Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Stochastic neurons and hard non-linearities can be useful for a number of reasons in deep learning models, but in many cases they pose a challenging problem: how to estimate the gradient of a loss function with respect to the input of such stochastic or non-smooth neurons? I.e., can we "back-propagate" through these stochastic neurons? We examine this question, existing approaches, and present three novel families of solutions, applicable in different settings. In particular, it is demonstrated that a simple biologically plausible formula gives rise to an an unbiased (but noisy) estimator of the gradient with respect to a binary stochastic neuron firing probability. Unlike other estimators which view the noise as a small perturbation in order to estimate gradients by finite differences, this estimator is unbiased even without assuming that the stochastic perturbation is small. The second approach we propose decomposes the operation of a binary stochastic neuron into a stochastic binary part and a smooth differentiable part, which approximates the expected effect of the pure stochastic binary neuron to first order. A third approach involves the injectiion of additive or multiplicative noise in a computational graph that is otherwise differentiable. To explore these approaches we consider a small-scale version of *conditional computation*, where sparse stochastic units form a distributed representation of gaters that can turn off in combinatorially many ways large chunks of the computation performed in the rest of the neural network. In this case, it is important that the gating units produce an actual 0 most of the time. The resulting sparsity can be potentially be exploited to greatly reduce the computational cost of large deep networks for which conditional computation would be useful.

1 Introduction and Background

Many learning algorithms and in particular those based on neural networks or deep learning rely on gradient-based learning. To compute exact gradients, it is better if the relationship between parameters and the training objective is continuous and generally smooth. If it is only constant by parts, i.e., mostly flat, then gradient-based learning is impractical. This was what motivated the move from neural networks based on so-called formal neurons, with a hard threshold output, to neural networks whose units are based on a sigmoidal non-linearity, and the well-known back-propagation algorithm to compute the gradients (Rumelhart *et al.*, 1986).

We call the *computational graph* or *flow graph* the graph that relates inputs and parameters to outputs and training criterion. Although it had been taken for granted by most researchers that smoothness of this graph was a necessary condition for exact gradient-based training methods to work well, recent successes of deep networks with rectifiers and other "non-smooth" non-linearities (Glorot *et al.*, 2011; Krizhevsky *et al.*, 2012a; Goodfellow *et al.*, 2013) clearly question that belief: see Section 2 for a deeper discussion.

1

In principle, even if there are hard decisions (such as the treshold function typically found in formal neurons) in the computational graph, it is possible to obtain *estimated gradients* by introducing *perturbations* in the system and observing the effects. Although finite-difference approximations of the gradient appear hopelessly inefficient (because independently perturbing each of N parameters to estimate its gradient would be N times more expensive than ordinary back-propagation), another option is to introduce *random perturbations*, and this idea has been pushed far (and experimented on neural networks for control) by Spall (1992) with the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm.

As discussed here (Section 2), non-smooth non-linearities and stochastic perturbations can be combined to obtain reasonably low-variance estimators of the gradient, and a good example of that success is with the recent advances with *dropout* (Hinton *et al.*, 2012; Krizhevsky *et al.*, 2012b; Goodfellow *et al.*, 2013). The idea is to multiply the output of a non-linear unit by independent binomial noise. This noise injection is useful as a regularizer and it does slow down training a bit, but not apparently by a lot (maybe 2-fold), which is very encouraging. The symmetry-breaking and induced sparsity may also compensate for the extra variance and possibly help to reduce ill-conditioning, as hypothesized by Bengio (2013).

However, it is appealing to consider noise whose amplitude can be modulated by the signals computed in the computational graph, such as with *stochastic binary neurons*, which output a 1 or a 0 according to a sigmoid probability. Short of computing an average over an exponential number of configurations, it would seem that computing the exact gradient (with respect to the average of the loss over all possible binary samplings of all the stochastic neurons in the neural network) is impossible in such neural networks. The question is whether good estimators (which might have bias and variance) or similar alternatives can be computed and yield effective training. We come up here with three reasonable solutions to this problem, present theoretical results about them, and small-scale experiments to validate that training can be effective, in the context where one wants to use such stochastic units to gate computation, as described below.

