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

arXiv:1511.06297v1 [cs.LG] 19 Nov 2015

ABSTRACT

Deep learning has become the state-of-art tool in many applications, but the evaluation and training of deep models can be time-consuming and computationally expensive. Dropout has been shown to be an effective strategy to sparsify computations (by not involving all units), as well as to regularize 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 for blocks of units as a reinforcement learning problem. 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.

Keywords Neural Networks, Conditional Computing, REINFORCE

1 INTRODUCTION

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 problem (Hochreiter, 1991; Bengio et al., 1994), 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 as a form of regularization to prevent overfitting (Wager et al., 2013). It has now become standard in most convolutional neural network architectures.

The primary motivation for the current work is to provide a sensible approach to reduce computation time in deep networks. As opposed to the original dropout, we propose to learn input-dependent dropout 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. We present the problem

IPR2025-01399
CrowdStrike EX1022 Page 1

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 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, continuous state and discrete action MDP $\langle \mathcal{S}, \mathcal{U}, P(\cdot \mid s, u), C \rangle$ with C the cost function and $P(\cdot \mid s, u)$ 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 train a policy which allows the parameters 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:

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

where the σ_i denotes the *participation* probability (complement of the dropout probability), to be computed from the activations $\mathbf{s}$ 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 output of a typical hidden layer $h(x)$ that uses this policy is multiplied element-wise with the mask $\mathbf{u}$ sampled from the probabilities σ, and becomes $(h(x) \otimes \mathbf{u})$.

3 LEARNING SIGMOID-BERNOULLI POLICIES

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

$$\nabla_{\theta_l} \mathcal{L}(\theta) = \mathbb{E}\left\{ C(\mathbf{x}) \nabla_{\theta_l} \log \pi(\mathbf{u}^{(l)} \mid \mathbf{s}^{(l)}) \right\} \tag{2}$$

since θ_l only appears in the l-th 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 modern hardware. Under the mini-batch setting, $\mathbf{s}^{(l)}$ becomes a matrix and $\pi(\cdot \mid \cdot)$ 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:

$$\nabla_{\theta_l} \mathcal{L}(\theta) \approx \frac{1}{m_b} \sum_{i=1}^{m_b} C(\mathbf{x}_i) \nabla_{\theta_l} \log \pi(\mathbf{u}_i^{(l)} \mid \mathbf{s}_i^{(l)}) = \frac{1}{m_b} \mathbf{c}^\top \nabla_{\theta_l} \log \pi(\mathbf{U}^{(l)} \mid \mathbf{S}^{(l)}) \tag{3}$$

where $C(\mathbf{x}_i)$ is the total cost for input $\mathbf{x}_i$ and m_b 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 $\mathcal{R}_{\theta_l}\{\cdot\} = \mathbf{c}^\top \nabla_{\theta_l}$ for the differential operator.

$$\nabla_{\theta_l} \mathcal{L}(\theta) \approx \frac{1}{m_b} \mathcal{R}_{\theta_l} \left\{ \log \pi(\mathbf{U}^{(l)} \mid \mathbf{S}^{(l)}) \right\} \tag{4}$$

IPR2025-01399
CrowdStrike EX1022 Page 2

3.2 SPARSITY AND VARIANCE REGULARIZATIONS

In order to favour dropout policies with *sparse* actions, we add two penalty terms L_b and L_e that depend on some target sparsity rate τ. The first term pushes the policy distribution π to activate each unit with probability τ in expectation over the data. The second term pushes the policy distribution to have the desired sparsity of activations for each example. Thus, for a low τ, a valid configuration would be to learn a few high probability activations for some part of the data and low probability activations for the rest of the data, which results in having activation probability τ in expectation.

$$L_b = \sum_j^n \|\mathbb{E}\{\sigma_j\} - \tau\|_2 \qquad L_e = \mathbb{E}\{\|(\frac{1}{n}\sum_j^n \sigma_j) - \tau\|_2\} \tag{5}$$

Since we are in a minibatch setting, these expectations can be approximated over the minibatch:

$$L_b \approx \sum_j^n \|\frac{1}{m_b}\sum_i^{m_b}(\sigma_{ij}) - \tau\|_2 \qquad L_e \approx \frac{1}{m_b}\sum_i^{m_b} \|(\frac{1}{n}\sum_j^n \sigma_{ij}) - \tau\|_2 \tag{6}$$

We finally add a third term, L_v, in order to favour the aforementioned configurations, where units only have a high probability of activation for certain examples, and low for the rest. We aim to maximize the variances of activations of each unit, across the data. This encourages units' activations to be varied, and while similar in spirit to the L_b term, this term explicitly discourages learning a uniform distribution.

