Particle Swarm Optimization Submission for WMT16 Tuning Task

Viktor Kocur
CTU in Prague
FNSPE
kocurvik@fjfi.cvut.cz

Ondřej Bojar
Charles University in Prague
MFF ÚFAL
bojar@ufal.mff.cuni.cz

Abstract

This paper describes our submission to the Tuning Task of WMT16. We replace the grid search implemented as part of standard minimum-error rate training (MERT) in the Moses toolkit with a search based on particle swarm optimization (PSO). An older variant of PSO has been previously successfully applied and we now test it in optimizing the Tuning Task model for English-to-Czech translation. We also adapt the method in some aspects to allow for even easier parallelization of the search.

1 Introduction

Common models of statistical machine translation (SMT) consist of multiple features which assign probabilities or scores to possible translations. These are then combined in a weighted sum to determine the best translation given by the model. Tuning within SMT refers to the process of finding the optimal weights for these features on a given tuning set. This paper describes our submission to WMT16 Tuning Task[1], a shared task where all the SMT model components and the tuning set are given and task participants are expected to provide only the weight settings. We took part only in English-to-Czech system tuning.

Our solution is based on the standard tuning method of Minimum Error-Rate Training (MERT, Och, 2003). The MERT algorithm described in Bertoldi et al. (2009) is the default tuning method in the Moses SMT toolkit (Koehn et al., 2007). The inner loop of the algorithm performs optimization on a space of weight vectors with a given translation metric[2]. The standard optimization is a variant of grid search and in our work, we replace it with the Particle Swarm Optimization (PSO, Eberhart et al., 1995) algorithm.

Particle Swarm Optimization is a good candidate for an efficient implementation of the inner loop of MERT due to the nature of the optimization space. The so-called Traditional PSO (TPSO) has already been tested by Suzuki et al. (2011), with a success. Improved versions of the PSO algorithm, known as Standard PSO (SPSO), have been summarized in Clerc (2012).

In this paper, we test a modified version of the latest SPSO2011 algorithm within the Moses toolkit and compare its results and computational costs with the standard Moses implementation of MERT.

2 MERT

The basic goal of MERT is to find optimal weights for various numerical features of an SMT system. The weights are considered optimal if they minimize an automated error metric which compares the machine translation to a human translation for a certain tuning (development) set.

Formally, each feature provides a score (sometimes a probability) that a given sentence e in goal language is the translation of the foreign sentence f. Given a weight for each such feature, it is possible to combine the scores to a single figure and find the highest scoring translation. The best translation can then be obtained by the following formula:

$$e^* = \underset{e}{\operatorname{argmax}} \sum_i \lambda_i \log\left(p_i(e|f)\right) = g_p(\lambda) \quad (1)$$

[1] http://www.statmt.org/wmt16/tuning-task/

[2] All our experiments optimize the default BLEU but other metrics could be directly tested as well.

Proceedings of the First Conference on Machine Translation, Volume 2: Shared Task Papers, pages 518–524,
Berlin, Germany, August 11-12, 2016.

The process of finding the best translation e^* is called decoding. The translations can vary significantly based on the values of the weights, therefore it is necessary to find the weights that would give the best result. This is achieved by minimizing the error of the machine translation against the human translation:

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\, err_f(g_p(\lambda), e_{human}) \qquad (2)$$

The error function can also be considered as a negative value of an automated scorer. The problem with this straight-forward approach is that decoding is computationally expensive. To reduce this cost, the decoder is not run for every considered weight setting. Instead, only some promising settings are tested in a loop (called the "outer loop"): given the current best weights, the decoder is asked to produce n best translation for each sentence of the tuning set. This enlarged set of candidates allows us to estimate translation scores for similar weight settings. An optimizer uses these estimates to propose a new vector of weights and the decoder then tests this proposal in another outer loop. The outer loop is stopped when no new weight setting is proposed by the optimizer or no new translations are found by the decoder. The run of the optimizer is called the "inner loop", although it need not be iterative in any sense. The optimizer tries to find the best weights so that the least erroneous translations appear as high as possible in the n-best lists of candidate translations.