1.1   More Motivations and Conditional Computation

One motivation for studying stochastic neurons is that stochastic behavior may be a required ingredient in *modeling biological neurons*. The apparent noise in neuronal spike trains could come from an actual noise source or simply from the hard to reproduce changes in the set of input spikes entering a neuron's dendrites. Until this question is resolved by biological observations, it is interesting to study how such noise – which has motivated the Boltzmann machine (Hinton *et al.*, 1984) – may impact computation and learning in neural networks.

Stochastic neurons with binary outputs are also interesting because they can easily give rise to *sparse representations* (that have many zeros), a form of regularization that has been used in many representation learning algorithms (Bengio *et al.*, 2013). Sparsity of the representation corresponds to the prior that, for a given input scene, most of the explanatory factors are irrelevant (and that would be represented by many zeros in the representation).

As argued by Bengio (2013), sparse representations may be a useful ingredient of *conditional computation*, by which only a small subset of the model parameters are "activated" (and need to be visited) for any particular example, thereby greatly reducing the number of computations needed per example. Sparse gating units may be trained to select which part of the model actually need to be computed for a given example.

Binary representations are also useful as keys for a hash table, as in the semantic hashing algorithm (Salakhutdinov and Hinton, 2009). Trainable stochastic neurons would also be useful inside recurrent networks to take hard stochastic decisions about temporal events at different time scales. This would be useful to train multi-scale temporal hierarchies [1] such that back-propagated gradients could quickly flow through the slower time scales. Multi-scale temporal hierarchies for recurrent nets have already been proposed and involved exponential moving averages of different time constants (El Hihi and Bengio, 1996), where each unit is still updated after each time step. Instead, identifying key events at a high level of abstraction would allow these high-level units to only be updated when needed (asynchronously), creating a fast track (short path) for gradient propagation through time.

---

[1]Daan Wierstra, personal communication

1.2 Prior Work

The idea of having stochastic neuron models is of course very old, with one of the major family of algorithms relying on such neurons being the Boltzmann machine (Hinton *et al.*, 1984). Another biologically motivated proposal for synaptic strength learning was proposed by Fiete and Seung (2006). It is based on small zero-mean *i.i.d.* perturbations applied at each stochastic neuron potential (prior to a non-linearity) and a Taylor expansion of the expected reward as a function of these variations. Fiete and Seung (2006) end up proposing a gradient estimator that looks like a *correlation between the reward and the perturbation*, very similar to what we obtain in Section 3. However, unlike ours, their estimator is only unbiased in the limit of small perturbations.

Gradient estimators based on stochastic perturbations have long been shown to be much more efficient than standard finite-difference approximations (Spall, 1992). Consider N quantities u_i to be adjusted in order to minimize an expected loss $L(u)$. A finite difference approximation is based on measuring separately the effect of changing each one of the parameters, e.g., through $\frac{L(u)-L(u-\epsilon e_i)}{\epsilon}$, or even better, through $\frac{L(u+\epsilon e_i)-L(u-\epsilon e_i)}{2\epsilon}$, where $e_i = (0, 0, \cdots, 1, 0, 0, \cdots, 0)$ where the 1 is at position i. With N quantities (and typically $O(N)$ computations to calculate $L(u)$), the computational cost of the gradient estimator is $O(N^2)$. Instead, a perturbation-based estimator such as found in Simultaneous Perturbation Stochastic Approximation (SPSA) (Spall, 1992) chooses a random perturbation vector z (e.g., isotropic Gaussian noise of variance σ^2) and estimates the gradient of the expected loss with respect to u_i through $\frac{L(u+z)-L(u-z)}{2z_i}$. So long as the perturbation does not put too much probability around 0, this estimator is as efficient as the finite-difference estimator but requires $O(N)$ less computation. However, like the algorithm proposed by Fiete and Seung (2006) this estimator becomes unbiased only as the perturbations go towards 0. When we want to consider all-or-none perturbations (like a neuron sending a spike or not), it is not clear if these assumptions are appropriate. An advantage of the approaches proposed here is that they do not require that the perturbations be small to be valid.

Another estimator of the expected gradient through stochastic neurons was proposed by Hinton (2012) in his lecture 15b. The idea is simply to back-propagate through the hard threshold function (1 if the argument is positive, 0 otherwise) as if it had been the identity function. It is clearly a biased estimator, but when considering a single layer of neurons, it has the right sign (this is not guaranteed anymore when back-propagating through more hidden layers). We call it the **straight-through** (ST) estimator. A possible variant investigated here multiplies the gradient on h_i by the derivative of the sigmoid.

