Conditional computation in neural networks using a decision-theoretic approach

Pierre-Luc Bacon
School of Computer Science
McGill University
Montreal, Canada
pbacon@cs.mcgill.ca

Emmanuel Bengio
School of Computer Science
McGill University
Montreal, Canada
bengioe@gmail.com

Joelle Pineau
School of Computer Science
McGill University
Montreal, Canada
jpineau@cs.mcgill.ca

Doina Precup
School of Computer Science
McGill University
Montreal, Canada
dprecup@cs.mcgill.ca

Abstract

Deep learning has become the state-of-art tool in many applications, but the evaluation and training of such models is very time-consuming and expensive. Dropout has been used in order to make the computations sparse (by not involving all units), as well as to regularize the models. In typical dropout, nodes are dropped uniformly at random. Our goal is to use reinforcement learning in order to design better, more informed dropout policies, which are data-dependent. We cast the problem of learning activation-dependent dropout policies as a reinforcement learning problem. We propose a reward function motivated by information theory, which captures the idea of wanting to have parsimonious activations while maintaining prediction accuracy. We develop policy gradient algorithms for learning policies that optimize this loss function and present encouraging empirical results showing that this approach improves the speed of computation without significantly impacting the quality of the approximation.

Keywords: Neural Networks, Conditional Computing, REINFORCE

Acknowledgements

Funding for this work was provided in part by NSERC and FQRNT

1 Motivation

Large-scale neural networks, and in particular deep learning architectures, have seen a surge in popularity in recent years, due to their impressive empirical performance in complex supervised learning tasks, including state-of-the-art performance in image and speech recognition (He et al. 2015). Yet the task of training such networks remains a challenging optimization problem. Several related problems arise: very long training time (several weeks on modern computers, for some problems), potential for over-fitting (whereby the learned function is too specific to the training data and generalizes poorly to unseen data), and more technically, the vanishing gradient (Hochreiter et al. 2001) problem (whereby the gradient information gets increasingly diffuse as it propagates from layer to layer).

A technique called *dropout* was introduced by Hinton et al. 2012 as a way to reduce overfitting by breaking the tendency for "co-adaptations" between nodes. Dropout is a simple modification to standard backpropagation, whereby each node in the hidden layers can be skipped over (or "dropped") from a given round of training with $1/2$ probability. At test time, the input examples are rescaled by $1/2$ to compensate for the fact that twice as many units are active. Dropout can be interpreted to perform a form of regularization to prevent overfitting (Wager, Wang, and Liang 2013), and is related to the notion of "robustness" in statistics. It has now become standard in most convolutional neural network architectures.

The motivation for the current work is to build on the idea of dropout to address the vanishing gradient problem in deep networks. As opposed to dropout, we learn input-dependent dropout probabilities for every node while trying to jointly minimize the prediction errors at the output and the number of participating nodes at every layer. Highly focused computation paths in the network have the advantage of both reducing the computational load with sparser matrix operations, while concentrating the back-propagated errors over fewer nodes. We present the problem formulation, and our solution to the proposed optimization problem, using policy search methods. Preliminary results are included for standard classification benchmarks.

2 Problem formulation

We cast the problem of learning the input-dependent dropout probabilities at each layer in the framework of Markov Decision Processes (MDP) (Puterman 1994). We define a discrete time, finite state and action MDP $\langle \mathcal{S}, \mathcal{A}, P\left\{\cdot \,|\, s, u\right\}, c\rangle$ with c the cost function and $P\left\{\cdot \,|\, s, u\right\}$ the distribution over the next state given that action u is taken in state s. An action $\mathbf{u} \in \{0,1\}^k$ in this model consists in the application of a dropout mask over the units of a given layer. We define the state space of the MDP over the vector-valued activations of all nodes at the previous layer.

As in the original dropout paper (Hinton et al. 2012), each node in a given layer has an associated Bernoulli distribution, which determines its probability of being dropped. We will train a policy which allows the parameter of the distribution to be different for each node of a layer, and adapt to a given input. We define the policy as an n-dimensional Bernoulli distribution:

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

where the σ_i denotes the *participation* probability (opposite of the dropout probability), to be computed from the activations s of the layer below. We denote the sigmoid function by sigm, the weight matrix by $\mathbf{Z}$, and the bias vector by $\mathbf{o}$. The sigmoid-Bernoulli policy can be viewed as introducing an additional layer with the same number of nodes as in the associated hidden layer.