Our algorithm replaces the inner loop of MERT. It is therefore important to describe the properties of the inner loop optimization task.

Due to finite number of translations accumulated in the n-best lists (across sentences as well as outer loop iterations), the error function changes only when the change in weights leads to a change in the order of the n-best list. This is represented by numerous plateaus in the error function with discontinuities on the edges of the plateaus. This prevents the use of simple gradient methods. We can define a local optimum not in a strict mathematical sense but as a plateau which has only higher or only lower plateaus at the edges. These local optima can then be numerous within the search space and trap any optimizing algorithm, thus preventing convergence to the global optimum which is desired.

Another problem is the relatively high dimensionality of the search space. The Tuning Task model has 21 features but adding sparse features, we can get to thousands of dimensions.

These properties of the search space make PSO an interesting candidate for the inner loop algorithm. PSO is stochastic so it doesn't require smoothness of the optimized function. It is also highly parallelizable and gains more power with more CPUs available, which is welcome since the optimization itself is quite expensive. The simplicity of PSO also leaves space for various improvements.

3 PSO Algorithm

The PSO algorithm was first described by Eberhart et al. (1995). PSO is an iterative optimization method inspired by the behavior of groups of animals such as flocks of birds or schools of fish. The space is searched by individual particles with their own positions and velocities. The particles can inform others of their current and previous positions and their properties.

3.1 TPSO

The original algorithm is defined quite generally. Let us formally introduce the procedure. The search space S is defined as

$$S = \bigotimes_{d=1}^{D} [min_d, max_d] \qquad (3)$$

where D is the dimension of the space and min_d and max_d are the minimal and maximal values for the d-th coordinate. We try to find a point in the space which maximizes a given function $f : S \mapsto \mathbb{R}$.

There are p particles and the i-th particle in the n-th iteration has the following D-dimensional vectors: position $\mathbf{x}_i^n$, velocity $\mathbf{v}_i^n$, and two vectors of maxima found so far: the best position $\mathbf{p}_i^n$ visited by the particle itself and the best known position $\mathbf{l}_i^n$ that the particle has learned about from others.

In TPSO algorithm, the $\mathbf{l}_i^n$ vector is always the globally best position visited by any particle so far.

The TPSO algorithm starts with simple initialization:

$$\mathbf{x}_i^0 = rand(S) \tag{4}$$

$$\mathbf{v}_i^0 = \frac{rand(S) - \mathbf{x}_i^0}{2} \tag{5}$$

$$\mathbf{p}_i^0 = \mathbf{x}_i^0 \tag{6}$$

$$\mathbf{l}_i^0 = \underset{j}{\operatorname{argmax}}\, f(\mathbf{p}_j^0) \tag{7}$$

where the function $rand(S)$ generates a random vector from space S with uniform distribution.

The velocity for the next iteration is updated as follows:

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + U(0,1)\phi_p\mathbf{p}_i^t + U(0,1)\phi_l\mathbf{l}_i^t \tag{8}$$

where $U(0,1)$ denotes a random number between 0 and 1 with uniform distribution. The parameters w, ϕ_p, $\phi_l \in (0,1)$ are set by the user and indicate a slowdown, and the respective weight for own vs. learned optimum.

All the following vectors are then updated:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{9}$$

$$\mathbf{p}_i^{t+1} = \mathbf{x}_i^{t+1} \quad \text{if } f(\mathbf{x}_i^{t+1}) > f(\mathbf{p}_i^t) \tag{10}$$

$$\mathbf{l}_i^{t+1} = \underset{j}{\operatorname{argmax}}(f(\mathbf{p}_j^{t+1})) \tag{11}$$

The process continues with the next iteration until all of the particles converge to proximity of a certain point. Other stopping criteria are also used.