2 Non-Smooth Stochastic Neurons

One way to achieve gradient-based learning in networks of stochastic neurons is to build an architecture in which noise is injected so that gradients can sometimes flow into the neuron and can then adjust it (and its predecessors in the computational graph) appropriately.

In general, we can consider the output h_i of a stochastic neuron as the application of a deterministic function that depends on some noise source z_i and on some differentiable transformation a_i of its inputs (typically, a vector containing the outputs of other neurons) and internal parameters (typically the bias and incoming weights of the neuron):

$$h_i = f(a_i, z_i) \tag{1}$$

So long as $f(a_i, z_i)$ in the above equation has a non-zero gradient with respect to a_i, gradient-based learning (with back-propagation to compute gradients) can proceed. In this paper, we consider the usual affine transformation $a_i = b_i + \sum_j W_{ij} x_{ij}$, where x_i is the vector of inputs into neuron i.

For example, if the noise z_i is added or multiplied somewhere in the computation of h_i, gradients can be computed as usual. Dropout noise (Hinton *et al.*, 2012) and masking noise (in denoising auto-encoders (Vincent *et al.*, 2008)) is multiplied (just after the neuron non-linearity), while in semantic hashing (Salakhutdinov and Hinton, 2009) noise is added (just before the non-linearity). For example, that noise can be binomial (dropout, masking noise) or Gaussian (semantic hashing).

However, if we want h_i to be binary (like in stochastic binary neurons), then f will have derivatives that are 0 almost everywhere (and infinite at the threshold), so that gradients can never flow.

3

There is an intermediate option that we put forward here: choose f so that it has two main kinds of behavior, with zero derivatives in some regions, and with significantly non-zero derivatives in other regions. We call these two states of the neuron respectively the insensitive state and the sensitive state.

2.1 Noisy Rectifier

A special case is when the insensitive state corresponds to $h_i = 0$ and we have sparsely activated neurons. The prototypical example of that situation is the rectifier unit (Nair and Hinton, 2010; Glorot *et al.*, 2011), whose non-linearity is simply $\max(0, \arg)$, i.e.,

$$h_i = \max(0, z_i + a_i)$$

where z_i is a zero-mean noise. Although in practice Gaussian noise is appropriate, interesting properties can be shown for z_i sampled from the logistic density $p(z) = \text{sigm}(z)(1 - \text{sigm}(z))$, where $\text{sigm}(\cdot)$ represents the logistic sigmoid function.

In that case, we can prove nice properties of the noisy rectifier.

Proposition 1. *The noisy rectifier with logistic noise has the following properties: (a) $P(h_i > 0) = \text{sigm}(a_i)$, (b) $E[h_i] = s_+(a_i)$, where $s_+(x) = \log(1 + \exp(x))$ is the softplus function and the random variable is z_i (given a fixed a_i).*

Proof.

$$
\begin{aligned}
P(h_i > 0) &= P(z_i > -a_i) = 1 - \text{sigm}(-a_i) \\
&= 1 - \frac{1}{1 + e^{a_i}} = \frac{1}{1 + e^{-a_i}} = \text{sigm}(a_i)
\end{aligned}
$$

which proves (a). For (b), we use the facts $\frac{d\,\text{sigm}(x)}{dx} = \text{sigm}(x)(1 - \text{sigm}(x))$, $s_+(u) \to u$ as $u \to \infty$, change of variable $x = a_i + z_i$ and integrate by parts:

$$
\begin{aligned}
E[h_i] &= \lim_{t \to \infty} \int_{z_i > -a_i}^{t} (a_i + z_i)\text{sigm}(z_i)(1 - \text{sigm}(z_i))dz_i \\
&= \lim_{t \to \infty} \left([x\,\text{sigm}(x - a_i)]_0^t - \int_0^t \text{sigm}(x - a)dx \right) \\
&= \lim_{t \to \infty} [x\,\text{sigm}(x - a_i) - s_+(x - a_i)]_0^t \\
&= a_i + s_+(-a_i) = s_+(a_i)
\end{aligned}
$$

$\square$

Let us now consider two cases:

1. If $f(a_i, 0) > 0$, the basic state is *active*, the unit is generally sensitive and non-zero, but sometimes it is shut off (e.g., when z_i is sufficiently negative to push the argument of the rectifier below 0). In that case gradients will flow in most cases (samples of z_i). If the rest of the system sends the signal that h_i should have been smaller, then gradients will push it towards being more often in the insensitive state.

