CONDITIONAL COMPUTATION IN NEURAL NETWORKS FOR FASTER MODELS

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau & Doina Precup
School of Computer Science
McGill University
Montreal, Canada
`{ebengi,pbacon,jpineau,dprecup}@cs.mcgill.ca`

ABSTRACT

Deep learning has become the state-of-art tool in many applications, but the evaluation of expressive deep models can be unfeasible on resource-constrained devices. The conditional computation approach has been proposed to tackle this problem (Bengio et al., 2013; Davis & Arel, 2013). It operates by selectively activating only parts of the network at a time. We propose to use reinforcement learning as a tool to optimize conditional computation policies. More specifically, we cast the problem of learning activation-dependent policies for dropping out blocks of units as reinforcement learning. We propose a learning scheme motivated by computation speed, capturing the idea of wanting to have parsimonious activations while maintaining prediction accuracy. We apply a policy gradient algorithm for learning policies that optimize this loss function and propose a regularization mechanism that encourages diversification of the dropout policy. We present encouraging empirical results showing that this approach improves the speed of computation without impacting the quality of the approximation.

1 MOTIVATION

Recent approaches (Bengio et al., 2013; Davis & Arel, 2013) have proposed the use of *conditional computation* in order to speed up deep network calculations. Conditional computation refers to activating only some of the units in a network, in an input-dependent fashion. As an intuitive example, if we think we're looking at a car, we only need to compute the activations of the vehicle detecting units, not of all features that a network could possible compute. The immediate effect of activating fewer units is that propagating information through the network can be faster, both at training time and at test time. However, one needs to be able to decide intelligently which units to turn on and off, depending on the input data. This is typically achieved with some form of gating structure, learned in parallel with the original network.

Conditional computation can naturally be viewed as a sequential decision making problem. At each layer, given information on the current state of the computation (what has been computed so far), one has to decide which units to activate. There is also a natural, delayed reward measure, given by the error at the output of the network; one can further impose rewards that encourage sparsity in the computation, or a fast computation time. This view makes this problem a natural fit for reinforcement learning algorithms.

We explore the formulation of conditional computation using reinforcement learning (Sutton & Barto, 1998). We propose to learn input-dependent activation probabilities for every node (or blocks of nodes), while trying to jointly minimize the prediction errors at the output and the number of participating nodes at every layer, thus reducing the computational load. One can also think of our method as being related to standard dropout (Hinton et al., 2012), which has been used as a tool to both regularize and speed up the computation. However, we emphasize that dropout is in fact a form of "unconditional" computation, in which the computation paths are data-independent. Therefore, usual dropout is less likely to lead to specialized computation paths within a network.

We present the problem formulation, and our solution to the proposed optimization problem, using policy search methods (Deisenroth et al., 2013). Preliminary results are included for standard classification benchmarks.

2 MODEL

Our model consists in a fully-connected neural network, joined with stochastic per-layer policies that activate or deactivate equally-sized blocks of nodes of the neural network in an input-dependent manner, both at train *and* test time. We learn per-layer l stochastic policies, where the state space is the layer's input and the action space is to activate or turn off each of the layer's blocks of units:

$$\pi^{(l)}(\mathbf{u}\,|\,\mathbf{s}) = \prod_{i=1}^{k} \sigma_i^{u_i}(1-\sigma_i)^{(1-u_i)} \qquad \sigma_i = [\text{sigm}(\mathbf{Z}^{(l)}\mathbf{s} + \mathbf{d}^{(l)})]_i \tag{1}$$

σ_i represents the participation probability of a block, which is sampled independently for each example at each forward pass in the model.

We learn these policies using REINFORCE (Williams, 1992), which is a policy gradient method. In the minibatch setting, given the vector of costs $C(\mathbf{x})$ and the probability $\pi(.|.)$ of each action, it gives us the gradient with respect to our parameters:

$$\nabla_{\theta_l}\mathcal{L} \approx \frac{1}{m_b}\sum_{i=1}^{m_b} C(\mathbf{x}_i)\nabla_{\theta_l}\log\pi^{(l)}(\mathbf{u}_i^{(l)}\,|\,\mathbf{s}_i^{(l)}) = \frac{1}{m_b}\mathbf{c}^\top\nabla_{\theta_l}\log\pi^{(l)}(\mathbf{U}^{(l)}\,|\,\mathbf{S}^{(l)}) \tag{2}$$

where $\theta_l = \{\mathbf{Z}^{(l)}, \mathbf{d}^{(l)}\}$. Note that the vector-Jacobian product can be computed efficiently via so-called Hessian-free methods (Pearlmutter, 1994; Martens, 2010).