3.2 Modified SPSO2011

We introduce a number of changes to the algorithm SPSO2011 described by Clerc (2012).

In SPSO2011 the global best position $\mathbf{l}_i^t$ is replaced by the best position the particle has received information about from other particles. In the original SPSO2011 this is done in a synchronized fashion: after every iteration, all particles send their best personal positions to m other particles. Every particle chooses the best position it has received in the current iteration and sets its $\mathbf{l}_i^t$ accordingly. This generalization of $\mathbf{l}_i^t$ is introduced in order to combat premature convergence to a local optimum.

To avoid waiting until all particles finish their computation, we introduce per-particle memory of "learned best positions" called the "neighbourhood set" (although its members do not have to be located in any close vicinity). This set of best positions is limited to k elements, each new addition over the limit k replaces the oldest information. To establish the "global" optimum $\mathbf{l}_i^t$, every particle consults only its set of learned best positions.

The algorithm starts with the initialization of particle vectors given by the equations (4-6). The $\mathbf{l}_i^0$ is initialized with the value of $\mathbf{p}_i^0$. The sets of learned best positions are initialized as empty.

Two constants affect computations given below: w is again the slowdown and c controls the "expansion" of examined neighbourhood of each particle. We set w and c to values that (as per Bonyadi and Michalewicz, 2014) ensure convergence:

$$w = \frac{1}{2ln(2)} \approx 0.721 \tag{12}$$

$$c = \frac{1}{2} + ln(2) \approx 1.193 \tag{13}$$

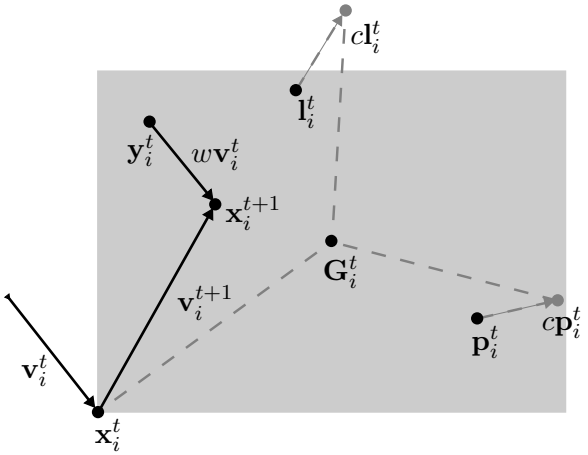

Figure 1: Construction of the particle position update. The grey area indicates $P(\mathbf{G}, \mathbf{x})$.

For the update of velocity, it is first necessary to calculate a "center of gravity" $\mathbf{G}_i^t$ of three points: the current position $\mathbf{x}_i^t$, a slightly "expanded" current best position $\mathbf{p}_i^t$ and a slightly expanded best position known by colleagues $\mathbf{l}_i^t$. The "expansion" of the positions is controlled by c and directed outwards from $\mathbf{x}_i^t$:

$$\mathbf{G}_i^t = \mathbf{x}_i^t + c \cdot \frac{\mathbf{p}_i^t + \mathbf{l}_i^t - 2\mathbf{x}_i^t}{3} \tag{14}$$

To introduce further randomness, $\mathbf{x}_i^t$ is relocated to a position $\mathbf{y}_i^t$ sampled from the uniform distribution in the area $P(\mathbf{G}_i^t, \mathbf{x}_i^t)$ formally defined as:

$$P(\mathbf{G}, \mathbf{x}) = \bigotimes_{d=1}^{D} \Big[G_d - |G_d - x_d|, G_d + |G_d - x_d| \Big] \tag{15}$$

Our $P(\mathbf{G}, \mathbf{x})$ is a hypercube centered in $\mathbf{G_i^t}$ and touching $\mathbf{x}_i^t$, see Figure 1 for an illustration. The original SPSO2011 used a d-dimensional ball with the center in $\mathbf{G}$ and radius $\|\mathbf{G} - \mathbf{x}\|$ to avoid the bias of searching towards points on axes. We are less concerned about this and opt for a simpler and faster calculation.

