Biologically Inspired Computing

Coursework

Training an Artificial Neural Network using Particle Swarm Optimisation

Sam Fay-Hunt — `sf52@hw.ac.uk`
Kamil Szymczak — `ks...@hw.ac.uk`

November 22, 2020

Contents

1 Introduction **1**

2 Development Rationale **1**

3 Testing Methodology **1**

4 Results **2**

5 Discussion and Conclusion **3**

A References **4**

1 Introduction

Our solution is written in Python, relying heavily on numpy, pandas and python standard libraries. We also made use of matplotlib for plotting graphs. We used an OOP approach to maintain high organisation and extensibility. We used jupyter notebooks to demonstrate how to use our codebase, and to present our findings.

We have decided to design and implement the ANN using an object orientated approach. We have a class that contains the whole ANN which contains a list of Layer objects, each Layer object holds neurons, activation function it is using, weights and optionally a bias.

Similarly, we have used an OOP approach for PSO implementation where an PSO object holds all the information that includes hyperparameters and global data such as best fitness particle, as well as a list of particles that are objects themselves.

2 Development Rationale

Our rationale was to create 2 submodules: ANNModel and PSO that could be used to build a fully connected neural network or perform PSO independently. We wanted the submodules to be completely decoupled to allow PSO to work on arbitrary Optimisation problems.

ANNModel is only able to create a fully connected neural network...

The ANNModel's design was inspired by TensorFlow & Keras(*Module* 2020; Team 2020), specifically when defining the shape of the neural network. For example you can instantiate the empty network, define the input and result vectors, the layers and then, finally, compile the model. Once compiled you can perform a single pass on the model with either random weights or activations, biases and weights defined by a vector.

The PSO class utilises a Particle class to abstract away some complexity. To use the PSO class define the hyperparameters in the constructor (as described in the documentation), and then specify the fitness function and search dimensions for PSO.

We created an interface for PSO Called Optimisable, any class that properly implements this interface can be used with our PSO implementation. The beauty of this technique is that it allowed us to implement this interface on our PSO class and construct a (PSO) optimiser for our PSO hyperparameters for a specific model shape, for clarity we refer to this outer PSO optimiser as "meta-PSO" and the inner PSO optimiser as "inner-PSO".

This interface also allowed us to create some wrapper classes(PSOHistory, PSOFittest) that can store detailed data about all the hyperparameter settings of the model being optimised.

3 Testing Methodology

As explained in the previous section, we used a method to find the good PSO hyperparameters by applying PSO to another PSO that itself tries to optimize our ANN for a defined dataset. This allowed us to investigate a wide search space of potential optimal hyperparameters for the PSO that was used to optimize the given ANN.

These are the hyperparameters that our meta PSO algorithm searches for `https://www2.macs.hw.ac.uk/~sf52/Bio-Comp-docs/html/_modules/Coursework/PSO/pswarm.html#PSO.dimension_` `vec`

Flaws in testing:

- We use a fixed model structure for our neural network during testing, cannot generalise to other models.
- The fitness function we used was simply 1/loss, this was easy to implement but may have impacted the ability of PSO to escape local maximum. In future a linear fitness value may have been preferred.
- During meta-PSO we only took the mean of 10 inner-PSO runs to evaluate the fitness of each inner-PSO hyperparameter configuration. More would have been better, but impractical in terms of time.

4 Results

Findings in (Garcia-Nieto and Alba 2012; García-Nieto and Alba 2011) indicate that 6 to 8 informants is generally a good number of informants that each particle should have. Our own findings support this.

	Experiment	Data	Fitness	Loss	Score*
Cubic	Best ANN params	Train	4437	0	97%
		Test	6847	0	100%
	10 run mean from best PSO params	Train	47	0.028	12%
		Test	61	0.022	9%
Linear	Best ANN params	Train	3.005e+17	0	100%
		Test	3.380e+17	0	100%
	10 run mean from best PSO params	Train	223	0.033	31%
		Test	219	0.029	34%
Tanh	Best ANN params	Train	236154	0	100%
		Test	74997	0	100%
	10 run mean from best PSO params	Train	49	0.078	26%
		Test	55	0.101	22%
Sine	Best ANN params	Train	487.8	0.002	31%
		Test	459.3	0.002	39%
	10 run mean from best PSO params	Train	12.28	0.083	11%
		Test	12.02	0.086	8%
Complex	Best ANN params	Train	20.17	0.05	9%
		Test	16.05	0.062	18%
	10 run mean from best PSO params	Train	7.83	0.129	12%
		Test	20.96	0.049	17%
XOR	Best ANN params	Train	9096925098444960	0	100%
		Test	16.05	0	100%
	10 run mean from best PSO params	Train	479.64	0.098	86%
		Test	587.04	0.072	91%

5 Discussion and Conclusion

A References

References

Garcia-Nieto, José and Enrique Alba (July 7, 2012). "Why Six Informants Is Optimal in PSO". In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. GECCO '12. New York, NY, USA: Association for Computing Machinery, pp. 25–32. ISBN: 978-1-4503-1177-9. DOI: `10.1145/2330163.2330168`. URL: `https://doi.org/10.1145/2330163.2330168` (visited on 11/21/2020).

García-Nieto, José and Enrique Alba (Jan. 1, 2011). "Empirical Computation of the Quasi-Optimal Number of Informants in Particle Swarm Optimization". In: Genetic and Evolutionary Computation Conference, GECCO'11, pp. 147–154. DOI: `10.1145/2001576.2001597`.

Module (2020). *Module: Tf.Keras.Layers — TensorFlow Core v2.3.0*. URL: `https://www.tensorflow.org/api_docs/python/tf/keras/layers` (visited on 11/22/2020).

Team, Keras (2020). *Keras Documentation: The Model Class*. URL: `https://keras.io/api/models/model/#model-class` (visited on 11/22/2020).