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 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:

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

(1)

$$\sigma_i$$ 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 \(C(x)\) and the probability \(\pi(.|.)\) of each action, it gives us the gradient with respect to our parameters:

$$\nabla_{\theta_l} \mathcal{L} \approx \frac{1}{mb} \sum_{i=1}^{mb} C(x_i) \nabla_{\theta_l} \log \pi^{(l)}(u^{(l)}_i | s^{(l)}_i) = \frac{1}{mb} c^T \nabla_{\theta_l} \log \pi^{(l)}(U^{(l)} | S^{(l)})$$  

(2)

where \(\theta_l = \{Z^{(l)}, d^{(l)}\}\). 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).

$$L_b = \sum_j ||\mathbb{E}\{\sigma_j\} - \tau||_2 \quad L_c = \mathbb{E}\{||(\frac{1}{n} \sum_j \sigma_j) - \tau||_2\} \quad L_v = - \sum_j \text{var}_j(\sigma_{ij})$$

\(L_b\) and \(L_c\) force each block to be activated with probability \(\tau\), 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/
### 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


