Course: Natural Computing

*9. Metaheuristics and Neural Networks

J. Michael Herrmann

School of Informatics, University of Edinburgh

michael.herrmann@ed.ac.uk, +44 131 6 517177

History of MHO (20th century)

1809 Jean-Baptiste Lamarck: *Philosophie Zoologique*

1859 Charles Darwin: *On the Origin of Species.*

1930 Ronald Fisher: *The Genetical Theory of Natural Selection*

1953 J. Watson & F. Crick: *Molecular structure of nucleic acids*.

≈1965 Evolution Strategies, Evolutionary Programming

1975 Genetic Algorithms

1983 Simulated Annealing

≈1990 Genetic Programming

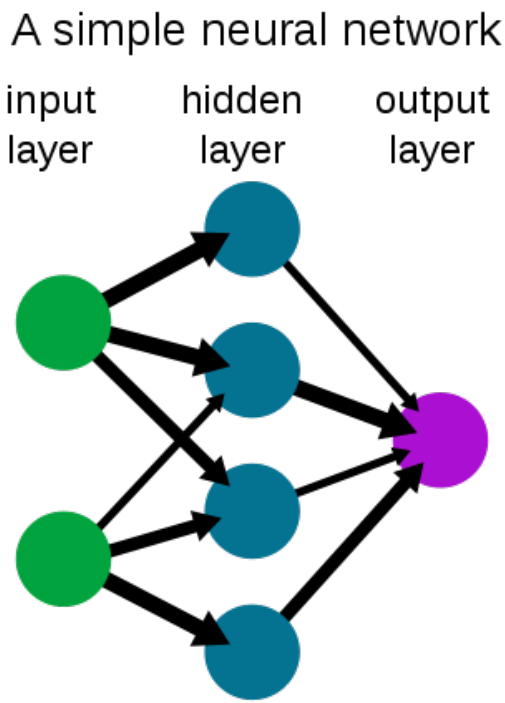
≈1995 ACO (1992), Particle Swarm Optimisation (1995), Differential Evolution (1997)

History of MHO (21st century)

- ≈2000 Hyperheuristics, co-operative co-evolution
- 2000s PSO-like algorithms, benchmarks, multi-objective optimisation, parameter adaptation, topologies, integration, potential applications and actual applications
- 2002 NeuroEvolution of Augmenting Topologies (NEAT)
- 2005 Parallel metaheuristics
- 2009 Matheuristics: Hybridisation with programming
- 2014 Data-driven metaheuristics (continued in 2020s)
- 2017 Learnheuristics: Hybridisation with machine learning
- 2019 Automated algorithm selection and construction
- 2022 Program synthesis

Overview: Metaheuristics and Neural Networks

- Learning vs. search
- Neural combinatorial optimisation
- Deep learning for combinatorial optimisation



MHO and artificial neural networks

Metaheuristic Search

- Biologically-inspired
- Fitness maximisation
- Typically populations of individual searchers
- Random selection/ noise/initialisation
- Random modification operators

Neural Networks

- Brain-inspired
- Minimisation of errors
- Collective processing by neural units
- Random drop-out/ noise/initialisation
- Stochastic gradient (mini-batches)

Literature

Ludwig, S.A. at al (2016) Natural computing algorithms. Springer (Chs. 13, 14, 15)
Iba, H. (2018) Evolutionary approach to machine learning and deep neural networks. Springer.
Iba, H. and Noman, N., 2020. Deep neural evolution. Springer.

Are neural networks MHO? yes, but ...

- Deep neural networks (DNN) are successful on challenging problems where enough relevant data are available
- Neural networks are cooperative, but not population-based
- Artificial neural networks (ANN) in machine learning are not *models* of biological neural networks, i.e. it's a metaphor
- ANN are successful where precise models are unavailable
- ANN require parameter tuning and architecture selection
- ANN use data excellently, yet not efficiently
- Validation and verification of ANN/DNN is difficult, robustness remains a problem
- Large-scale biological neural networks have evolved
- Large-scale biological neural networks may (possibly) implement MHO (G. Edelman's *Neural Darwinism*)

Why are DNN more successful than shallow networks?

