Master thesis project

Probabilistic Approach to Adaptive Computational Time in Neural Networks

May 2017

Abstract

This master thesis project intends to develop a model that learns a task dependent optimization algorithm and approximates a posterior halting distribution over time steps by means of optimizing a variational lower bound on the log-likelihood. We hope to show that our ACT optimizer uses on average less computational steps towards an optimum than the baseline LSTM optimizer from [1] and that the posterior halting distribution can be exploited to dynamically determine the optimization step when computation should stop.

Summary of the project

The master thesis project briefly described here is based on the recent surge of interest in using neural networks to learn optimization procedures, applying a range of innovative meta-learning techniques ([5], [1], [2]). In particular our work takes the model described in [1] as a baseline (referred to LSTM[1] optimizer hereafter). The approach taken in this study focuses on learning how to utilize gradient observations over time, of a function f to be optimized, in order to achieve fast learning of the underlying model. The authors show that such a model outperforms w.r.t. convergence speed and final minimum, hand-designed optimizers like ADAM, RMSprop or SGD on tasks like simple quadratic function optimization or training of a neural network for image classification. The loss function used to train the LSTM optimizer considers the sum of the separate function evaluations $f(\mathbf{x}_t)$ at query points $\mathbf{x}_{1:T}$ for a certain time horizon T i.e. $L(\phi, \mathbf{x}) = \mathbb{E}_{p(f)}\left[\sum_{t=1}^{T} f(\mathbf{x}_t)\right]$, where ϕ denotes the parameters of the LSTM optimizer and t refers to the discrete time steps. As will be outlined below, our idea is to endow the model with additional information (aka evidence) that can be interpreted as a kind of *weighting* or *crediting* of the individual time step contributions to the final outcome of the iterative optimization algorithm. This information should be exploited by the model and hence reduce the average number of computational steps needed to solve a specific optimization task.

Another lead of this project comes from the recent work of [3], [7], [4] and [6] in which the authors pursue different approaches to induce a machine learning model with the ability to dynamically adjust the computational time needed to solve a specific task. We are particularly interested in the research of Alex Graves [3] that outlines an approach of *Adaptive Computational Time* (ACT) applied to Recurrent Neural Networks (RNNs) that learn how many computational steps to take between receiving an input and emitting an output.

The work of [3] is important because it makes a significant contribution towards gradient-based approaches for learning the number of computational steps in a neural computation graph. The RNN learns a so called *halting distribution* by augmenting the network output with sigmoidal halting units. In addition the objective function is extended with a so called *ponder cost* which acts as a (time) penalty. The stopping policy of the model is based on the cumulative probability of the halting distribution. Although there is some controversy whether the ponder cost is truly differentiable the experimental results especially on the four supervised, synthetic tasks reveal that the approach works. The ACT approach is well suited to be combined with an iterative optimization approach but we will pursue a probabilistic approach to ACT in our research.

Hence this master thesis project combines the ideas of [1] and [3]. We take the LSTM optimizer from [1] as a baseline model and develop the following extensions: (1) a loss function that encourages the model to use less iterative steps than the baseline LSTM optimizer; (2) an approximation of a discrete posterior distribution over time steps t that specifies the probability that computation stops at step t. The posterior distribution will be exploited to dynamically determine the optimization step when computation should stop.

[1] Long Short Term Memory module

In order to train our model (denoted *ACT optimizer* hereafter) we optimize a variational *lower bound* on the log-likelihood

$$\log p(\mathbf{x}) \geq \mathcal{L}(\phi, \theta, \mathbf{x}) = \sum_{t=0}^{\infty} q_\phi(t|\mathbf{x}) \Big(\log p_\theta(\mathbf{x}|t) - \log \frac{q_\phi(t|\mathbf{x})}{p(t)} \Big) \ , \tag{1}$$

where $\mathbf{x}$ denotes an observed, continuous random variable (e.g. the target values of a regression function), t is a discrete latent random variable that represents the time steps and θ respectively ϕ are the parameters of the model and the RNN inference network. The project drew inspiration from the work of [8] in which the authors develop an algorithm to translate the problem of solving a Markov Decision Problem (MDP) into a problem of likelihood maximization. From their work we extracted the idea of decomposing an infinite-horizon MDP problem into a mixture of finite-time MDPs which is a useful concept in optimization procedures where the number of necessary optimization steps (i.e. horizon T) is not known in advance.

More specific we decompose $q(t|\mathbf{x})$ into

$$q(t|\mathbf{x}) = \sum_{T=0}^{\infty} q(t|\mathbf{x}, T) p(T) \ , \tag{2}$$

and substitute 2 into 1 which results after some algebraic operations in the following formalization of the lower bound

$$\log p(\mathbf{x}) \geq \sum_{T=0}^{\infty} p(T) \sum_{t=0}^{T} q_\phi(t|\mathbf{x}, T) \Big(\log p_\theta(\mathbf{x}|t, T) - \log \frac{q_\phi(t|\mathbf{x}, T)}{p(t|T)} \Big) \ . \tag{3}$$

A more detailed outline of the pursued approach can be found in section 4 of the following document.

The performance of the LSTM and ACT optimizers will be compared by means of the following experiments: (1) optimization of 2D quadratic functions; (2) optimization of simple regression functions; (3) optimization of more complex (regression) functions[2]; (4) optimization of a simple neural network trained on MNIST dataset.

Planning

1. February was used for literature study;
2. March was used to start a proof of concept in which both models were evaluated on 2d quadratic function optimization;
3. April, evaluate proof of concept, write introduction, related work & approach section for final paper;
4. May, run experiments for generalized linear regression functions, record results, prepare new experiments with more complex (regression) functions;
5. June, 01–06–2017 to 18–06–2017 vacation, run new experiments on more complex (regression) functions, record results;
6. July & August run experiments in which the models train a simple neural network (details to be decided);
7. September, write on final paper, wrap up last results, may be re-do short experiment;
8. October, intend to graduate.

References

[1] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and de Freitas, N. Learning to learn by gradient descent by gradient descent. *arXiv:1606.04474 [cs]* (June 2016). arXiv: 1606.04474.

[2] Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., and de Freitas, N. Learning to Learn for Global Optimization of Black Box Functions. *arXiv:1611.03824 [cs, stat]* (Nov. 2016). arXiv: 1611.03824.

[2] details to be decided

[3] Graves, A. Adaptive Computation Time for Recurrent Neural Networks. *arXiv:1603.08983 [cs]* (Mar. 2016). arXiv: 1603.08983.

[4] Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. Variable Computation in Recurrent Neural Networks. *arXiv:1611.06188 [cs, stat]* (Nov. 2016). arXiv: 1611.06188.

[5] Li, K., and Malik, J. Learning to Optimize. *arXiv:1606.01885 [cs, math, stat]* (June 2016). arXiv: 1606.01885.

[6] Li, Z., Yang, Y., Liu, X., Wen, S., and Xu, W. Dynamic Computational Time for Visual Attention. *arXiv:1703.10332 [cs]* (Mar. 2017). arXiv: 1703.10332.

[7] Odena, A., Lawson, D., and Olah, C. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780* (2017).

[8] Toussaint, M., and Storkey, A. Probabilistic Inference for Solving Discrete and Continuous State Markov Decision Processes. In *Proceedings of the 23rd International Conference on Machine Learning* (New York, NY, USA, 2006), ICML '06, ACM, pp. 945–952.