The new velocity is set to include the previous velocity (reduced by w) as well as the speedup caused by the random relocation:

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + \mathbf{y}_i^t - \mathbf{x}_i^t \tag{16}$$

Finally, the particle position is updated:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + \mathbf{y}_i^t \tag{17}$$

The optimized function is evaluated at the new position $\mathbf{x}_i^{t+1}$ and the particle's best position is updated if a new optimum was found. In any case, the best position $\mathbf{p}_i^{t+1}$ together with its value is sent to m randomly selected particles (possibly including the current particle) to be included in their sets of learned best positions as described above. The particle then sets its $\mathbf{l}_i^{t+1}$ to best position from its own list of learned positions.

The next iteration continues with the updated vectors. Normally, the algorithm would terminate when all particles converge to a close proximity to each other, but it turns out that this often leads to premature stopping. There are many other approaches possible to this problem (Xinchao, 2010; Evers and Ben Ghalia, 2009), but we choose a simple restarting strategy: when the particle is sending out its new best position and value to m fellows, the manager responsible for this checks if this value was not reported in the previous call (from any other particle). If it was, then the current particle is instructed to restart itself by setting all of its vectors to random initial state.[3] The neighborhood set is left unchanged. The restart prevents multiple particles exploring the same area.

The drawback of restarts is that the stopping criterion is never met. In our first version, we ran the algorithm for a fixed number of position updates, specifically 32000. Later, we changed the algorithm to terminate after the manager has seen 3200 position updates without any update of the global best position. In the following section, we refer to the former as PSO without the termination condition (PSO) and the latter as PSO with the termination condition (PSO-T).

Properties of SPSO2011 have been investigated by Bonyadi and Michalewicz (2014). We use a slightly different algorithm, but our modifications should have an effect only on rotational invariance, which is not so much relevant for our purpose. Aside from the discussion on the values of w and c with respect to the convergence of all particles to the same point, Bonyadi and Michalewicz also mention that SPSO2011 is not guaranteed to converge to a local optimum. Since our search space is discontinuous with plateaus, the local convergence in the mathematical sense is not especially useful anyway.

4 Implementation

We implemented the algorithm described above with one parameter, the number of particles. We set the size of the neighborhood set, denoted k above, to 4 and the number of random particles receiving the information about a particle's best position so far (m) to 3.

The implementation of our version of the PSO algorithm is built within the standard Moses code. The algorithm itself creates a reasonable parallel structure with each thread representing a single particle.

We use similar object structure as the baseline MERT implementation. The points are represented by their own class which handles basic arithmetic and stream operations. The class carries not only the vector of the current position but also its associated score.

Multiple threads are maintained by the standard Moses thread pools (Haddow, 2012). Every thread ("Task" in Moses thread pools) corresponds to a particle and is responsible for calculating its search in the space using the class `PSOOptimizer`. There are no synchronous iterations, each particle proceeds at its own pace.

All optimizers have access to a global manager object of class `PSOManager`, see Figure 2 for an illustration. The manager provides methods for the optimizers to get the best vector $\mathbf{l}_i^t$ from the

[3] The use of score and not position is possible due to the nature of the space in which a same score of two points very likely means that the points are equivalent.

Run	PSO-16	PSO-64	PSO-T-16	PSO-T-64	MERT-16
1	14.5474	15.6897	15.6133	15.6613	14.5470
2	17.3292	18.7340	18.7437	18.4464	18.8704
3	18.9261	18.9788	18.9711	18.9069	19.0625
4	19.0926	19.2060	19.0646	19.0785	19.0623
5	19.1599	19.2140	19.0968	19.0738	19.1992
6	19.2444	19.2319	-	19.0772	19.1751
7	19.2470	19.2383	-	-	19.0480
8	19.2613	19.2245	-	-	19.1359
12	-	-	-	-	19.1625

Table 1: The final best BLEU score after the runs of the inner loop for PSO without and with the termination condition with 16 and 64 threads respectively and standard Moses MERT implementation with 16 threads.