- 3-layer neural networks are universal approximators: Weights form a search space with error minimisation as objective
- Shallow network dynamics often trapped by plateaus or local minima
- Optimisation of shallow networks requires stochastic gradients realised by noise or mini-batches

- DNN represent problem structure from local to high-level features, but without making building blocks explicit
- DNN dynamics has many directions to escape from any local optima
- DNN optimisation uses successfully stochastic gradients realised by mini-batches or dropout

DNN: Deep Neural Networks

Learning vs. search: MHO for shallow networks

- Shallow networks use less time and resources than DNN, but training of shallow networks is more difficult
- Weights form a search space with error minimisation as objective
- Search for weights of shallow networks has medium complexity
- Combination of gradients and population-based operators
- A noisy fitness function is provided by batch-based training

Ojha et al. (2017) Metaheuristic design of feedforward neural networks. arXiv.

MHO supporting neural networks learning[1]

- Initialisation of weights and biasses
- Architecture optimisation: Nodes, weight-sharing (convolution), number of layers, widths
- Hyperparameter optimisation (learning rate, regularisation, batch size)
- Learning algorithms and model selection (hyperheuristics of neural network learning)
- Choice of training samples in small-data problems
- Benefits from MHO for Recurrent NN may be larger than for feed-forward NN[2]

[1] X. Yao (1993) A review of evolutionary artificial neural networks," Int. J. Intell. Syst. 8:4, 539-567.
[2] C.-F. Juang (2004) A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Trans. Syst., Man, Cybern. B34:2, 997–1006.

Neuroevolution

Deep neural networks

- can themselves be optimised and analysed by MHO
- can extract and represent complex features of the problem, they may not help simplicity, but can increase the power of the algorithms
- can represent features of a problem class and thus provide a natural implementation of a metaheuristics

Neuroevolution

- Optimisation of neural networks using evolutionary algorithms (Risto Miikkulainen, 1990s)
- (NeuroEvolution of Augmenting Topologies (NEAT, Stanley and Miikkulainen, 2002)
- Related approaches:
  - Evolutionary Acquisition of Neural Topologies (EANT, Kassahun et al., 2005)
  - Hypercube-based NEAT (HyperNEAT, Stanley et al., 2009)
  - Heterogeneous Flexible Neural Trees (HFNT, Ojha et al., 2017)
  - Self-Adaptive NeuroEvolution (SANE, Shuai et al., 2023)

Neuroevolution

- Genotype: MHO
- Encodes network structure
- Modifies network structure by mutation, crossover and selection
- Provides a population of networks

- Phenotype: NN
- Realises network
- Training based on labelled data governed by hyperparameters
- Training error provides fitness information

Schrum, J., Miikkulainen, R. (2014) Evolving multimodal behavior with modular neural networks in Ms. Pac-Man. In: Proc. GECCO, 325–332.

Neuroevolution

- Start with a population of 1-layer NN (passive input layer and processing output layer)
- Inrease complexity
  - inserting new neurons (forming thus new layer)
  - creating a new connections
  - counteracted by pruning
- Representation ambiguity: Subunits [ABC] equivalent to [CBA], to avoid reduces offspring like [ABA], [CBC]:
  - track history of genes (global innovation number)
- Hyper(cube-based)NEAT: Indirect encoding of network components by a Compositional Pattern Producing Network (CPPN): Express ($2D$) patterns by network structure localised in D dimensions (or use GP)

Encoding and modifying neural networks

Encoding

- Connection matrix
- List
- Grammatical
- Cellular

Innovations

- Numbered to simplify the identification (and the diversity) problem

Fitness sharing

- Fitness is divided by measure of structural similarity

Modifying

- Crossover based on innovations avoids meaningless permutations of hidden units
- Mutations:
  - Random change of real valued weight
  - Add (remove) connection/arc with random weight
  - Add (remove) unit (rarely) with unit weight

Results on optimisation of weights

- Early studies with SA[1] or tabu search (TS)[2] showed that performance can be better than back-propagation (BP)
- RBF networks (a.k.a. *extreme learning machines*) tend to be more efficient than both BP and TS
- Hybrid algorithms (e.g. PSO + BP for local search around global best) can be better than either component[3]
- How they compare to second-order methods (e.g. Levenberg-Marquardt or natural gradient) depends on problem
- Research in MOO, ensemble methods, quantum, co-evolution

[1] J. Engel (1988) Teaching feed-forward neural networks by simulated annealing. Complex Syst. **2**:6, 641-648.