2. If $f(a_i, 0) = 0$, the basic state is *inactive*, the unit is generally insensitive and zero, but sometimes turned on (e.g., when z_i is sufficiently positive to push the argument of the rectifier above 0). In that case gradients will not flow in most cases, but when they do, the signal will either push the weighted sum lower (if being active was not actually a good thing for that unit in that context) and reduce the chances of being active again, or it will push the weight sum higher (if being active was actually a good thing for that unit in that context) and increase the chances of being active again.

So it appears that even though the gradient does not always flow (as it would with sigmoid or tanh units), it might flow sufficiently often to provide the appropriate training information. The important thing to notice is that even when the basic state (second case, above) is for the unit to be insensitive and zero, *there will be an occasional gradient signal* that can draw it out of there.

4

One concern with this approach is that one can see an asymmetry between the number of times that a unit with an active state can get a chance to receive a signal telling it to become inactive, versus the number of times that a unit with an inactive state can get a signal telling it to become active.

Another potential and related concern is that some of these units will "die" (become useless) if their basic state is inactive in the vast majority of cases (for example, because their weights pushed them into that regime due to random fluctations). Because of the above asymmetry, dead units would stay dead for very long before getting a chance of being born again, while live units would sometimes get into the death zone by chance and then get stuck there. What we propose here is a simple mechanism to *adjust the bias of each unit* so that in average its "firing rate" (fraction of the time spent in the active state) reaches some pre-defined target. For example, if the moving average of being non-zero falls below a threshold, the bias is pushed up until that average comes back above the threshold.

2.2 STS Units: Stochastic Times Smooth

We propose here a novel form of stochastic unit that is related to the noisy rectifier and to the stochastic binary unit, but that can be trained by ordinary gradient descent with the gradient obtained by back-propagation in a computational graph. We call it the STS unit (Stochastic Times Smooth). With a_i the activation before the stochastic non-linearity, the output of the STS unit is

$$p_i = \text{sigm}(a_i)$$
$$b_i \sim \text{Binomial}(\sqrt{p_i})$$
$$h_i = b_i\sqrt{p_i} \tag{2}$$

Proposition 2. *The STS unit benefits from the following properties: (a) $E[h_i] = p_i$, (b) $P(h_i > 0) = \sqrt{\text{sigm}(a_i)}$, (c) for any differentiable function f of h_i, we have*

$$E[f(h_i)] = f(p_i) + o(\sqrt{p_i})$$

as $p_i \to 0$, and where the expectations and probability are over the injected noise b_i, given a_i.

Proof. Statements (a) and (b) are immediately derived from the definitions in Eq. 2 by noting that $E[b_i] = P(b_i > 0) = P(h_i > 0) = \sqrt{p_i}$. Statement (c) is obtained by first writing out the expectation,

$$E[f(h_i)] = \sqrt{p_i}f(\sqrt{p_i}) + (1 - \sqrt{p_i})f(0)$$

and then performing a Taylor expansion of f around p_i as $p_i \to 0$):

$$f(\sqrt{p_i}) = f(p_i) + f'(p_i)(\sqrt{p_i} - p_i) + o(\sqrt{p_i} - p_i)$$
$$f(0) = f(p_i) + f'(p_i)(-p_i) + o(p_i)$$

where $\frac{o(x)}{x} \to 0$ as $x \to 0$, so we obtain

$$
\begin{aligned}
E[f(h_i)] &= \sqrt{p_i}(f(p_i) + f'(p_i)(\sqrt{p_i} - p_i)) + (1 - \sqrt{p_i})(f(p_i) + f'(p_i)(-p_i)) + o(\sqrt{p_i}) \\
&= \sqrt{p_i}f(p_i) + f'(p_i)(p_i - p_i\sqrt{p_i}) + f(p_i)(1 - \sqrt{p_i}) - f'(p_i)(p_i - p_i\sqrt{p_i}) + o(\sqrt{p_i}) \\
&= f(p_i) + o(\sqrt{p_i})
\end{aligned}
$$

using the fact that $\sqrt{p_i} > p_i$, $o(p_i)$ can be replaced by $o(\sqrt{p_i})$. $\qquad\square$

Note that this derivation can be generalized to an expansion around $f(x)$ for any $x \le \sqrt{p_i}$, yielding

$$E[f(h_i)] = f(x) + (x - p_i)f'(x) + o(\sqrt{p_i})$$

where the effect of the first derivative is canceled out when we choose $x = p_i$. It is also a reasonable choice to expand around p_i because $E[h_i] = p_i$.