$$\begin{aligned} L_v &= -\sum_j^n \text{var}\ \mathbb{E}_i\{\sigma_{ij}\} \\ &\approx -\sum_j^n \frac{1}{m_b}\sum_i^{m_b}\left(\sigma_{ij} - \left(\frac{1}{m_b}\sum_i^{m_b}\sigma_{ij}\right)\right)^2 \end{aligned} \tag{7}$$

3.3 ALGORITHM

We interleave the learning of the network parameters and the learning of the policy parameters. We first update the network *and* policy parameters to minimize the following regularized loss function via backpropagation (Rumelhart et al., 1988):

$$\mathcal{L} = -\log P(\mathbf{Y} \mid \mathbf{X}, \theta) + \lambda_s(L_b + L_e) + \lambda_v(L_v) + \lambda_{L2}\|\theta\|^2$$

where λ_s can be understood as a trade-off parameter between prediction accuracy and parsimony of computation (obtained through sparse node activation), and λ_v as a trade-off parameter between a stochastic policy and a more input dependent saturated policy. We then minimize the cost function C with a REINFORCE-style approach to update the policy parameters (Williams, 1992):

$$C = -\log P(\mathbf{Y} \mid \mathbf{X}, \theta)$$

As previously mentioned, we use minibatch stochastic gradient descent as well as minibatch policy gradient updates.

3.4 BLOCK POLICY DROPOUT

To achieve computational gain, instead of dropping single units in hidden layers, we drop contiguous (equally-sized) groups of units together (*independently* for each example in the minibatch), thus reducing the action space as well as the number of probabilities to compute and sample.
As such, there are two potential speedups. First, the policy is much smaller and faster to compute. Second, it offers a computational advantage in the computation of the hidden layer themselves, since we are now performing a matrix multiplication of the following form:

$$C = ((A \otimes M_a)B) \otimes M_c$$

where M_a and M_c are binary mask matrices that resemble this (here there are 3 *blocks* of size 2):

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ & & \ldots & & & \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

This allows us to quickly perform matrix multiplication by only considering the non-zero output elements in C as well as the non-zero elements in $A \otimes M_a$.

IPR2025-01399
CrowdStrike EX1022 Page 3

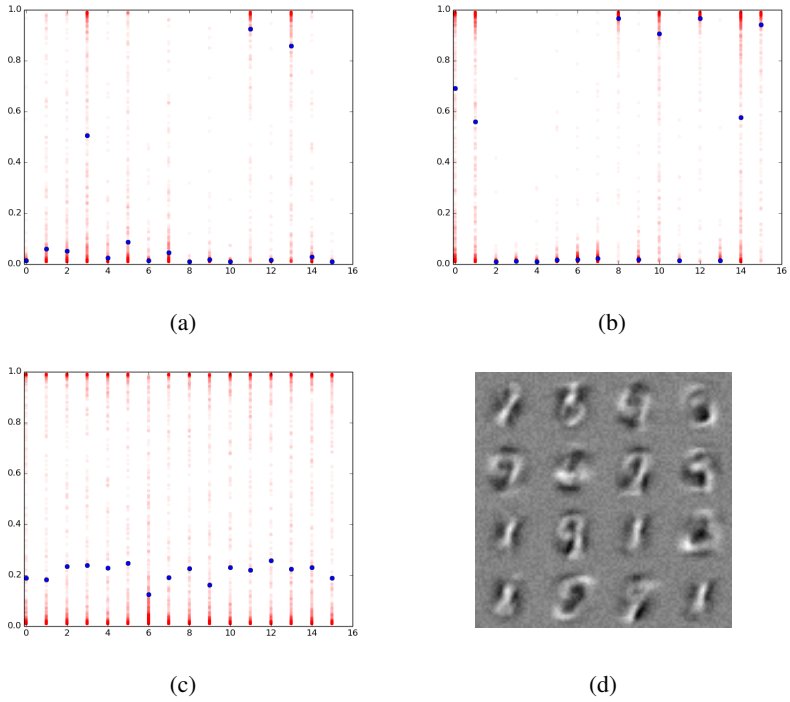


Figure 1: MNIST, (a,b,c), probability distribution of the policy, each example's probability (y axis) of activating each unit (x axis) is plotted as a transparent red dot. Redder regions represent more examples falling in the probability region. Plot (a) is for class '0', (b) for class '1', (c) for all classes. (d), weight matrix of the policy.