2.1 REGULARIZATION

In order to guide policy search and force it to learn sparse policies (so that we gain in terms of computation), we added several regularization terms, which are minimized using gradient descent (and not REINFORCE).

$$L_b = \sum_j^n \|\mathbb{E}\{\sigma_j\} - \tau\|_2 \qquad L_e = \mathbb{E}\{\|(\frac{1}{n}\sum_j^n \sigma_j) - \tau\|_2\} \qquad L_v = -\sum_j^n \text{var}_i\{\sigma_{ij}\}$$

L_b and L_e force each block to be activated with probability τ, respectively across a minibatch, and with respect to the other blocks. L_v encourages policies not to be uniform policies and to be more input-dependent.

3 EXPERIMENTS

The proposed model was implemented within Theano (Bergstra et al., 2010). In addition to using optimizations offered by Theano, we also implemented specialized matrix multiplication code that skips deactivated blocks thus reducing computational load. We found speedups of up to 5-10x on CPU and 2-4x on GPU using naive implementations.[1]

We first experimented on **MNIST** (LeCun et al., 1998) as a sanity check. We found that our policy gradient method would learn to activate certain blocks depending on the class of the input. Unfortunately models required to correctly classify MNIST digits are too small to benefit from our approach in terms of computation speed.

We then turned to the **CIFAR-10** (Krizhevsky & Hinton, 2009) and Street View House Numbers (**SVHN**) (Netzer et al., 2011) datasets. We find that on both these datasets, it is possible to train models that achieve similar accuracy with significantly reduced runtime, as illustrated in figure 1.

We also find that by varying the weight of our regularization terms (2.1) we can reliably control the trade-off in our model between runtime and accuracy. By making policies more sparse, we can gain in computation time at the expense of losing some accuracy. This is illustrated in figure 2.

[1]Implementations used in this paper are available at `http://github.com/bengioe/condnet/`

dataset	model	test error	τ	#blocks	block size	test time
CIFAR-10	condnet	**.497**	1/16	10,10	64	2.0s (5.3×)
CIFAR-10	bdNN	.629	0.17	10,10	64	1.93s (5.3×)
CIFAR-10	NN	**.497**	-	480,480	1	8.34s
SVHN	condnet	**.073**	1/22	25,22	32	10.2s(1.4×)
SVHN	NN	.091	-	1280,1056	1	16.8s

Figure 1: condnet: our approach, NN: Neural Network without the conditional activations, bdNN, block dropout Neural Network using a uniform policy. *test time* is the time required to do a full pass over the test dataset using the implementation, on a **CPU**, running on a **single core**; in parenthesis is the speedup factor compared to runtime without a specialized implementation.

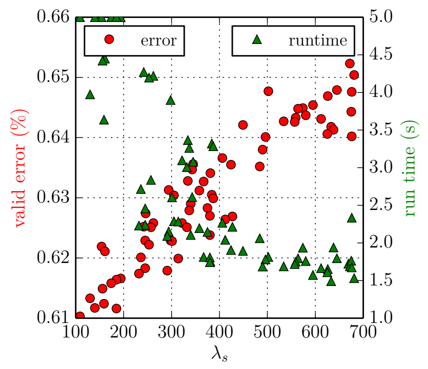

Figure 2: CIFAR-10, each pair of circle and triangle is an experiment made with a given lambda (x axis), resulting in a model with a certain error and running time (y axes). As λ_s increases the running time decreases, but so does performance.

4 Conclusion

This paper presents a method for tackling the problem of conditional computation in deep networks by using reinforcement learning. We propose a type of parameterized conditional computation policy that maps the activations of a layer to a Bernoulli mask. The reinforcement signal accounts for the loss function of the network in its prediction task, while the policy network itself is regularized to account for the desire to have sparse computations. The REINFORCE algorithm is used to train policies to optimize this cost function.

Our experiments show that it is possible to train such models at the same levels of accuracy as their standard counterparts. Additionally, it seems possible to execute these similarly accurate models faster due to their sparsity. Furthermore, the model has a few simple parameters that allow to explicitly control the trade-off between accuracy and running time.

While the speedup that we obtain is appreciable, it seems that larger speedups could be achieved. Other model parameterizations or domain specific parametrizations such as convolutional neural networks could yield significantly faster speedups. Our experiments show that using Reinforcement Learning to address the problem of conditional computation is a sensible approach, and that a next step would be to find models that computationally take advantage of such sparse computations.

Acknowledgments

The authors gratefully acknowledge financial support for this work by the Samsung Advanced Institute of Technology (SAIT), the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds de recherche du Québec - Nature et Technologies (FQRNT).

References

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*, 2013.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013. doi: 10.1561/2300000021. URL http://dx.doi.org/10.1561/2300000021.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 735–742, 2010. URL http://www.icml2010.org/papers/458.pdf.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5. Granada, Spain, 2011.

Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Comput.*, 6(1):147–160, January 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.1.147. URL http://dx.doi.org/10.1162/neco.1994.6.1.147.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL http://dx.doi.org/10.1007/BF00992696.