3 Unbiased Estimator of Gradient for Stochastic Binary Neurons

The above proposals cannot deal with non-linearities like the indicator function that have a derivative of 0 almost everywhere. Let us consider this case now, where we want some component of our model to take a hard binary decision but allow this decision to be stochastic, with a probability that is a continuous function of some quantities, through parameters that we wish to learn. We will also assume that many such decisions can be taken in parallel with independent noise sources z_i driving the stochastic samples. Without loss of generality, we consider here a set of binary decisions, i.e.,

the setup corresponds to having a set of stochastic binary neurons, whose output h_i influences an observed future loss L. In the framework of Eq. 1, we could have for example

$$h_i = f(a_i, z_i) = \mathbf{1}_{z_i > \mathrm{sigm}(a_i)} \tag{3}$$

where $z_i \sim U[0, 1]$ is uniform and $\mathrm{sigm}(u) = 1/(1 + \exp(-u))$ is the sigmoid function. We would ideally like to estimate how a change in a_i would impact L in average over the noise sources, so as to be able to propagate this estimated gradients into parameters and inputs of the stochastic neuron.

Theorem 1. *Let h_i be defined as in Eq. 3, with $L = L(h_i, c_i, c_{-i})$ a loss that depends stochastically on h_i, c_i the noise sources that influence a_i, and c_{-i} those that do not influence a_i, then $\hat{g}_i = (h_i - \mathrm{sigm}(a_i)) \times L$ is an unbiased estimator of $g_i = \frac{\partial E_{z_i, c_{-i}}[L|c_i]}{\partial a_i}$ where the expectation is over z_i and c_{-i}, conditioned on the set of noise sources c_i that influence a_i.*

Proof. We will compute the expected value of the estimator and verify that it equals the desired derivative. The set of all noise sources in the system is $\{z_i\} \cup c_i \cup c_{-i}$. We can consider L to be an implicit deterministic function of all the noise sources (z_i, c_i, c_{-i}), $E_{v_z}[\cdot]$ denotes the expectation over variable v_z, while $E[\cdot|v_z]$ denotes the expectation over all the other random variables besides v_z, i.e., conditioned on v_Z.

$$
\begin{aligned}
E[L|c_i] &= E_{c_{-i}}[E_{z_i}[L(h_i, c_i, c_{-i})]] \\
&= E_{c_{-i}}[E_{z_i}[h_i L(1, c_i, c_{-i}) + (1 - h_i)L(0, c_i, c_{-i})]] \\
&= E_{c_{-i}}[P(h_i = 1|a_i)L(1, c_i, c_{-i}) + P(h_i = 0|a_i)L(0, c_i, c_{-i})] \\
&= E_{c_{-i}}[\mathrm{sigm}(a_i)L(1, c_i, c_{-i}) + (1 - \mathrm{sigm}(a_i))L(0, c_i, c_{-i})]
\end{aligned}
\tag{4}
$$

Since a_i does not influence $P(c_{-i})$, differentiating with respect to a_i gives

$$
\begin{aligned}
g_i \stackrel{\text{def}}{=} \frac{\partial E[L|c_i]}{\partial a_i} &= E_{c_{-i}}\Big[\frac{\partial \mathrm{sigm}(a_i)}{\partial a_i}L(1, c_i, c_{-i}) - \frac{\partial \mathrm{sigm}(a_i)}{\partial a_i}L(0, c_i, c_{-i})|c_i\Big] \\
&= E_{c_{-i}}[\mathrm{sigm}(a_i)(1 - \mathrm{sigm}(a_i))(L(1, c_i, c_{-i}) - L(0, c_i, c_{-i})|c_i]
\end{aligned}
\tag{5}
$$