[2] R. Battiti & G. Tecchiolli (1995) Training neural nets with the reactive tabu search. IEEE Trans. Neural Netw. **6**:5, 1185-1200.

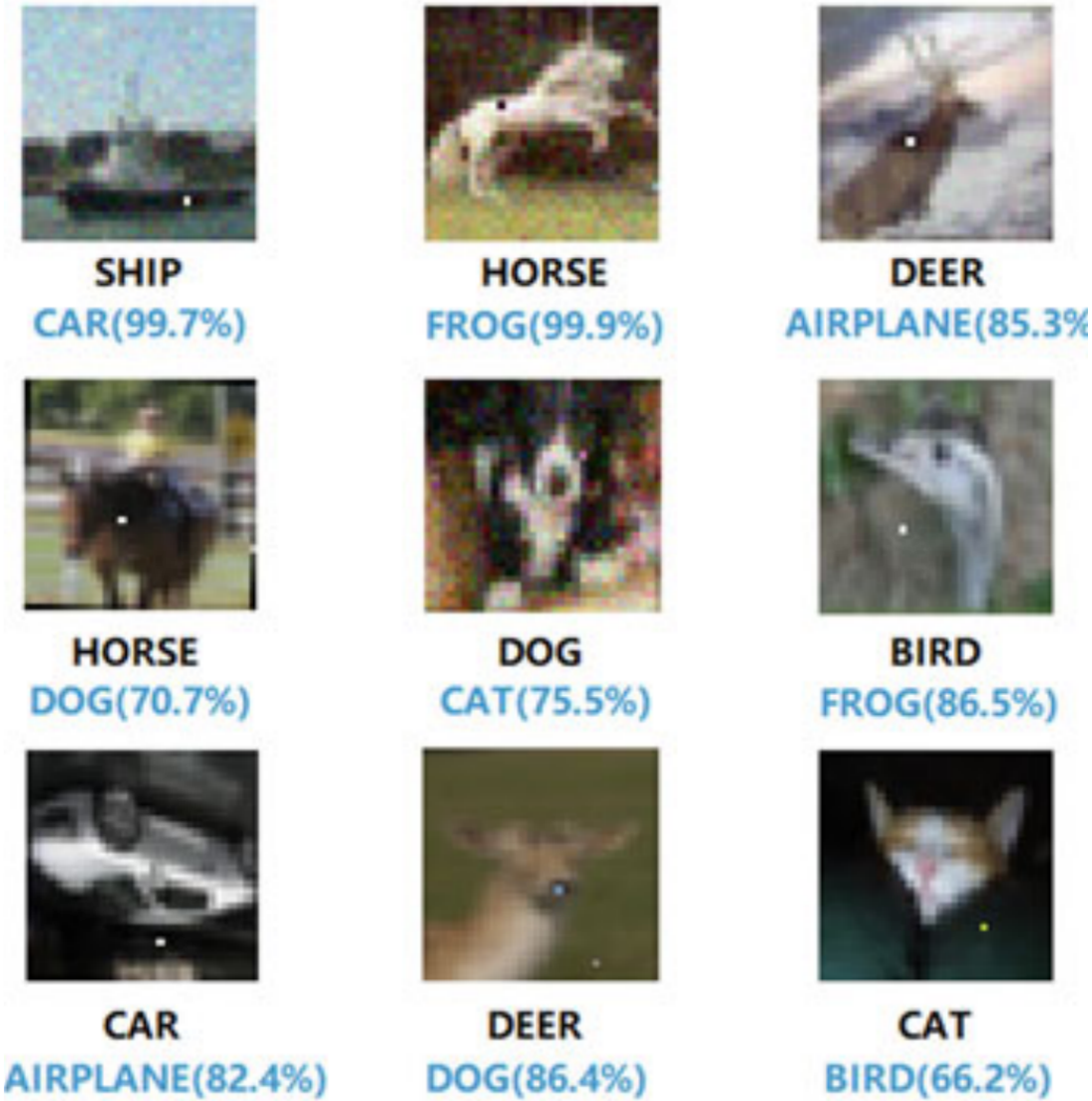
[3] Jing-Ru Zhang et al. (2007) A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. Appl. Math. Comp. **185**, 1026–1037.