3 Learning sigmoid-Bernoulli policies

We use the likelihood-ratio method (Williams 1992) to learn the parameters of the sigmoid-Bernoulli policies. Since the dimensionality of the observation space can change at each decision step, we learn L disjoint policies (one for each layer of the deep network). As a consequence, the summation in the REINFORCE gradient disappears and becomes:

$$\nabla_{\theta_i} \mathcal{L}(\theta) = \mathbb{E}\left\{(C(\tau) - b)\nabla_{\theta_i} \log \pi(\mathbf{u}_i \,|\, \mathbf{s}_i)\right\} \tag{2}$$

since θ_i only appears in the ith decision stage and the gradient is zero otherwise.

Estimating (2) from samples requires propagating through many instances at a time, which we achieve through mini-batches of size m_b . This approach has the advantage of making optimal use of the fast matrix-matrix capabilities of GPU hardware. Under the mini-batch setting, $\mathbf{s}_i$ becomes a matrix and $\pi(\cdot, \,|\, \cdot)$ a vector of dimension m_b . Taking the gradient of the parameters with respect to the log action probabilities can then be seen as forming a Jacobian. We can thus re-write the empirical average in matrix form:

$$\nabla_{\theta_i} \mathcal{L}(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} (C(\mathbf{x}_n) - b)\nabla_{\theta_i} \log \pi(\mathbf{u}_i \,|\, \mathbf{s}_i) = \frac{1}{N} \mathbf{c}^\top \nabla_{\theta_i} \log \pi(\mathbf{U}_i \,|\, \mathbf{S}_i) \tag{3}$$

where $C(\mathbf{x}_n)$ is the total cost for input $\mathbf{x}_n$ and N is the number of examples in the mini-batch. The term $\mathbf{c}^\top$ denotes the row vector containing the total costs for every example in the mini-batch.

3.1 Fast vector-Jacobian multiplication

While Eqn (3) suggests that the Jacobian might have to be formed explicitly, Pearlmutter 1994 showed that computing a differential derivative suffices to compute left or right vector-Jacobian (or Hessian) multiplication. The same trick has also recently been revived with the class of so-called "Hessian-free" (Martens 2010) methods for artificial neural networks. Using the notation of Pearlmutter 1994, we write $\mathcal{R}_{\theta_i}\{\cdot\} = \mathbf{c}^\top \nabla_{\theta_i}$ for the differential operator.

$$\nabla_{\theta_i} \mathcal{L}(\theta) \approx \frac{1}{N} \mathcal{R}_{\theta_i} \{\log \pi(\mathbf{U}_i \,|\, \mathbf{S}_i)\} \tag{4}$$

3.2 Kullback-Leibler constraint

In order to favor dropout policies with *sparse* actions, we could add a penalty term for active units sampled at every layer, e.g. using an L_0 or L_1 norm. This approach is akin to taking a sample of the activation levels in expectation over all possible decisions and initial states (input example). For computational reasons, we instead use an approach similar to Lee, Ekanadham, and Ng 2008, penalizing for the Kullback-Leibler (KL) divergence of the current probability over actions with respect to some base distribution, defined as:

$$\mathrm{KL}(\pi \| q) = \sum_{\mathbf{u} \in \mathcal{U}} \pi(\mathbf{u} \,|\mathbf{s}_i) \log \frac{\pi(\mathbf{u} \,|\, \mathbf{s}_i)}{q(\mathbf{u} \,|\, \mathbf{s}_i)}$$

Because the KL divergence of independent distributions is additive (Cover and Thomas 2006), however, and given the form chosen for the policies, we can simplify this as:

$$\mathrm{KL}(\pi \| q) = \sum_{i=1}^{L} \sigma_i (\log \sigma_i - \log q_i) + (1 - \sigma_i)(\log(1 - \sigma_i) - \log(1 - q_i)) \tag{5}$$

3.3 Algorithm

We interleave standard learning of the network parameters by backpropagation (Rumelhart, Hinton, and Williams 1988) with a REINFORCE-style approach to update the policy. Two functions are computed: the negative log likelihood loss for the network parameters under mini-batch $\mathbf{X}$ for output $\mathbf{Y}$ and the full set of parameters θ; the cumulative cost from the MDP, for each example in the mini-batch:

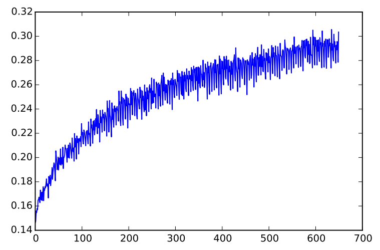
$$C(\tau) = -\log P(\mathbf{Y} \,|\, \mathbf{X}, \theta) + \lambda \sum_{i=1}^{L} \mathrm{KL}(\pi \,|\, q) \tag{6}$$

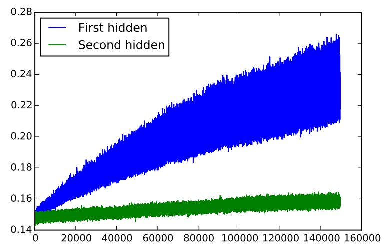
where λ can be understood has a tradeoff parameter between prediction accuracy and parsimony of computation (obtained through sparse node activation). The distribution q encodes the desired target sparsity.

4 Experiments

The proposed model was implemented within Theano (Bergstra et al. 2010), a standard library for deep learning and neural networks. We first evaluated the performance of a standard neural network with 28x28 inputs, 500 hidden units, and 10 outputs over the MNIST dataset. An L_2 penalty with regularization coefficient of 0.001 was imposed on the weight matrices and a learning rate of 0.01 was maintained throughout the stochastic gradient steps. The weight matrices were initialized using the heuristic of Glorot and Bengio 2010. We also used early stopping (Bishop 2006) to avoid overfitting. The dataset of 60000 examples was fed through the network in mini-batches of size $m_b = 20$ for a maximum of 1000 epochs. A test set of 10000 examples was used to measure performance. Under these conditions, a test error of 1.7% was achieved.

Based on the same architecture, we then proceeded to train a dropout policy using the method outlined above. In order to explore the effect of the tradeoff parameter λ in (6), we randomly initialized the parameters of our policy so that the initial number of active units was about 10-15%. Since the negative likelihood term should completely dominate in (6) in this case, we would expect that the optimal policy would achieve its goal by recruiting as many units as possible. This behavior can be observed in Figure 1a where about 30% of the units were active when the testing error was the lowest. While the test error of 2% did not suffer significantly, the number of training epochs increased compared to the *vanilla*


(a)



(b)

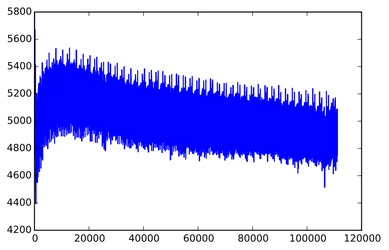

(c)

Figure 1: (a)-(b), mean activation during learning for 500 and 800-800 network. (c), entropy of the policy during training

network. This can be explained by the fact that there are now two kinds of updates (the network updates and the policy updates), and therefore the learning problem is perhaps more complicated.

As in the experiment of Hinton et al. 2012, we also tried a network consisting of two hidden layers of 800 units. We kept the same learning rates (and initial conditions) but used a tradeoff parameter λ of 0.0001. Interestingly, the rate of mean activations over time differs for the two layers. We can observe in figure 1b that the first layer increases the number of active units much faster than the second layer. This phenomenon seems to agree with the general view that the deeper levels of the network process higher level concepts. The higher mean activation would be explained by the higher computational demand for teasing apart the complexities at the lower levels. When plotting the entropy of the first policy, we see that it decreases as more structure is discovered and becomes more deterministic.

One of the main advantages of state-of-art deep network implementations is the fact that they can exploit GPU computations. The approach presented so far, however, does not take any GPU considerations into account. We also explored using "block dropout" where, instead of predicting/generating dropout for a single unit, in each instance at a time, we will drop out the same units for all instances in a block (a computation which can be done more efficiently on a GPU). Using roughly the same architecture but with slightly increased sparity levels ($\leq 15\%$), we get around 10% speedups (with blocks of size 32) on just the forward pass through such a network. By increasing sparsity we achieve even greater speedups (compared to individual dropout decisions for each instance and each node). These numbers do not include the computation of the dropout, but for reasonably big block sizes, this should be significantly less expensive.

Using the same dropout on a mini-batch can be viewed as a restriction on the space of possible policies. Hence, it increases the policy bias, for the sake of faster computation. We are currently exploring how this approach affects the quality of the learning itself. More experimentation along these lines is ongoing.