First consider that since $h_i \in \{0, 1\}$,

$$L(h_i, c_i, c_{-i}) = h_i L(1, c_i, c_{-i}) + (1 - h_i)L(0, c_i, c_{-i})$$

$h_i^2 = h_i$ and $h_i(1 - h_i) = 0$, so

$$
\begin{aligned}
\hat{g}_i \stackrel{\text{def}}{=} (h_i - \mathrm{sigm}(a_i))L(h_i, c_i, c_{-i}) &= h_i(h_i - \mathrm{sigm}(a_i))L(1, c_i, c_{-i}) \tag{6} \\
&\quad + (h_i - \mathrm{sigm}(a_i))(1 - h_i)L(0, c_i, c_{-i})) \\
&= h_i(1 - \mathrm{sigm}(a_i))L(1, c_i, c_{-i}) \tag{7} \\
&\quad - (1 - h_i)\mathrm{sigm}(a_i)L(0, c_i, c_{-i}). \tag{8}
\end{aligned}
$$

Now let us consider the expected value of the estimator $\hat{g}_i = (h_i - \mathrm{sigm}(a_i))L(h_i, c_i, c_{-i})$.

$$
\begin{aligned}
llE[\hat{g}_i] &= E[h_i(1 - \mathrm{sigm}(a_i))L(1, c_i, c_{-i}) - (1 - h_i)\mathrm{sigm}(a_i)L(0, c_i, c_{-i})] \\
&= E_{c_i, c_{-i}}[\mathrm{sigm}(a_i)(1 - \mathrm{sigm}(a_i))L(1, c_i, c_{-i}) - (1 - \mathrm{sigm}(a_i))\mathrm{sigm}(a_i)L(0, c_i, c_{-i})] \\
&= E_{c_i, c_{-i}}[\mathrm{sigm}(a_i)(1 - \mathrm{sigm}(a_i))(L(1, c_i, c_{-i}) - L(0, c_i, c_{-i}))]
\end{aligned}
\tag{9}
$$

which is the same as Eq. 5, i.e., the expected value of the estimator equals the gradient of the expected loss, $E[\hat{g}_i] = g_i$. $\qquad\square$

Corollary 1. *Under the same conditions as Theorem 1, and for any (possibly unit-specific) constant $\bar{L}_i$ the centered estimator $(h_i - \mathrm{sigm}(a_i))(L - \bar{L}_i)$, is also an unbiased estimator of $g_i = \frac{\partial E_{z_i, c_{-i}}[L|c_i]}{\partial a_i}$. Furthermore, among all possible values of $\bar{L}_i$, the minimum variance choice is*

$$\bar{L}_i = \frac{E[(h_i - \mathrm{sigm}(a_i))^2 L]}{E[(h_i - \mathrm{sigm}(a_i))^2]}, \tag{10}$$

which we note is a weighted average of the loss values L, whose weights are specific to unit i.

Proof. The centered estimator $(h_i - \mathrm{sigm}(a_i))(L - \bar{L}_i)$ can be decomposed into the sum of the uncentered estimator $\hat{g}_i$ and the term $(h_i - \mathrm{sigm}(a_i))\bar{L}_i$. Since $E_{z_i}[h_i|a_i] = \mathrm{sigm}(a_i)$, $E[\bar{L}_i(h_i - \mathrm{sigm}(a_i))|a_i] = 0$, so that the expected value of the centered estimator equals the expected value of the uncentered estimator. By Theorem 1 (the uncentered estimator is unbiased), the centered estimator is therefore also unbiased, which completes the proof of the first statement.

Regarding the optimal choice of $\bar{L}_i$, first note that the variance of the uncentered estimator is

$$Var[(h_i - \mathrm{sigm}(a_i))L] = E[(h_i - \mathrm{sigm}(a_i))^2 L^2] - E[\hat{g}_i]^2.$$