More examples

- Aditya Rawal, Jason Liang, and Risto Miikkulainen: Discovering Gated Recurrent Neural Network Architectures.
- Travis Desell, AbdElRahman A. ElSaid, and Alexander G. Ororbia: Investigating Deep Recurrent Connections and Recurrent Memory Cells Using Neuro-Evolution.
- Victor Costa, Nuno Lourenço, João Correia, and Penousal Machado: Neuroevolution of Generative Adversarial Networks.
- Danilo Vasconcellos Vargas: One-Pixel Attack: Understanding and Improving Deep Neural Networks with Evolutionary Computation.

Iba, H. and Noman, N., 2020. Deep neural evolution. Springer.

One-Pixel Attack



Danilo Vasconcellos Vargas: One-Pixel Attack: Understanding and Improving Deep Neural Networks with Evolutionary Computation. *Deep Neural Evolution*, Springer, pp.401-430.

Conclusion on neuro-evolution

- Weight optimisation can work well for
  - complex problems
  - small networks
  - few data
- Neuroevolution is a hyperheuristics with NN as basic algorithm
- Optimisation of neural network structure can be beneficial if evaluation is relatively cheap
- Hybrids and co-evolving systems has been studies, but there is still potential

Neural combinatorial optimization

- Optimise the parameters of the recurrent neural network using a policy gradient method
- TSP
  - Distribution over different city permutations
  - (Negative) tour length as the reward signal (maximisation)
  - near optimal results on 2D Euclidean graphs with ≤ 100 nodes
- For the KnapSack problem optimal solutions for instances with up to 200 items
- Previous work (Vinyals et al., 2015) on TSP: Training of a neural network to predict the sequence of visited cities using supervised signals from an approximate solver

I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio (2017)
Neural combinatorial optimization with reinforcement learning

Neural combinatorial optimization for TSP

- Find permutation π of cities $(i = 1, \ldots, n)$ that minimises tour length

$$L(\pi|s) = \sum_{i=1}^{n-1} \left\| x_{\pi(i)} - x_{\pi(i+1)} \right\| + \left\| x_{\pi(n)} - x_{\pi(1)} \right\|$$

  where $s = \{x_i\}_{i=1,\ldots,N}$ is the set of positions of the cities
- Start with a probabilistic policy that favours short tours and use chain rule

$$p(\pi|s) = \prod_{i=1}^{n} p(\pi(i)|\pi(1, \ldots, i-1), s)$$

I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio (2017)
Neural combinatorial optimization with reinforcement learning

Neural combinatorial optimization

- The neural network receives the city's positions (so far) as inputs and produces the coordinates of the next city as output (Vinyals et al., 2015).
- Once the probabilities are represented one can sample tours and check which one is shortest among the samples
- A batch of B samples is used to improve neural network by a policy gradient method
  - change network parameters θ by gradient descent w.r.t expected tour length. The gradient is estimated by

$$\frac{1}{B}\sum_{k=1}^{B}\left(L\left(\pi_k|s_k\right)-b\left(s_k\right)\right)\nabla_\theta \log p_\theta\left(\pi_k|s_k\right)$$

    which increases probability of paths that are shorter than b given the city positions
  - adapt b to represent average (e.g. exponential moving average)

I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio (2017)
Neural combinatorial optimization with reinforcement learning

Neural combinatorial optimization

- Using a neural network to learn promising directions in search space
- Biased random search with adaptive bias
- 100 cities is not a competitive result, and it does not seem like the method scales up to larger instances
- Although a promising approach, it is not clear whether it is currently better than ACO

I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio (2017)
Neural combinatorial optimization with reinforcement learning

Surrogate models

- Cost of fitness evaluation is usually the limiting factor in real-world applications
- Surrogate models are proxies for the fitness function
- Co-evolution of model and optimiser
  - model guides search ("exploitation")
  - optimiser provides data ("exploration")
- Domain models can implement known biases
- Partial models can represent constraints or one of several finesses in MOO

Can a neural network learn to solve SAT problems?

