Supplementary Material: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation

Anonymous Author(s)
Affiliation
Address
email

A Details of the Experiments

We frame our experiments using a conditional computation architecture. We limit ourselves to a simple architecture with 4 affine transforms. The output layer consists of an affine transform followed by softmax over the 10 MNIST classes. The input is sent to a gating subnetwork and to an experts subnetwork, as in Figure 1 of the main paper. There is one gating unit per expert unit, and the expert units are hidden units on the main path. Each gating unit has a possibly stochastic non-linearity (different under different algorithms evaluated here) applied on top of an affine transformation of the gater path hidden layer (400 tanh units that follow another affine transformation applied on the input). The gating non-linearity are either Noisy Rectifiers, Smooth Times Stochastic (STS), Stochastic Binary Neurons (SBN), Straight-through (ST) or (non-noisy) Rectifiers over 2000 units.

The expert hidden units are obtained through an affine transform without any non-linearity, which makes this part a simple linear transform of the inputs into 2000 expert hidden units. Together, the gater and expert form a conditional layer. These could be stacked to create deeper architectures, but this was not attempted here. In our case, we use but one conditional layer that takes its input from the vectorized 28x28 MNIST images. The output of the conditional layer is the element-wise multiplication of the (non-linear) output of the gater with the (linear) output of the expert. The idea of using a linear transformation for the expert is derived from an analogy over rectifiers which can be thought of as the product of a non-linear gater ($\mathbf{1}_{h_i>0}$) and a linear expert (h_i).

A.1 Sparsity Constraint

Computational efficiency is gained by imposing a sparsity constraint on the output of the gater. All experiments aim for an average sparsity of 10%, such that for 2000 expert hidden units we will only require computing approximately 200 of them in average. Theoretically, efficiency can be gained by only propagating the input activations to the selected expert units, and only using these to compute the network output. For imposing the sparsity constraint we use a KL-divergence criterion for sigmoids and an L1-norm criterion for rectifiers, where the amount of penalty is adapted to achieve the target level of average sparsity.

A sparsity target of $s = 0.1$ over units $i = 1...2000$, each such unit having a mean activation of p_i within a mini-batch of 32 propagations, yields the following KL-divergence training criterion:

$$\mathrm{KL}(s||p) = -\lambda \sum_i \left(s \log p_i + (1-s) \log 1 - p_i\right) \tag{1}$$

where λ is a hyper-parameter that can be optimized through cross-validation. In the case of rectifiers, we use an L1-norm training criteria:

$$\mathrm{L1}(p) = \lambda \sum |p_i| \tag{2}$$

In order to keep the effective sparsity s_e (the average proportion of non-zero gater activations in a batch) of the rectifier near the target sparsity of $s = 0.1$, λ is increased when $s_e > s + 0.01$, and reduced when $s_e < s - 0.01$. This simple approach was found to be effective at maintaining the desired sparsity.

A.2 Beta Noise

There is a tendency for the KL-divergence criterion to keep the sigmoids around the target sparsity of $s = 0.1$. This is not the kind of behavior desired from a gater, it indicates indecisiveness on the part of the gating units. What we would hope to see is each unit maintain a mean activation of 0.1 by producing sigmoidal values over 0.5 approximately 10% of the time. This would allow us to round sigmoid values to zero or one at test time.

In order to encourage this behavior, we introduce noise at the input of the sigmoid, as in the semantic hashing algorithm (**?**). However, since we impose a sparsity constraint of 0.1, we have found better results with noise sampled from a Beta distribution (which is skewed) instead of a Gaussian distribution. To do this we limit ourselves to hyper-optimizing the β parameter of the distribution, and make the α parameter a function of this β such that the mode of the distribution is equal to our sparsity target of 0.1. Finally, we scale the samples from the distribution such that the sigmoid of the mode is equal to 0.1. We have observed that in the case of the STS, introducing noise with a β of approximately 40.1 works bests. We considered values of 1.1, 2.6, 5.1, 10.1, 20.1, 40.1, 80.1 and 160.1. We also considered Gaussian noise and combinations thereof. We did not try to introduce noise into the SBN or ST sigmoids since they were observed to have good behavior in this regard.

A.3 Test-time Thresholds

Although injecting noise is useful during training, we found that better results could be obtained by using a deterministic computation at test time, thus reducing variance, in a spirit of dropout (**?**). Because of the sparsity constraint with a target of 0.1, simply thresholding sigmoids at 0.5 does not yield the right proportion of 0's. Instead, we optimized a threshold to achieve the target proportion of 1's (10%) when running in deterministic mode.

A.4 Hyperparameters

The noisy rectifier was found to work best with a standard deviation of 1.0 for the gaussian noise. The stochastic half of the Smooth Times Stochastic is helped by the addition of noise sampled from a beta distribution, as long as the mean of this distribution replaces it at test time. A beta of 40.1 was found to work best. The Stochastic Binary Neuron required a learning rate 100 times smaller for the gater (0.001) than for the main part (0.1). The Straight-Through approach worked best without multiplying the estimated gradient by the derivative of the sigmoid, i.e. estimating $\frac{\partial L}{\partial a_i}$ by $\frac{\partial L}{\partial h_i}$ where $h_i = \mathbf{1}_{z_i > \mathrm{sigmoid}(a_i)}$ instead of $\frac{\partial L}{\partial h_i}(1 - \mathrm{sigmoid}(a_i))\mathrm{sigmoid}(a_i)$. Unless indicated otherwise, we used a learning rate of 0.1 throughout the architecture.

We use momentum for STS, have found it have a negative effect for SBN, and have little to no effect on the ST and Noisy Rectifier units. In all cases we explored using hard constraints on the norms of incoming weights to the neurons. We found that imposing a maximum norm of 2 works best in most cases.