Natural Computing: Assigment 2

By Greg Brimble & Paul Georgiou

Task 1

1.1 Fitness Function

We used the following error function when training the neural network:

$$\epsilon = \frac{\sum_{i=1}^{N} \frac{(\text{output}_i - \text{label}_i)^2}{2}}{N}$$

Where:

- i is the index of a particular data point in the training data set,
- N is the total number of data points in the training data set,
- output_i is the calculated value (between 1 and -1) of the particular data point with index, i, and
- label_i is the expected value (either 1 or -1) of the particular data point with index, i.

Although we simply used this error function as a target to minimize, it could, of course, be converted to a fitness function which one would instead try to maximize, by simply using the following:

$$f = 1 - \epsilon$$

1.2 Search Space

We defined our neural network as having:

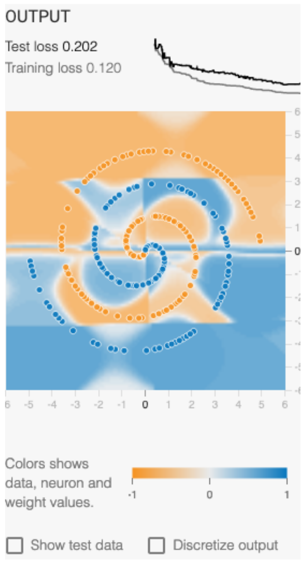
- four inputs:
  - x
  - y
  - $\sin(x)$
  - $\sin(y)$
- a single hidden layer with six nodes,
- and a single output node.

With each having a bias (11), and the final two layers having a combined 30 weights $(1*6+6*4=30)$ this creates a search space of 41 dimensions $(11+30=41)$ for our PSO implementation to solve. Note however, that the biases in the four input nodes are not used in the actual calculation, so 'optimizing' these four dimensions has no effect. The 'true' dimensionality of the problem is therefore 37, despite our implementation optimizing for 41.

1.3 Results

Our adapted Tensorflow Playground (original) can be found here: https://gregbrimble.github.io/playground. The configuration defaults to using PSO to train the neural network, with 25 particles, $\omega = 0.5, \alpha_1 = 2, \text{and} \alpha_2 = 2$.

With this configuration, Figure 1 renders a sample output after 350 iterations.

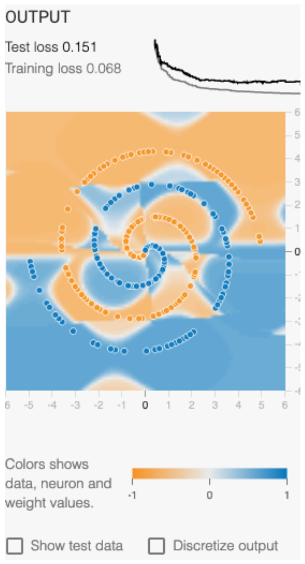

OUTPUT

Test loss 0.202

Training loss 0.120

Figure 1: PSO after 350 iterations

Figure 2 clearly demonstrates that no further significant improvements are made to the model beyond approximately 350 iterations.


OUTPUT

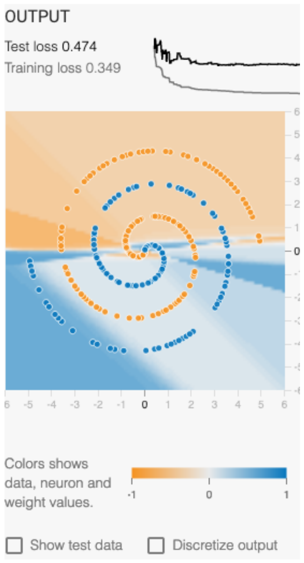
Test loss 0.151

Training loss 0.068

Figure 2: PSO after 1000 iterations

1.4 Comparison against Linear Inputs

Restricting the network to only linear inputs (x and y) significantly curtails the model's ability to fit to the spiral training data. As seen in Figure 3, the minimum training loss is quickly reached, with very little improvement seen with future iterations. The model itself is functionally very simple: practically a 50/50 divide along the spirals' centers.


OUTPUT

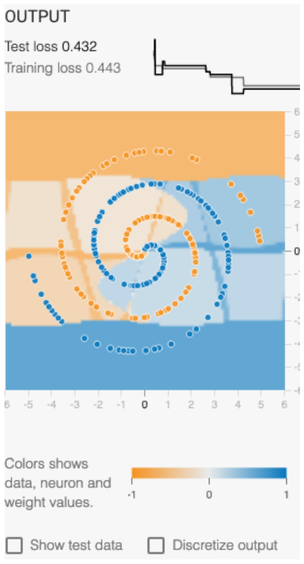
Test loss 0.474

Training loss 0.349

Figure 3: PSO trained with linear features

1.5 Effect of PSO Parameters

Using a higher ω value results in stuttered learning (best seen in the loss sparkline) and a generally poorer model, as demonstrated in Figure 4.