4 EXPERIMENTS

4.1 MODEL IMPLEMENTATION

The proposed model was implemented within Theano (Bergstra et al., 2010), a standard library for deep learning and neural networks. In addition to using optimizations offered by Theano, we also implemented specialized matrix multiplication code for the operation exposed in section 3.4.
A straightforward and fairly naive CPU implementation of this operation yielded speedups of up to 5-10x, while an equally naive GPU implementation yielded speedups of up to 2-4x, both for sparsity rates of under 20% and acceptable matrix and block sizes.[1]

We otherwise use fairly standard methods for our neural network. The weight matrices are initialized using the heuristic of Glorot & Bengio (2010). We use a constant learning rate throughout minibatch SGD. We also use early stopping (Bishop, 2006) to avoid overfitting.

4.2 MODEL EVALUATION

We first evaluate the performance of our model on the MNIST digit dataset. We use a single hidden layer of 16 blocks of 16 units (256 units total), with a target sparsity rate of $\tau = 6.25\% = 1/16$, learning rates of 10^{-3} for the neural network and 5×10^{-5} for the policy, $\lambda_v = \lambda_s = 200$ and $\lambda_{L2} = 0.005$. Under these conditions, a test error of around 2.3% was achieved. A normal neural network with the same number of hidden units achieves a test error of around 1.9%, while a normal neural network with a similar *amount* of computation (multiply-adds) being made (32 hidden units) achieves a test error of around 2.8%.

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

IPR2025-01399
CrowdStrike EX1022 Page 4

model	test error	τ	#blocks	block size	test time	speedup
condnet	0.511	1/24	24,24	64	6.8s(26.2s)	3.8×
condnet	0.514	1/16	16,32	16	1.4s (8.2s)	5.7×
condnet	**0.497**	1/16	10,10	64	2.0s(10.4s)	5.3×
bdNN	0.629	0.17	10,10	64	1.93s(10.3s)	5.3×
bdNN	0.590	0.2	10,10	64	2.8s(10.3s)	3.5×
NN	0.560	-	64,64	1	1.23s	-
NN	0.546	-	128,128	1	2.31s	-
NN	**0.497**	-	480,480	1	8.34s	-

Figure 2: CIFAR-10, condnet: our approach, NN: Neural Network without the conditional dropout, bdNN, block dropout Neural Network using a uniform policy. 'speedup' is how many times faster the forward pass is when using a specialized implementation (3.4). '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 time without the optimization.

Looking at the activation of the policy (1c), we see that it tends towards what was hypothesized in section 3.2, i.e. where examples activate most units with low probability and some units with high probability. We can also observe that the policy is input-dependent in figures 1a and 1b, since we see different activation patterns for inputs of class '0' and inputs of class '1'. Since the computation performed in our model is sparse, one could hope that it achieves this performance with less computation time, yet we consistently observe that models that deal with MNIST are too small to allow our specialized (3.4) sparse implementation to make a substantial difference. We include this result to highlight conditions under which it is less desirable to use our model.

Next, we consider the performance of our model on the CIFAR-10 (Krizhevsky & Hinton, 2009) image dataset. A brief hyperparameter search was made, and a few of the best models are shown in figure 2. These results show that it is possible to achieve similar performance with our model (denoted condnet) as with a normal neural network (denoted NN), yet using sensibly reduced computation time. A few things are worth noting; we can set τ to be lower than 1 over the number of blocks, since the model learns a policy that is actually not as sparse as τ, mostly because REINFORCE pulls the policy towards higher probabilities on average. For example our best performing model has a target of $1/16$ but learns policies that average an 18% sparsity rate (we used $\lambda_v = \lambda_b = 20$, except for the first layer $\lambda_v = 40$, we used $\lambda_{L2} = 0.01$, and the learning rates were 0.001 for the neural net, 10^{-5} and 5×10^{-4} for the first and second policy layers respectively). The neural networks without conditional dropout are trained with L2 regularization as well as regular dropout (Hinton et al., 2012).
We also train networks with the same architecture as our models, using block dropout, but with a uniform policy (as in original dropout) instead of a learned conditional one. This model (denoted bdNN) does not perform as well as our model, showing that the dropout noise by itself is not sufficient, and that learning a policy is required to fully take benefit of this architecture. Finally we tested our model on the Street View House Numbers (SVHN) (Netzer et al., 2011) dataset, which also yielded encouraging results (figure 3). As we restrain the capacity of the models (by increasing sparsity or decreasing number of units), condnets retain acceptable performance with low run times, while plain neural networks suffer highly (their performance dramatically decreases with lower run times).
The best condnet model has a test error of 7.3%, and runs a validation epoch in 10s (14s without speed optimization), while the best standard neural network model has a test error of 9.1%, and runs in 16s. Note that the variance in the SVHN results (figure 3) is due to the mostly random hyperparameter exploration, where block size, number of blocks, τ, λ_v, λ_b, as well of learning rates are randomly picked. The normal neural network results were obtained by varying the number of hidden units of a 2-hidden-layer model.

