Combining Neural Networks with Meta-heuristics

- Neural Networks (NN)
- Particle Swarm Optimization (PSO)
- Metaheuristics
  - Why bother?
- Optimizing NN using PSO
  - Training
  - Hyperparameter selection
  - Model structure

Neural Networks

- Very quick refresher

A bioinspired network of neurons used for predictive modeling. Data fed through network from input layer (IL) to output layer (OL), passing one or many hidden layers (HL) in the process.

Neural networks using feed-forward and back-propagation are typical.

Networks train by feeding data forward through the network, calculating the error as a function of an expected output against the actual output, this error is passed backwards through the network and the synapse weights are updated as a function of the error, synapse weight, and network parameters (learning rate, momentum, etc).

Neural Networks (cont'd)

```
procedure nn(training examples) is:
  initialize network structure
  initialize random weights for each synapse
  for each epoch do:
    for each training example d do:
      feed-forward d through layers 0..n
      find error e as a function of expected vs actual output
      back-propagate e through layers n..0
      update weights w as a function of w, e, network parameters
    endfor
  endfor
endprocedure
```

Neural Networks (cont'd)

- Error / loss

Error is a function of what a neuron outputs (which itself is a function of the summing and activation functions) against what is expected. An example might be the squared error:

```
procedure error(expected, actual) is:
  sum := 0
  for i in expected do:
    sum += (expected[i] - actual[i])^2
  endfor
  return sum
endprocedure
```

Minimizing the error is the goal of neural network: a minimal error suggests better classification performance. You could say that *optimizing* this is the goal of a neural network. Also consider the mean squared error (MSE) of a network.

Particle Swarm Optimization

- Very quick refresher

An optimization algorithm which iteratively improves candidate solutions to a problem by mimicking behavior of birds.

Particles are placed into the search space at random, and using a heuristic function, will update their positions based on particle attributes, swarm parameters, and PSO parameters:

- Particle attributes
  - Position
  - Velocity
  - Personal best fitness
- Swarm parameters
  - Swarm best position
  - Swarm best fitness
- PSO parameters
  - Inertial weight (measure of resistance to change in velocity)
  - Cognitive coefficient (global exploration, local exploitation)
  - Social coefficient (global exploitation, local exploration)

Goal is to find an extremum of the heuristic function.

Particle Swarm Optimization (cont'd)

```
procedure pso() is:
  for each epoch do:
    for each particle p in swarm s do:
      for each dimension d in search space do:
        wei := ω * p.velocity
        cog := c1 * r1 * (p.bestPosition[d] - p.position[d])
        soc := c2 * r2 * (s.bestPosition[d] - p.position[d])
        newVelocity[d] := wei + cog + soc
        newPosition[d] := p.position[d] + newVelocity[d]
      endfor
      p.velocity := newVelocity
      p.position := newPosition
    endfor
    for each particle p in swarm s do:
      if heuristic(p.position) < p.fitness do:
        p.fitness := heuristic(p.position)
      endif
    endfor
    update swarm, particle bests
  endfor
endprocedure
```

Metaheuristics

- A higher-level procedure designed to find an heuristic for an optimization problem.
  - Guides the search process to find optimal solution
  - Typically non-deterministic
  - Metaheuristics are problem agnostic, generalized, and have many applications
- Recall as neural network becomes trained, MSE is reduced.
- i.e. optimization problem with a continuous and known search space.
- Can add PSO, GA, ACO, ABC, GWO, BA, SA, etc to neural networks as a metaheuristic.
  - Can train network
  - Can determine optimal network parameters (e.g. LR, MR)
  - Can determine optimal model structure (e.g. number of hidden neurons)

Metaheuristics (cont'd)

- BP in neural networks is inefficient in several regards:
  - Vanishing or exploding gradients
  - Overfitting or underfitting
  - Historically slow, but improvements over last ten-ish years
  - Relies on differential calculus
- BP was one of the first methods to train FF network.
- Humans are creatures of habit, thus BP became the "go to" approach.
- Rather than use BP to train network, why not use a metaheuristic like PSO to train the network?
  - No vanishing/exploding gradients (no gradient at all, in fact)
  - *Can* be fast
  - Doesn't require differential calculus
  - Doesn't require the problem to be differentiable

Metaheuristics (cont'd)

- Problems inherent with BP already mentioned, but what about PSO?
  - More parameters to worry about
  - Is non-deterministic
  - Lacks a mathematical foundation, i.e. is not particularly robust
  - Empirically found to be inefficient for search spaces with gargantuan dimensionality
    - Tradeoff between size of problem vs. speed of solution
    - Very large network = large number of synapse weights = use BP
  - While BP can get stuck in local extrema, PSO can fail to find global extrema entirely
- Despite this, there is merit in adapting PSO for NN.
  - Using BI algorithm to train BI algorithm in lieu of calculus has a nice ring to it
  - Small problems show favorable results

