Reinforcement learning of conditional computation policies for neural networks

Emmanuel Bengio, Pierre-Luc Bacon, Ryan Lowe
Joelle Pineau, Doina Precup

June 23, 2016
Motivation

- Using RL to take decisions inside deep architectures
- Running high capacity models on low-end devices
  - Phones, mobile devices, electric wheelchair, etc.
- Large networks, fast evaluation?
  - sparse models
  - pruning, quantization
  - lazy/conditional evaluations
Conditional Computation

- Learn a *gater* function
  - which parts of the model to evaluate
  - which parts are useful
- Learn the main model concurrently
Conditional Computation with RL

- Build model with parameters $\theta$
- Divide parameters/computation into disjoint subsets of $\theta/H$
- Learn a *gating policy* $\pi_\omega(x)$ with separate parameters $\omega$
  - stochastic policy with binary actions (on/off)
  - one action per subset
  - state space is $x$
- Learn the main model($\theta$) concurrently
The REINFORCE estimator

\( k \)-Bernoulli policy:

\[
\sigma = \text{sigm}(W^{(\omega)} x + b^{(\omega)})
\]

\[
u_i \sim \text{Bern}(\sigma_i)
\]

\[
\pi(u \mid x) = \prod_{i=1}^{k} \sigma_i^{u_i} (1 - \sigma_i)^{(1-u_i)}
\]

\[
\nabla_{\omega} L = \sum_j (\text{cost}(x_j) - b) \nabla_{\omega} \log \pi_{\omega}(u_j \mid x_j)
\]

\( b \) is an exponential moving average of the costs.
Policy Regularization

- $L_b = \sum_{j}^{n} \| \mathbb{E}\{\sigma_j\} - \tau \|_2$
  each unit is $\tau$ in $\mathbb{E}$ over the $\text{xs}$

- $L_e = \mathbb{E}\{\| (\frac{1}{n} \sum_{j}^{n} \sigma_j) - \tau \|_2 \}$
  the mean of units is $\tau$ for some $x$

- $L_v = -\sum_{j}^{n} \text{var}_i\{\sigma_{ij}\}$
  encourage input-dependent units
Fully-Connected Architecture

color:red: policy

color:blue: network
Convolutional Architecture

red: policy
blue: network

All kinds of parametrizations are possible
Policies (fully connected, MNIST)
Policy Regularization

- \( L_b = \sum_j^n \| \mathbb{E}\{\sigma_j\} - \tau \|_2 \)
  each unit is \( \tau \) in \( \mathbb{E} \) over the \( \mathbf{x}s \)

- \( L_e = \mathbb{E}\{\| (\frac{1}{n} \sum_j^n \sigma_j) - \tau \|_2 \} \)
  the mean of units is \( \tau \) for some \( \mathbf{x} \)

- \( L_v = - \sum_j^n \text{var}_i \{ \sigma_{ij} \} \)
  encourage input-dependent units
Fully-Connected Results

- MNIST, CIFAR-10, SVHN
- Same or better accuracy than conventional NN
- up to $5 \times$ faster
- $>25-50 \times$ less computations

![Graph showing time of validation vs. valid error percentage for NN and condnet]
## Convnet Results

<table>
<thead>
<tr>
<th>model</th>
<th>test error</th>
<th>N</th>
<th>$\tau$</th>
<th>test time</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv-condnet</td>
<td>.157</td>
<td>4</td>
<td>0.5</td>
<td>1.03s</td>
</tr>
<tr>
<td>conv-condnet</td>
<td>.167</td>
<td>4</td>
<td>0.3</td>
<td>0.84s</td>
</tr>
<tr>
<td>conv-condnet</td>
<td>.176</td>
<td>4</td>
<td>0.2</td>
<td>0.66s</td>
</tr>
<tr>
<td>conv-condnet</td>
<td>.173</td>
<td>2</td>
<td>0.5</td>
<td><strong>0.58s</strong></td>
</tr>
<tr>
<td>conv-NN</td>
<td>.159</td>
<td>4</td>
<td>-</td>
<td>1.07s</td>
</tr>
</tbody>
</table>

CIFAR-10 results for conditional convnets
Conclusion

- It works!
- Similar accuracy, lower forward-pass time
- Much less computations being done
  (25% active nodes $\rightarrow$ ~6.25% computations, 10% $\rightarrow$ 1%)

- Does not scale very well (REINFORCE?)
- Hard to compete with dense computations
- Chicken and egg problem during learning