4.3 Effects of regularization

The added regularization proposed in section 3.2 seems to play an important role in our ability to train the conditional model. When using only the prediction score, we observed that the algorithm tried to compensate by recruiting more units and saturating their participation probability, or even

IPR2025-01399
CrowdStrike EX1022 Page 5

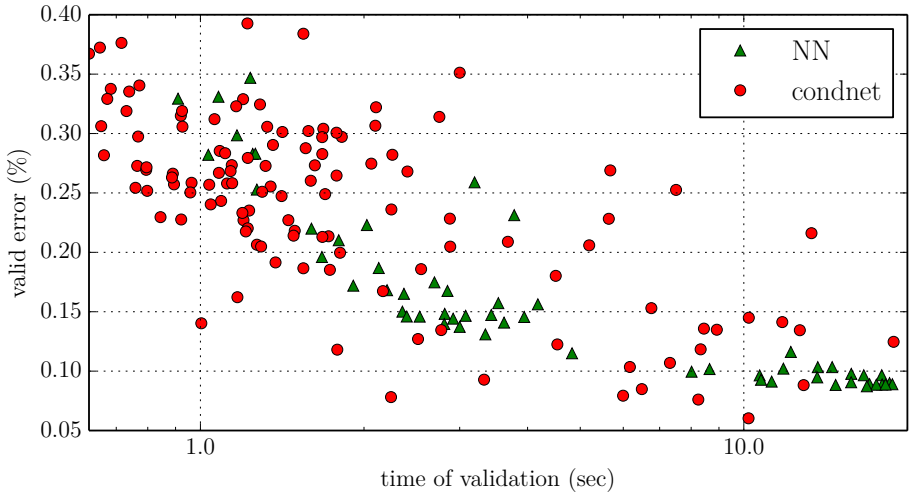


Figure 3: SVHN, each point is an experiment. The x axis is the time required to do a full pass over the valid dataset (log scale, lower is better).

model	test error	τ	#blocks	block size	test time	speedup
condnet	0.183	1/11	13,8	16	1.5s(2.2s)	1.4×
condnet	0.139	1/25,1/7	27,7	16	2.8s (4.3s)	1.6×
condnet	**0.073**	1/22	25,22	32	10.2s(14.1s)	1.4×
NN	0.116	-	288,928	1	4.8s	-
NN	0.100	-	800,736	1	10.7s	-
NN	0.091	-	1280,1056	1	16.8s	-

Figure 4: SVHN results (see fig 2)

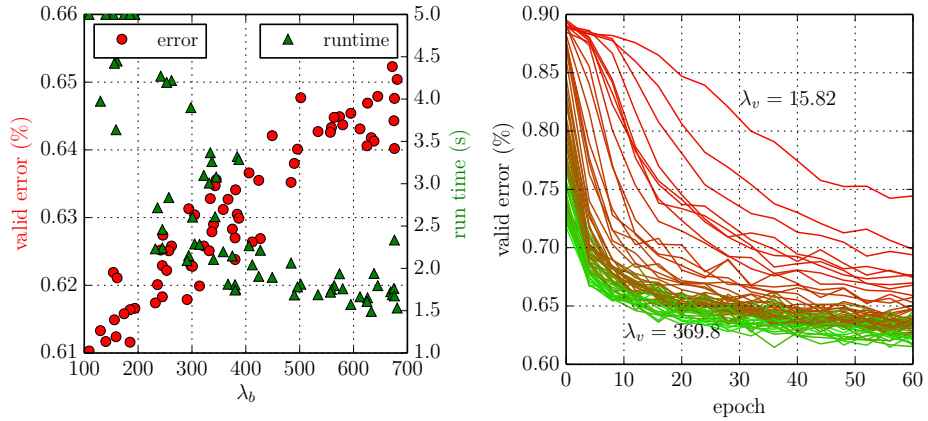
failed by dismissing very early what were probably considered *bad* units.
In practice, the variance regularization term L_v only slightly affects the prediction accuracy and learned policies of models, but we have observed that it significantly speeds up the training process, probably by encouraging policies to become less uniform earlier in the learning process. This can be seen in figure 5b, where we train a model with different values of λ_v. When λ_v is increased, the first few epochs have a much lower error rate.