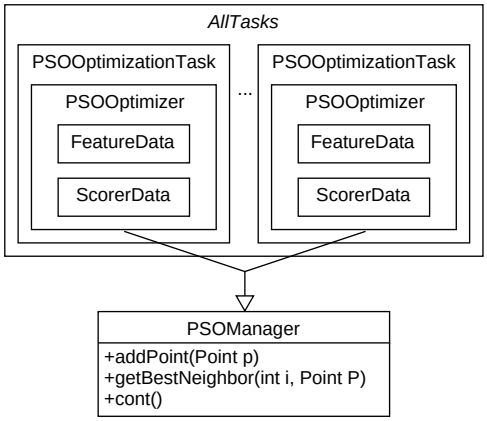

Figure 2: Base structure of our PSO algorithm

neighborhood set, to report its best position to the random m particles (`addPoint`) and to check if the optimization should still run (`cont`) or terminate. The method `addPoint` serves two other purposes: incrementing an internal counter of iterations and indicating through its return value whether the reporting particle should restart itself.

Every optimizer has its own `FeatureData` and `ScorerData`, which are used to determine the score of the investigated points. As of now, the data is loaded serially, so the more threads we have, the longer the initialization takes. In the baseline implementation of MERT, all the threads share the scoring data. This means that the data is loaded only once, but due to some unexpected locking, the baseline implementation never gains speedups higher than 1.5, even with 32 threads, see Table 2 below.

This structure allows an efficient use of multiple cores. Methods of the manager are fast compared to the calculations performed in the optimizers. The only locking occurs when threads are trying to add points; read access to the manager can be concurrent.

5 Results

We ran the tuning only for the English to Czech part of the tuning task. We filtered and binarized the model supplied by the organizers to achieve better performance and smaller memory costs.

For the computation, we used the services of Metacentrum VO. Due to the relatively high memory demands we used two SGI UV 2000 machines: one with 48x 6-core Intel Xeon E5-4617 2.9GHz and 6TB RAM and one with 48x 8-core Intel Xeon E5-4627v2 3.30GHz and 6TB RAM. We ran the tuning process on 16 and 64 CPUs, i.e. with 16 and 64 particles, respectively. We submitted the weights from the 16-CPU run. We also ran a test run using the standard Moses MERT implementation with 16 threads for a comparison.

Table 1 shows the best BLEU scores at the end of each inner loop (as projected from the n-best lists on the tuning set of sentences). Both methods provide similar results. Since the methods are stochastic, different runs will lead to different best positions (and different scores).

Comparison of our implementation with with the baseline MERT on a test set is not necessary. Both implementations try to maximize BLEU score, therefore any overtraining occurring in the baseline MERT occurs also in our implementation and vice versa.

Table 2 shows the average run times and reached scores for 8 runs of the baseline MERT and our PSO and PSO-T, starting with the same