5 Related work

Ba and Frey 2013 proposed a learning algorithm called *standout* for computing an input-dependent dropout distribution at every node. As opposed to our stage-wise method, standout computes a one-shot dropout mask over the entire network, conditioned on the input to the network. Bengio, Léonard, and Courville 2013 introduce Stochastic Times Smooth neurons as gaters for conditional computation within a deep neural network. STS neurons are highly non-linear and non-differentiable functions learned using estimators of the gradient obtained through REINFORCE. They allow a sparse binary gater to be computed as a function of the input, thus reducing total computations in the (then sparse) activation of hidden layers.

Stollenga et al. 2014 recently proposed to learn a sequential decision process over the filters of a convolutional neural network (CNN). As in our work, a direct policy search method was chosen to find the parameters of a control policy. Their MDP formulation differs from ours mainly in the notion of decision "stage". In their model, an input is first fed through a network and the activations computed during forward propagation then served to the next decision stage. The goal of the policy is to select relevant filters from the previous stage so as to improve the decision accuracy on the current example. They also use a gradient-free evolutionary algorithm, in contrast to our policy search method.

The Deep Sequential Neural Network (DSNN) model of Denoyer and Gallinari 2014 is possibly closest to our approach. The control process is carried over the layers of the network and uses the output of the previous layer to compute actions. The REINFORCE algorithm is used to train the policy with the reward/cost function being defined as the loss at the output in the base network. DSNN considers the general problem of choosing between between different type of mappings in a composition of functions. However, they propose to use a Gibbs policy, which may make this method too expensive in networks with a large number of nodes.

6 Conclusion

We presented our first steps towards solving the problem of conditional computation in deep networks by using reinforcement learning. We proposed a type of parametrized dropout policy that maps the activations of a layer to a Bernoulli mask. The reinforcement signal accounts both for the loss function of the network in its prediction task, as well as the desire to have sparse computations (as a proxy for fast learning). The REINFORCE (Williams 1992) algorithm used to train policies to optimize this reward. Our experiments showed that it is possible to train such models at the same levels of accuracy as their standard counterparts. When valuing only the prediction score, we saw that the algorithm tried compensate by recruiting more units. Although we were able to train our model, the increased number of training iterations currently hinders its viability. A larger parameter sweep might allow for more suitable learning rates to be found. It is possible that the momentum based aggressive learning rate strategy of Hinton et al. 2012 might be needed. Another possibility would be to explore asynchronous policy updates. The base neural network could be first trained normally but augmented with the dropout policy only after a sufficient number of steps. The intuition would be that some compression/compilation could be discovered by our algorithm after a maturation phase.

The use of REINFORCE could be replaced by a more efficient policy search algorithm, and also, perhaps, one in which rewards as described above are replaced by a more sequential notion of return. The more direct use of computation time (rather than the current KL penalty) may also prove beneficial. In general, we consider conditional computation to be an area in which reinforcement learning could be a very useful approach, which deserves to be studied further.

References

Ba, Jimmy and Brendan Frey (2013). "Adaptive dropout for training deep neural networks". In: *Advances in Neural Information Processing Systems 26*. Ed. by C.J.C. Burges et al. Curran Associates, Inc., pp. 3084–3092.

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

Bergstra, James et al. (2010). "Theano: a CPU and GPU Math Expression Compiler". In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. Austin, TX.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Cover, Thomas M. and Joy A. Thomas (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience.

Denoyer, Ludovic and Patrick Gallinari (2014). "Deep Sequential Neural Network". In: *CoRR* abs/1410.0510.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pp. 249–256.

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv preprint arXiv:1502.01852*.

Hinton, Geoffrey E. et al. (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580.

Hochreiter, Sepp et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*.

Lee, Honglak, Chaitanya Ekanadham, and Andrew Y. Ng (2008). "Sparse deep belief net model for visual area V2". In: *Advances in Neural Information Processing Systems 20*. Ed. by J.C. Platt et al. Curran Associates, Inc., pp. 873–880.

Martens, James (2010). "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.

Pearlmutter, Barak A. (1994). "Fast Exact Multiplication by the Hessian". In: *Neural Comput.* 6.1, pp. 147–160.

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. New York, NY, USA: John Wiley & Sons, Inc.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1988). "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.

Stollenga, Marijn F et al. (2014). "Deep Networks with Internal Selective Attention through Feedback Connections". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 3545–3553.

Wager, Stefan, Sida Wang, and Percy S Liang (2013). "Dropout training as adaptive regularization". In: *Advances in Neural Information Processing Systems*, pp. 351–359.

Williams, Ronald J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". English. In: *Machine Learning* 8.3-4, pp. 229–256.