- NeuroSAT: Train a neural network to predict whether a SAT problem is satisfiable
- After training on random SAT problems, NeuroSAT generalises to
  - larger and more difficult problems than seen during training
  - SAT problems encoding graph colouring, clique detection, dominating set, and vertex cover problems
- If verified and scalable then a truly metaheuristic approach
- So far, result not competitive with state-of-the-art SAT solvers

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, D. L. Dill
Learning a SAT Solver from Single-Bit Supervision (2018)

Can a neural network learn to solve SAT problems?

- NeuroSAT encoding of a formula,
  e.g. $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_5 \vee x_7)$
  - one node per clause $(x_k \vee \neg x_l \vee x_m)$, two nodes per variable (x_i and $\neg x_i$)
  - edges if a literal occurs in clause, i.e. an undirected bipartite graph
  - duplicating literals, duplicating clauses, and clauses with complementary literals need to be removed during preprocessing
- There are various networks:
  - initialisation of both parts
  - counter-propagation (back-and-forth for T times)
  - decision making (voting and averaging)
- Training goal: minimise sigmoid cross-entropy between output and known satisfiability

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, D. L. Dill
Learning a SAT Solver from Single-Bit Supervision (2018)

Can a neural network learn to solve SAT problems?

- Training data are clauses of different length (average 5) with 10 to 40 variables
- Data come in pairs with one satisfiable, one unsatisfiable, but different only in a single negation
- In most run, initially the formula is judged unsatisfiable with low confidence, later (before time T) the decision is taken in a kind of "phase-transition"
- 85% of unseen test examples are classified correctly (70% on satisfiable formulas)
- Although trained only for batches of small samples, also larger problems (e.g. with 120 variables) can be solved (with larger T) at better-than-chance level
- Also tested on formulas that are derived from other problems (i.e. different distribution)

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, D. L. Dill
Learning a SAT Solver from Single-Bit Supervision (2018)

Deep learning for combinatorial optimisation (CO)

- Use a DNN to repeatedly redefine the variation operator
- Certain CO problems can be solved that state-of-the-art algorithms cannot solve
- Certain CO problems can be solved that shallow networks cannot solve
- Certain CO problems can be solved that an end-to-end method cannot solve
- Solution appears polynomial in time (network size may increase exponentially)
- Neural networks may not suffice as a general CO solver, but there is much room for improvement
- At problem sizes of 70 nodes (TSP), results are suboptimal.

J.R. Caldwell, R.A Watson and C. Thies J.D. Knowles (2018) Deep optimisation: Solving combinatorial optimisation problems using deep neural networks

Deep Optimisation Algorithm

Initialise Model
while Optimising Model **do**
  Reset Solution
  while Optimising Solution **do**
    Perform model-informed variation to solution
    Calculate fitness change to solution
    if Deleterious fitness change **then**
      Reject change to solution
    else
      keep change to solution
  Update the model using optimised solution as a training example

J.R. Caldwell, R.A Watson and C. Thies J.D. Knowles (2018) Deep optimisation: Solving combinatorial optimisation problems using deep neural networks

Conclusions on MHO for NN

- DNN/LLM require a huge amount of energy (0.5% of world electrivity consumption, and 1.8% to 3.9% for global tech sector), also data may "run out" (see recent news): Efficiency needs to increase.
- For MHP: Surrogate models (as proxies for the fitness function) can help tp reduce the total cost of fitness evaluations
- Partial models can represent constraints or one of several finesses in MOO
- The *lecture on recent developments in MHO will include more examples (next week)