Outer Loop	CPUs	Wall Clock [s] MERT	PSO	PSO-T	Projected BLEU Reached MERT	PSO	PSO-T
1	1	186.24±10.63	397.28±2.13	62.37±19.64	14.50±0.03	13.90±0.05	13.84±0.05
1	4	123.51±3.58	72.75±1.12	21.94±4.63	14.51±0.03	14.48±0.08	14.46±0.06
1	8	135.40±8.43	43.07±0.78	15.62±3.40	14.52±0.04	14.53±0.05	14.42±0.12
1	16	139.43±8.00	33.00±1.37	14.59±2.21	14.53±0.02	14.51±0.08	14.48±0.10
1	24	119.69±4.43	32.20±1.62	16.89±3.16	14.52±0.02	14.55±0.06	14.47±0.07
1	32	119.04±4.47	33.42±2.16	19.16±2.92	14.53±0.03	14.50±0.04	14.50±0.07
3	1	701.18±47.13	1062.38±1.88	117.64±0.47	18.93±0.04	18.08±0.00	18.08±0.00
3	4	373.69±28.37	189.86±0.64	57.28±23.61	18.90±0.00	18.82±0.12	18.81±0.07
3	8	430.88±24.82	111.50±0.53	37.92±8.68	18.95±0.05	18.89±0.09	18.87±0.06
3	16	462.77±18.78	80.54±5.39	29.62±4.34	18.94±0.04	18.94±0.07	18.90±0.05
3	24	392.66±13.39	74.08±3.64	31.67±3.47	18.94±0.04	18.93±0.05	18.86±0.05
3	32	399.93±27.68	82.83±3.82	37.70±4.52	18.91±0.01	18.90±0.05	18.87±0.06

Table 2: Average run times and reached scores. The ± are standard deviations.

n-best lists as accumulated in iteration 1 and 3 of the outer loop. Note that PSO and PSO-T use only as many particles as there are threads, so running them with just one thread leads to a degraded performace in terms of BLEU. With 4 or 8 threads, the three methods are on par in terms of tuning-set BLEU. Starting from 4 threads, both PSO and PSO-T terminate faster than the baseline MERT implementation. Moreover the baseline MERT proved unable to utilize multiple CPUs efficiently, whereas PSO gives us up to 14-fold speedup.

In general, the higher the ratio of the serial data loading to the search computation time, the worse the speedup. The search in PSO-T takes much shorter time so the overhead of serial data loading is more apparent and PSO-T seems parallelized badly and gives only quadruple speedup. The reduction of this overhead is highly desirable.

6 Conclusion

We presented our submission to the WMT16 Tuning Task, a variant of particle swarm optimization applied to minimum error-rate training in statistical machine translation. Our method is a drop-in replacement of the standard Moses MERT and has the benefit of easy parallelization. Preliminary experiments suggest that it indeed runs faster and delivers comparable weight settings.

The effects on the number of iterations of the MERT outer loop as well as on the test-set performance have still to be investigated.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 645452 (QT21). This work has been using language resources stored and distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2015071). Computational resources were supplied by the Ministry of Education, Youth and Sports of the Czech Republic under the Projects CESNET (Project No. LM2015042) and CERIT-Scientific Cloud (Project No. LM2015085) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

References

Nicola Bertoldi, Barry Haddow, and Jean-Baptiste Fouet. 2009. Improved minimum error rate training in moses. *The Prague Bulletin of Mathematical Linguistics* 91:7–16.

Mohammad Reza Bonyadi and Zbigniew Michalewicz. 2014. Spso 2011: Analysis of stability; local convergence; and rotation sensitivity. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, pages 9–16.

Maurice Clerc. 2012. Standard particle swarm optimisation .

Russ C Eberhart, James Kennedy, et al. 1995. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*. New York, NY, volume 1, pages 39–43.

George I Evers and Mounir Ben Ghalia. 2009. Regrouping particle swarm optimization: a new global optimization algorithm with improved performance consistency across benchmarks. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, pages 3901–3908.

Barry Haddow. 2012. Adding Multi-Threaded Decoding to Moses. *Prague Bulletin of Mathematical Linguistics* 93:57–66.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*. Association for Computational Linguistics, pages 177–180.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics, pages 160–167.

Jun Suzuki, Kevin Duh, and Masaaki Nagata. 2011. Distributed minimum error rate training of smt using particle swarm optimization. In *IJCNLP*. pages 649–657.

Zhao Xinchao. 2010. A perturbed particle swarm algorithm for numerical optimization. *Applied Soft Computing* 10(1):119–124.