Now let us compute the variance of the centered estimator:

$$Var[(h_i - \mathrm{sigm}(a_i))(L - \bar{L}_i)] = E[(h_i - \mathrm{sigm}(a_i))^2 (L - \bar{L}_i)^2] - E[(h_i - \sigma(a_i))(L - \bar{L}_i)]^2$$
$$= E[(h_i - \mathrm{sigm}(a_i))^2 L^2] + E[(h_i - \mathrm{sigm}(a_i))^2 \bar{L}_i^2]$$
$$- 2E[(h_i - \mathrm{sigm}(a_i))^2 L\bar{L}_i] - (E[\hat{g}_i] - 0)^2$$
$$= Var[(h_i - \mathrm{sigm}(a_i))L] - \Delta \tag{11}$$

where $\Delta = 2E[(h_i - \mathrm{sigm}(a_i))^2 L\bar{L}_i] - E[(h_i - \mathrm{sigm}(a_i))^2 \bar{L}_i^2]$. Let us rewrite Δ:

$$\Delta = 2E[(h_i - \mathrm{sigm}(a_i))^2 L\bar{L}_i] - E[(h_i - \mathrm{sigm}(a_i))^2 \bar{L}_i^2]$$
$$= E[(h_i - \mathrm{sigm}(a_i))^2 \bar{L}_i (2L - \bar{L}_i)]$$
$$= E[(h_i - \mathrm{sigm}(a_i))^2 (L^2 - (L - \bar{L}_i)^2)] \tag{12}$$

Δ is maximized (to minimize variance of the estimator) when $E[(h_i - \mathrm{sigm}(a_i))^2 (L - \bar{L}_i)^2]$ is minimized. Taking the derivative of that expression with respect to $\bar{L}_i$, we obtain

$$2E[(h_i - \mathrm{sigm}(a_i))^2 (\bar{L}_i - L)] = 0$$

which, as claimed, is achieved for

$$\bar{L}_i = \frac{E[(h_i - \mathrm{sigm}(a_i))^2 L]}{E[(h_i - \mathrm{sigm}(a_i))^2]}. \qquad \square$$

Practically, we could get the lowest variance estimator (among all choices of the $\bar{L}_i$) by keeping track of two numbers (running or moving averages) for each stochastic neuron, one for the numerator and one for the denominator of the unit-specific $\bar{L}_i$ in Eq. 10. This would lead the lowest-variance estimator $(h_i - \mathrm{sigm}(a_i))(L - \bar{L}_i)$. Note how the unbiased estimator only requires broadcasting L throughout the network, no back-propagation and only local computation. Note also how this could be applied even with an estimate of future rewards or losses L, as would be useful in the context of reinforcement learning (where the actual loss or reward will be measured farther into the future, much after h_i has been sampled).
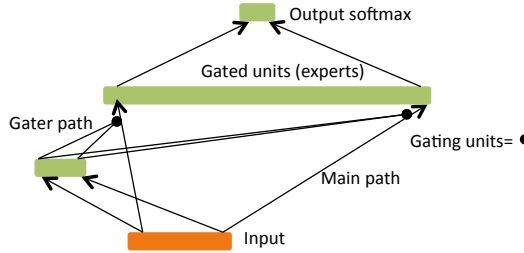

Figure 1: Conditional computation architecture: gater path on the left produces sparse outputs h_i which are multiplied elementwise by the expert outputs, hidden units H_i.

4 Conditional Computation Experiments

We consider the potential application of stochastic neurons in the context of conditional computation, where the stochastic neurons are used to select which parts of some computational graph should be actually computed, given the current input. The particular architecture we experimented with is a neural network with a large hidden layer whose units H_i will be selectively turned off by gating units h_i, i.e., the output of the i-th hidden unit is $H_i h_i$, as illustrated in Figure 1. For this to make sense in the context of conditional computation, we want h_i to be non-zero only a small fraction α of the time (10% in the experiments) while the amount of computation required to compute h_i

7

should be much less than the amount of computation required to compute H_i. In this way, we can first compute h_i and only compute H_i if $h_i \neq 0$, thereby saving much computation. To achieve this, we connect the previous layer to h_i through a bottleneck layer. Hence if the previous layer (or the input) has size N and the main path layer also has size N (i.e., $i \in \{1, \dots N\}$), and the gater's bottleneck layer has size $M \ll N$, then computing the gater output is $O(MN)$, which can reduce the main path computations from $O(N^2)$ to $O(\alpha N^2)$.
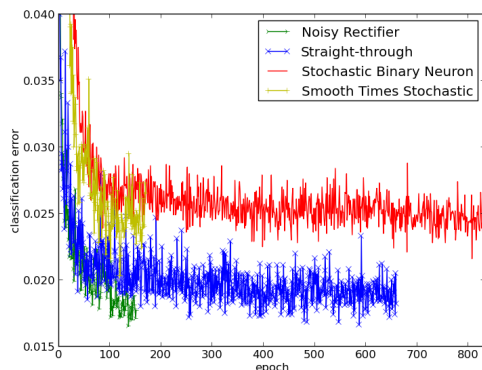
	train	valid	test
Noisy Rectifier	6.7e-4	1.52	1.87
Straight-through	3.3e-3	1.42	1.39
Smooth Times Stoch.	4.4e-3	1.86	1.96
Stoch. Binary Neuron	9.9e-3	1.78	1.89
Baseline Rectifier	6.3e-5	1.66	1.60
Baseline Sigmoid+Noise	1.8e-3	1.88	1.87
Baseline Sigmoid	3.2e-3	1.97	1.92