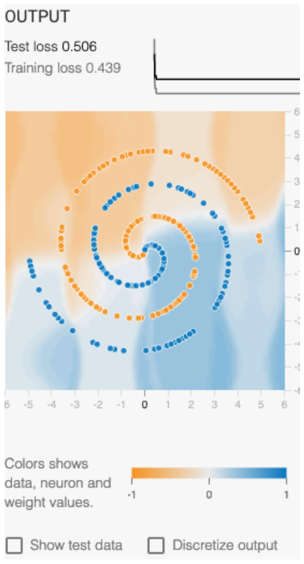

OUTPUT

Test loss 0.432

Training loss 0.443

Figure 4: PSO with a higher $\omega = 0.8$

Significantly changing α_1 and α_2 seriously hampered the ability for the model to train. Figure 5 and 6 demonstrate using $\alpha_1 = \alpha_2 = 3$ and $\alpha_1 = \alpha_2 = 1$ respectively.


OUTPUT

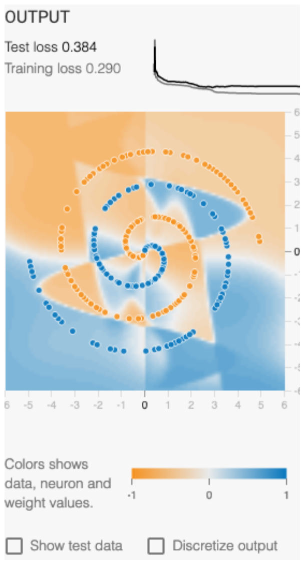
Test loss 0.506

Training loss 0.439

Colors shows data, neuron and weight values.

Show test data   Discretize output

Figure 5: PSO with $\alpha_1 = \alpha_2 = 3$


OUTPUT

Test loss 0.384

Training loss 0.290

Colors shows data, neuron and weight values.

Show test data   Discretize output

Figure 6: PSO with $\alpha_1 = \alpha_2 = 1$

And finally, increasing the number of particles did improve the training and test loss, but begins to show signs of overfitting to the data (Figure 7).

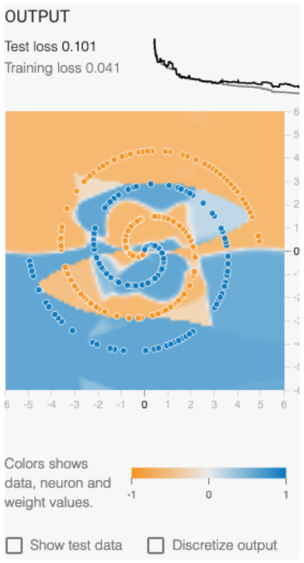


Figure 7: PSO with 100 particles

Task 2

Task 2 was completed in the attached Jupyter Notebook, **not** in the Tensorflow Playground. The neural network itself is trained using a standard gradient-descent and had six inputs: the same four as in the PSO task, as well as x^2 and y^2.

2.1 Evolving the Network Structure

To create a neural network which could be structurally evolved with GP, we firmly set the number of hidden layers to 4. Although this sounds like it would be too prescriptive, this value actually operates as an upper limit, where evolution is free to choose the number of nodes in each layer, and if zero or one, the layer is omitted.

The structure is encoded in a simple list of integers representing the width of each hidden layer at that index. For example, `[2, 3, 1, 4]` has a three hidden layers, with two nodes in the first, three in the second, (the third is omitted), and four in the final.

2.2 Further Evolutions

We decided not to do optimizations on the learning rate or batch size. These were kept at a constant 0.03 and 8, respectively.

2.3 Operators and Parameters of GA and Their Performance

The genetic algorithm uses uniform crossover, with a 90% probability of crossover, and 5% of mutation. We evaluated fitness as follows:

$$f = \frac{1}{1 + \text{MSE}}$$

Where MSE is the mean squared error. In the best performing network we generated, we obtained a mean squared error of 1.719, (fitness of 0.368) and a distinctly spiral-shaped model, as seen in Figure 8.

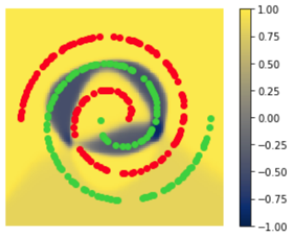


Figure 8: GA

2.4 Controlling Complexity

Upper limits were placed on the number of hidden layers, 4, and of each layers' width, 8. This helped to constrain the network shape, ensuring it didn't become too complex for the problem.

The neural network pictured in Figure 8 had a hidden layer network structure of `[1, 3, 8, 7]`. Since the first layer is only one node, it is actually omitted, giving an actual structure of `[3, 8, 7]`. Inspecting the nodes in that first hidden layer revealed that one node was highly dependant on the nonlinear inputs, and was primarily responsible for drawing the blue ring seen in the model. This was propagated through the rest of the network, while the rest of the nodes' influences were minimized.

Task 3

3.1 Additional Node Functions

3.2 Operators and Parameters of GP and Their Performance

3.3 Comparison with GA

3.4 Comparison with Cartesian Genetic Programs (CGPs)

3.5 Future Work