS on galaxy data.

An intriguing alternative take on this idea is take a sparse model discovered during architecture search; prune all operations with weights very close to zero; and fine tune it. Then we would have a fully trained model in just one step without running a second training run on the discovered architecture. While this might arguably no longer be an “architecture search,” it would be a flexible network design that could become a strong contender to architectures such as ResNet.

Our second goal is to categorize the robustness of DARTS across different variables, including with different random initializations and hyperparameter settings. One of the attractive properties of traditional machine learning approaches to “easy” datasets like MNIST is the wide range of initial configurations that will lead to successful network training. We hope to build up a similar intuition about DARTS: is it robust, with a big sweet spot, or finicky with a narrow band of hyperparameters that work?

Finally, if time permits, we would like to try at least one data set that fits more of the ImageNet paradigm: a large data set where data augmentation is not critical, but the capacity of the model to squeeze information into its parameters is the limiting factor. We are considering some health care data sets with image classification tasks.