Optimizing NN using PSO

- Many approaches.
  - Use PSO to train the NN
    - Eschews BP in favor of swarm intelligence
  - Use PSO to find hyperparameters for the NN
    - Removes guesswork in finding optimal NN parameters
    - Although you exchange LR, MR with PSO params
  - Use PSO to find optimal model structure
    - # of HL and HL size is innate to specific problem
    - A model which works well for one problem doesn't mean it works for all problems
    - Can be found through trial and error, or have another algorithm handle it
- Of primary import is training using PSO as much of the speed (or lack thereof) of BP lies within the need of an activation derivative.
  - Consider exponential (e^x) calculation is expensive; logistic sigmoid and hyperbolic tangent derivatives rely on performing many of these calculations
  - e.g. for 7000 training examples in a 64-25-10 network, using derivative of logistic sigmoid function for BP means 22.4million exponential calculations per epoch: PSO does zero

Optimizing NN using PSO (cont'd)

- Training using PSO

PSO takes a population of candidate solutions (particles) and changes their positions in the search space to become more/most fit. To adapt for neural network training…

Particles ← network synapse weights

e.g. 10-5-3 network contains (10 * 5) + (5 * 3) = 65 weights (excl. biases).

Therefore the search space has 65 dimensions. Each particle position represented as a 65-tuple with a velocity vector of the same size.

Fitness Function ← MSE

MSE is not necessarily your only option here. Consider cross-entropy, classification error, etc.

Optimizing NN using PSO (cont'd)

From this, the PSO algorithm has a search space and fitness function and can proceed as normal. As PSO is problem agnostic, no major changes are needed to the algorithm in adapting for NN with the exception that the best fit particle is returned to initialize network weights.

As PSO is non-deterministic, it is not assured to find the minimum of the MSE function, as it is still susceptible to the pitfalls of PSO. As well, PSO is parameterized just as a BP network would be.

Optimizing NN using PSO (cont'd)

```
procedure nn-pso(training examples) is:
  initialize network structure
  initialize swarm of particles at random
  for each epoch do:
    for each particle p in swarm s do:
      for each dimension d in search space do:
        wei := ω * p.velocity
        cog := c1 * r1 * (p.bestPosition[d] - p.position[d])
        soc := c2 * r2 * (s.bestPosition[d] - p.position[d])
        newVelocity[d] := wei + cog + soc
        newPosition[d] := p.position[d] + newVelocity[d]
      endfor
      p.velocity := newVelocity
      p.position := newPosition
    endfor
    for each particle p in swarm s do:
      if MSE(training examples, p.position) < p.fitness do:
        p.fitness := MSE(training examples, p.position)
      endif
    endfor
    update swarm, particle bests
  endfor
  initialize network weights as best particle position
endprocedure
```

```
procedure MSE(training examples, position) is:
  sum := 0
  outputs, targets := Empty
  for each training example t do:
    outputs.add(feedforward(t))
    targets.add(class for t)
  endfor
  for each element i in outputs do:
    sum += (outputs[i]-targets[i])^2
  endfor
  return sum / size(training examples)
endprocedure
```

NB: this just switches BP with PSO.

Optimizing NN using PSO (cont'd)

Training a network is just an optimization problem: finding a solution *a* such that MSE(*a*) ≤ MSE(*b*) for all *b* in the search space is essentially what a neural network does, with the search space being within the continuous boundaries of weight possibilities.

While the algorithm seems complex initially, it *can* be computationally cheaper than BP with activation derivative.

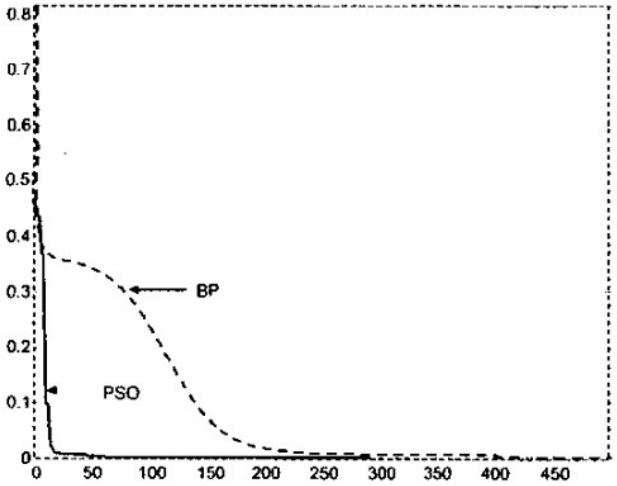
Why *can* it be and not *it is*? Think of each particle as its own network structure (as it is a set of weights): for 500 training examples and 10 particles, this is PSO being "trained" 5,000 times per epoch.

