***Progressive NAS***

The journey to predict the future or to automate mundane stuff or understand a large amount of data for building strategies has come a long way from statistical techniques to machine learning algorithms to neural nets and deep learning. The potential of these techniques and the corresponding results have motivated new use cases and served humanity to date.

Data Scientists, statisticians, and researchers have developed many techniques and architectures to serve specific purposes, and the architectures have been utilized in numerous other fields (a concept known as Transfer Learning). These hand-designed architectures used for a problem or even for related problems pose a question. Have we done enough? Is this the best we could do? Are there theoretically better models for the given data? Have we searched all or enough search space? Has there been a bias coming from the researcher and his experiences with search space?

Here comes NAS or Neural Architecture Search to the rescue. NAS automates network architecture engineering, learning network topology that can achieve the best possible performance on a certain task.

NAS methods generally have 3 components:

1. Search Space

2. Search algorithm

3. Model evolution

In this article, we would understand some early NAS architectures and implementation of an Amazon reviews text classification using Autokeras library for python.

## **NAS Architectures**

Neural Architecture Search has drawn the huge interest of the research community since 2015 and numerous research papers have been published since then and some in works even now. Here we will discuss some of the popular techniques from Reinforcement Learning based to Genetic Algorithm based to some based on Bayesian Optimisation.

### MetaQNN ([Baker et al. 2017](https://arxiv.org/abs/1611.02167))

An agent is trained to sequentially choose convolutional layers using Q-table ([*Q-learning*](https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html#q-learning-off-policy-td-control)) following a [ϵ-greedy](https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html#%CE%B5-greedy-algorithm) exploration and exploitation strategy. Validation accuracy is chosen as a reward.

here,

s – state

a – action

r – reward

alpha – learning rate

gamma – discount factor (0-immediate reward, 1-future reward)

Q – the value of an action in a particular state

t – current step

*Overview of MetaQNN (Image source:*[*Baker et al. 2017*](https://arxiv.org/abs/1611.02167)*)*

### NEAT (*NeuroEvolution of Augmenting Topologies*)

NEAT uses a [genetic algorithm (GA)](https://en.wikipedia.org/wiki/Genetic_algorithm) to update connection weights and network topology simultaneously. Each gene encodes the complete information about configuring a network, i.e. node weights and edges. The population grows by applying mutation to both the weights and edges, as well as the crossover between two-parent genes.

*NEAT mutations. (Image source: Fig 3 & 4 in*[*Stanley & Miikkulainen, 2002*](http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf)*)*

### PNAS (*Progressive NAS*; [Liu, et al 2018](https://arxiv.org/abs/1712.00559))

As the name suggests PNAS searches increasingly complex models at each step using a Sequential Model-based Bayesian Optimization (SMBO) strategy. PNAS works similar to A\* search, as it searches for models from simple to hard while simultaneously learning a surrogate function to guide the search.

*Progressive NAS algorithm. (Image source:*[*Liu, et al 2018*](https://arxiv.org/abs/1712.00559)*)*

## **Autokeras implementation of NAS**

The Autokeras library uses a similar implementation of NAS using Bayesian Optimization as seen in PNAS above. We will discuss the python implementation and no go into the algorithm as such.

For the demo let's look at the text classification examples using Amazon Review dataset. The dataset contained herewith is a subsample of the complete dataset (3.6M records) and only contains 10K records for simplicity.

Autokeras is the simplest to use when it comes to usage. Please follow along.

**Text

Description automatically generated**

The Autokeras library can be installed using pip.

**Graphical user interface, text, application

Description automatically generated**

Importing and viewing the Amazon dataset, contains 2 columns i.e. text and label.

**Graphical user interface, text, application, email

Description automatically generated** Splitting the data into train and test in a ratio of 4:1

**Graphical user interface, text, application, email

Description automatically generated**

Fit the model: This includes searching for the best architecture based on validation accuracy and then selecting and training the best model architecture.

Predict on the test set.

**Table

Description automatically generated with low confidence**

Compute Precision and Recall on test set predictions.

In this example, we have kept the tuning to the minimal i.e. epochs=2 and max\_trails = 1 for the demo purpose. The true potential of the beast can be unleashed by letting it free!

By not explicitly passing the values for epochs we let the algorithm decide the best number to maximize validation accuracy. The default value for max\_trials is 100 which means the algorithm would look for 100 different model architectures.

The dataset used for this problem has also been used in a similar article on Analytics Vidhya: https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

You can use this article for benchmarking.

The complete Autokeras code is available at the below link:

https://github.com/ankit51/autokeras\_examples/blob/main/Text\_Classification\_using\_Autokeras.ipynb

It would be a pleasure to take up any questions. Thank you for reading!