Figure 2: Left: Learning curves for the various stochastic approaches applied to the MNIST dataset. The y-axis indicates classification error on the validation set. Right: Comparison of training criteria and valid/test classification error (%) for stochastic (top) and baseline (bottom) gaters.

Experiments are performed with a gater of 400 hidden units and 2000 output units. The main path also has 2000 hidden units. A sparsity constraint is imposed on the 2000 gater output units such that each is non-zero 10% of the time, on average. The validation and test errors of the stochastic models are obtained after optimizing a threshold in their deterministic counterpart used during testing. See the appendix in supplementary material for more details on the experiments.

For this experiment, we compare with 3 baselines. The Baseline Rectifier is just like the noisy rectifier, but with 0 noise, and is also constrained to have a sparsity of 10%. The Baseline Sigmoid is like the STS and ST models in that the gater uses sigmoids for its output and tanh for its hidden units, but has no sparsity constraint and only 200 output units for the gater and main part. The baseline has the same hypothetical resource constraint in terms of computational efficiency (runtime), but has less total parameters (memory). The Baseline Sigmoid with Noise is the same, but with Gaussian noise added during training.

5 Conclusion

In this paper, we have motivated estimators of the gradient through highly non-linear non-differentiable functions (such as those corresponding to an indicator function), especially in networks involving noise sources, such as neural networks with stochastic neurons. They can be useful as biologically motivated models and they might be useful for engineering (computational efficiency) reasons when trying to reduce computation via conditional computation or to reduce interactions between parameters via sparse updates (Bengio, 2013). We have proven interesting properties of three classes of stochastic neurons, the noisy rectifier, the STS unit, and the binary stochastic neuron, in particular showing the existence of an unbiased estimator of the gradient for the latter. Unlike the SPSA (Spall, 1992) estimator, our estimator is unbiased even though the perturbations are not small (0 or 1), and it multiplies by the perturbation rather than dividing by it.

Experiments show that all the tested methods actually allow training to proceed. It is interesting to note that the gater with noisy rectifiers yielded better results than the one with the non-noisy baseline rectifiers. Similarly, the sigmoid baseline with noise performed better than without, *even on the training objective*: all these results suggest that injecting noise can be useful not just as a regularizer but also to help explore good parameters and fit the training objective. These results are particularly surprising for the stochastic binary neuron, which does not use any backprop for getting a signal into the gater, and opens the door to applications where no backprop signal is available. In terms of conditional computation, we see that the expected saving is achieved, without an important loss in performance. Another surprise is the good performance of the straight-through units, which provided the best validation and test error, and are very simple to implement.

8

References

Bengio, Y. (2013). Deep learning of representations: Looking forward. Technical Report arXiv:1305.0445, Universite de Montreal.

Bengio, Y., Courville, A., and Vincent, P. (2013). Unsupervised feature learning and deep learning: A review and new perspectives. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*.

El Hihi, S. and Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In *NIPS 8*. MIT Press.

Fiete, I. R. and Seung, H. S. (2006). Gradient learning in spiking neural networks by dynamic perturbations of conductances. *Physical Review Letters*, **97**(4).

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *AISTATS*.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *ICML'2013*.

Hinton, G. (2012). Neural networks for machine learning. Coursera, video lectures.

Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). Boltzmann machines: Constraint satisfaction networks that learn. Technical Report TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012a). ImageNet classification with deep convolutional neural networks. In *NIPS'2012*.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012b). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML'10*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Salakhutdinov, R. and Hinton, G. (2009). Semantic hashing. In *International Journal of Approximate Reasoning*.

Spall, J. C. (1992). Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, **37**, 332–341.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML 2008*.