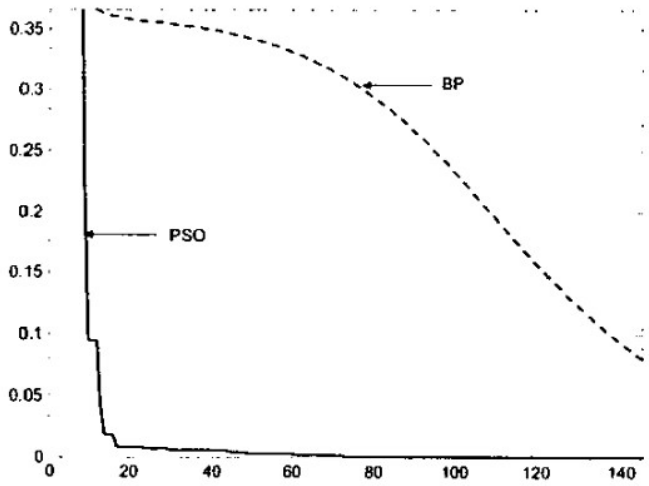
Compare to online BP using the same problem space as needing only 500 per epoch. Conclusion? PSO-NN is suited for smaller problem sizes.

The largest benefit is backpropagation is more expensive on a per epoch level; however, closing the gap is the problem size.

Optimizing NN using PSO (cont'd)

Gudise, et al. examined NN-BP vs NN-PSO in training against trivial non-linear function $y = 2x^2 + 1$. Using random uniform initial weight initialization for BP and PSO, the MSE was found for both using optimal parameters:





Termination conditions were MSE ≤ 0.01.

Optimizing NN using PSO (cont'd)

Also compared were number of arithmetic operations (additions and multiplications) and epochs to reach termination conditions.

- PSO parameters:
  - $\omega = 0.8$
  - $c_1 = c_2 = 2.0$
  - Swarm size = 25
- BP parameters:
  - Variable LR, MR (scheduling)
- Network parameters:
  - 2-4-1 structure
  - Otherwise standard FF network

	PSO	BP
Epochs	194	9,836
Operations	3.07m	18.80m

Optimizing NN using PSO (cont'd)

Khan, et al. compared various metaheuristics against BP, including PSO, using less trivial problem spaces in health classification:

Data Set	Model Structure	Examples	NN-BP Operations	NN-BP MSE	NN-PSO Operations	NN-PSO MSE
Cancer	9-h-2	699	261	3.24e-4	643	4.21e-5
Diabetes	8-h-2	768	20,675	2.21e-4	1,697	1.80e-5
Heart	35-h-2	920	150,948	5.32e-5	2,502	1.94e-5
Thyroid	21-h-3	7,200	1,460,835	1.10e-5	56,400	2.42e-5

While the benefit of NN-PSO against NN-BP is problem specific and this is but a small sample size of experiments, academic papers suggest NN-PSO converges quicker but does not always find global optima; however, almost uniformly NN-PSO performs less operations = faster. Performance (minimizing MSE) is problem specific but close results.

On average for situations where NN-PSO outperformed NN-BP, the computational cost is 1/22 in favor of NN-PSO. In every case experimented here, however, NN-PSO had an edge in either minimizing MSE and/or speed: in no case was NN-BP ideal for both.

Optimizing NN using PSO (cont'd)

Gudise, Khan et al. concluded using metaheuristics like PSO to train NN shows promise but dependent on problem space and scope as well as careful parameter selection.

Generally PSO is cheaper than BP but still has drawbacks of PSO including failure to meet optima. Improvement strategies for PSO can also be applied to mitigate to some degree.

BP was slow in its infancy but is constantly improving, esp. with GPU acceleration coming into vogue. Usefulness of metaheuristics to train may have uncertain future.

Further Reading

Using PSO to train NN in lieu of BP is not the only application. PSO can be used in tandem with BP to find optimal hyperparameters or model structure.

Arguments for NN-PSO's speed become irrelevant as PSO's addition to the FF-BP network is only to create better conditions for use.

Use of PSO in RNN's interesting: determining where synapses should exist when expert decision on structure impossible or impractical. Also useful for ELM.

References

Garro, Beatriz A & Sossa, Humberto & Vázquez, Roberto. (2011). Back-propagation vs particle swarm optimization algorithm: Which algorithm is better to adjust the synaptic weights of a feed-forward ANN?. International Journal of Artificial Intelligence. 7. 208-218.

Gudise, V.G. & Venayagamoorthy, Ganesh. (2003). Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks. Proceedings of the 2003 IEEE Svarm Intelligence Symposium. 110 - 117. 10.1109/SIS.2003.1202255.

Zajmi, Leke & Ahmed, Falah.Y.H.Ahmed & Jaharadak, Adam Amril. (2018). Concepts, Methods, and Performances of Particle Swarm Optimization, Backpropagation, and Neural Networks. Applied Computational Intelligence and Soft Computing. 2018. 1-7. 10.1155/2018/9547212.

Khan, Koffka & Ashok, Sahai. (2012). A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context. International Journal of Intelligent Systems and Applications. 4. 10.5815/ijisa.2012.07.03.

Garro, Beatriz A & Vázquez, Roberto. (2015). Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms. Computational Intelligence and Neuroscience. 2015. 10.1155/2015/369298.