It is possible to tune some hyperparameters to affect the point at which the trade-off between computation speed and performance lies, thus one could push the error downwards at the expense of also more computation time. This is suggested by figure 5a, which shows the effect of one such hyperparameter (λ_b) on both running times and performance for the CIFAR dataset. Here it seems that $\lambda \sim [300, 400]$ offers the best trade-off, yet other values could be selected, depending on the specific requirements of an application.

5 Related work

Ba & Frey (2013) proposed a learning algorithm called *standout* for computing an input-dependent dropout distribution at every node. As opposed to our layer-wise method, standout computes a one-shot dropout mask over the entire network, conditioned on the input to the network. Additionally, masks are unit-wise, while our approach uses masks that span blocks of units. Bengio et al. (2013) introduced 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 (Williams, 1992). They allow a sparse

IPR2025-01399
CrowdStrike EX1022 Page 6



(a) (b)

Figure 5: CIFAR-10, (a) 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_b increases the running time decreases, but so does performance. (b) The same model is being trained with different values of λ_v. Redder means lower, greener means higher.

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 problem formulation differs from ours mainly in the notion of decision "stage". In their model, an input is first fed through a network, the activations are computed during forward propagation then they are 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 & 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 (weights) in a composition of functions. However, they test their model on datasets in which different modes are proeminent, making it easy for a policy to distinguish between them.

Another point of comparison for our work are attention models (Mnih et al., 2014; Gregor et al., 2015; Xu et al., 2015). These models typically learn a policy, or a form of policy, that allows them to selectively attend to parts of their input *sequentially*, in a visual 2D environnement. Both attention and our approach aim to reduce computation times. While attention aims to perform dense computations on subsets of the inputs, our approach aims to be more general, since the policy focuses on subsets of the whole computation (it is in a sense more distributed). It should also be possible to combine these approaches, since one acts on the input space and the other acts on the representation space, altough the resulting policies would be much more complex, and not necessarily easily trainable.

6 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 block-dropout 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

IPR2025-01399
CrowdStrike EX1022 Page 7

for the desire to have sparse computations. The REINFORCE algorithm is used to train policies to optimize this cost. 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 control the trade-off between accuracy and running time.

The use of REINFORCE could be replaced by a more efficient policy search algorithm, and also, perhaps, one in which rewards (or costs) as described above are replaced by a more sequential variant. The more direct use of computation time as a cost may prove beneficial. In general, we consider conditional computation to be an area in which reinforcement learning could be very useful, and deserves further study.

All the running times reported in the Experiments section are for a CPU, running on a single core. The motivation for this is to explore deployment of large neural networks on cheap, low-power, single core CPUs such as phones, while retaining high model capacity and expressiveness. While the results presented here show that our model for conditional computation can achieve speedups in this context, it is worth also investigating adaptation of these sparse computation models in multi-core/GPU architectures; this is the subject of ongoing work.

ACKNOWLEDGEMENTS

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

Ba, Jimmy and Frey, Brendan. Adaptive dropout for training deep neural networks. In Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 3084–3092. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5032-adaptive-dropout-for-training-deep-neural-networks.pdf.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Nets*, pp. 157–166, 1994.

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

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

Bishop, Christopher M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Deisenroth, Marc Peter, Neumann, Gerhard, and Peters, Jan. 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.

Denoyer, Ludovic and Gallinari, Patrick. Deep sequential neural network. *CoRR*, abs/1410.0510, 2014. URL http://arxiv.org/abs/1410.0510.

Glorot, Xavier and Bengio, Yoshua. 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, 2010. URL http://www.jmlr.org/proceedings/papers/v9/glorot10a.html.

Gregor, Karol, Danihelka, Ivo, Graves, Alex, and Wierstra, Daan. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

IPR2025-01399
CrowdStrike EX1022 Page 8

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

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

Hochreiter, S. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, T.U. Münich, 1991.

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

Martens, James. 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.

Mnih, Volodymyr, Heess, Nicolas, Graves, Alex, and kavukcuoglu, koray. Recurrent models of visual attention. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2204–2212. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5542-recurrent-models-of-visual-attention.pdf.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. 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.

Pearlmutter, Barak A. 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.

Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

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

Stollenga, Marijn F, Masci, Jonathan, Gomez, Faustino, and Schmidhuber, Jürgen. Deep networks with internal selective attention through feedback connections. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3545–3553. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5276-deep-networks-with-internal-selective-attention-through-feedback-connections.pdf.

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

Williams, Ronald J. 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.

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Courville, Aaron, Salakhutdinov, Ruslan, Zemel, Richard, and Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

IPR2025-01399
CrowdStrike EX1022 Page 9