Population Based Training of Neural Networks (PBT)


(DeepMind, London, UK)
2017
Presentation Structure

• Overview of PBT
• Problem context
• Solution proposed
• Comparison with existing work
• Conclusion and discussion
Overview of PBT

• Algorithm for optimisation of neural network hyperparameters

• Naturally inspired approach based on metal annealing – note PBT is not a genetic algorithm!

• Dynamically assigns computational resources to most promising solutions

• Uses exploration + exploitation mechanism

• Hybridisation of parallel and sequential methods
Problem Context

Parallel optimisation

• E.g. grid search. Allows multiple models to be evaluated with little manual intervention

• Can cover the full solution space reasonably effectively

• Scope for significant speedup IF we assume hyperparameters independent from one another – not always wise…

• Assumes uniform prior over hyperparameters

• Often time exploring poor areas
Problem context
Sequential optimisation

- Example: hand-tuning after each run
- Example: Bayesian optimisation
- We choose (sample) hyperparameters based on a priori assumptions, as well as what we learn from running the network each time – minimises evaluations

\[ p(h|y) = \frac{p(h) \times p(y|h)}{p(y)} \]
Problem context

The “unreasonable success” of random search

- Random search generally much faster than grid search (more likely to modify important parameters – most hyperparameters have little effect on outcome)
- BUT still sampling from a uniform prior
- No ‘honing in’ on optimal solutions
- How to combine computational benefits of parallelisation, but still leverage knowledge gained from each (expensive) solution evaluation?
Solution proposed

Population based training

• Randomly initialise population of candidate solutions
• Evaluate solutions asynchronously for a while (# evaluations? Threshold?)
• When a solution is ‘ready’, use knowledge from population to decide whether to persist, or tack off and try a more promising alternative
• Important: no need for global synchronisation. Just copy more promising solution (+ some noise) and start from there
Solution proposed

Proposed algorithm

Algorithm 1 Population Based Training (PBT)

```
1: procedure TRAIN(P)
2:   for (θ, h, p, t) ∈ P (asynchronously in parallel) do
3:     while not end of training do
4:       θ ← step(θ|h)
5:       p ← eval(θ)
6:       if ready(p, t, P) then
7:         h', θ' ← exploit(h, θ, p, P)
8:         if θ ≠ θ' then
9:           h, θ ← explore(h', θ', P)
10:          p ← eval(θ)
11:       end if
12:     end if
13:   end while
14: return θ with the highest p in P
15: end procedure
```

▷ initial population P
▷ one step of optimisation using hyperparameters h
▷ current model evaluation
▷ use the rest of population to find better solution
▷ produce new hyperparameters h
▷ new model evaluation
▷ update population
▷ update population
Solution proposed

Exploit and explore

• Similar to cloning and mutation genetic operators – but note no recombination

• Typical exploitation: tournament selection, truncation, elitist. May copy entire alternative, or just hyperparameters, omitting model weights

• Explore: can be gradient-based, re-sampling from original prior, or adding random noise/perturbations

• Actual implementation simply applies a multiplier of either 1.2 or 0.8 to hyperparameters (mild perturbations) or 2.0/0.5 (aggressive)
Solution proposed

Output interpretation

- Solutions are not retrained from scratch – model weights are copied over
- So output is not a fixed set of optimal hyperparameters, but actually an adaptive schedule
Solution proposed

Annealing analogy

• PBT more akin to metal annealing than genetic algorithms
• When working metal, grains becomes brittle and needle-like, dislocations (abrupt changes in structure) introduced in stressed positions
• Heating to gentle glow breaks atomic bonds, relaxes molecular structure, dislocations fall away to stress-free positions
• Slow cooling results in gradual recrystallisation, atoms set in place but remain softer, malleable. Slower cooling = better grain growth
• Simulated annealing: aims to replicate this with hyperparameters (or weights...); over time, become less tolerant of poor solutions
Existing work

• Particle swarm optimisation uses knowledge from population, but keeps individuals separate, i.e. no branching

• Simulated annealing, is itself its own optimisation technique

• Obvious parallels with REGAL, but more different than at first glance

• Practical Bayesian optimisation of ML algorithms (Snoek, Lerochelle, Adams, 2012) approach from other direction, starting with a Gaussian Process, then parallelising it
Concluding thoughts

• No constant set of hyperparameters output, so really more ‘model optimisation’ than hyperparameter optimisation. Hyperparameters optimised just like weights, only with lower frequency updates

• Not wholly convinced by the decision to perturb parameters by 1.2/0.8 multiplier rather than adding random noise. (May be to keep better track of annealing schedule, though questionable benefit. As likely for sake of simplicity)
References


- DNA icon made from Icon Fonts (http://www.onlinewebfonts.com/icon) is licensed by CC BY 3.0


- Particle swarm optimisation and simulated annealing gifs obtains under CC0 1.0 https://en.wikipedia.org/wiki/Simulated_annealing https://en.wikipedia.org/wiki/Particle_swarm_optimization