BranchyNet: Fast Inference via EarlyExitingfrom Deep Neural Networks

ICPR 2016

Surat Teerapittayanon
Brad McDanel
H. T. Kung

Harvard John A. Paulson School of Engineering and Applied Sciences
■ Motivation and Background
  ■ Trend towards deeper networks
  ■ Auxiliary network structures (GoogLeNet)

■ BranchyNet
  ■ Architecture
  ■ Training
  ■ Inference

■ Experimental Results
■ Future Work
■ Conclusion

BranchyNet with 3 exits
TREND TOWARDS DEEPER NETWORKS

ImageNet Classification top-5 error (%)
Accuracy vs. Depth (ILSVRC workshop - Kaiming He)
GoogLeNet introduces auxiliary networks
- Provide regularization to main network
- Improves accuracy $\approx 1\%$
- Removed after training
- Only main network is used during inference

Can we leverage auxiliary networks to address inference runtime of deeper networks?
Easier input samples require lower level features for correct classification

Harder input samples require higher level features

Use early exit branches (auxiliary networks) to classify easier samples
  - No computation performed at higher layers

Requires mechanism for determining network confidence about a sample to use exit

Jointly training the main and early exit branches improves the quality of lower branches
  - Allowing more samples to exit at earlier points
New sample enters the network
- New sample enters the network
- Reaches Exit 1
New sample enters the network
Reaches Exit 1
Determined “confident”
- New sample enters the network
- Reaches Exit 1
- Determined “confident”
- Classifies sample
- No additional work performed at upper layers
New sample enters the network

BRANCHYNET EXAMPLE: HARD SAMPLE
- New sample enters the network
- Reaches Exit 1
- New sample enters the network
- Reaches Exit 1
- Determined “not confident”
- New sample enters the network
- Reaches Exit 1
- Determined “not confident”
- Continues up the main network (no re-computation of lower layers)
- New sample enters the network
- Reaches Exit 1
- Determined “not confident”
- Continues up the main network (no re-computation of lower layers)
- Must exit (classify sample) as Exit 2 is final exit point
MEASURING NETWORK CONFIDENCE

- Use entropy of softmax output to measure confidence

\[ \text{entropy}(y) = \sum_{c \in C} y_c \log y_c, \]

where \( y \) is a vector containing computed probabilities for all possible class labels and \( C \) is a set of all possible labels

- Choice of entropy versus other measures

Exit 1 Softmax Output

![Entropy Graphs]

- Digit Class 0: Entropy 0.3747
- Digit Class 1: Entropy 1.3120
- Pretrain main network first
- Add exit branches and train again
- The final loss function is the weighted sum of losses of all exits

\[ L_{\text{branchynet}}(\hat{y}, y; \theta) = \sum_{n=1}^{N} w_n L(\hat{y}_{\text{exit}_n}, y; \theta), \]

where \( N \) is the total number of exit points

- Early exit weights \( W_{1..N-1} = 1 \)
- Last exit weight \( W_N = 0.3 \)
1: procedure BranchyNetFastInference(x, T)
2:         for n = 1..N do
3:             z = f_{exit_n}(x)
4:             \hat{y} = \text{softmax}(z)
5:             e = \text{entropy}(\hat{y})
6:         if e < T_n then
7:             return arg max \hat{y}
8:         return arg max \hat{y}

Figure: BranchyNet Fast Inference Algorithm. x is an input sample, T is a vector where the n-th entry T_n is the threshold for determining whether to exit a sample at the n-th exit point, and N is the number of exit points of the network.
Network Architectures
- LeNet (on MNIST)
- AlexNet (on CIFAR-10)
POINTS ON THE CURVE FOUND BY SWEEPING OVER VALUES OF $T$

- In the case of more than one early exit, we take combinations of $T_i$ values.

ACCURACY IMPROVEMENT OVER BASELINE NETWORK (RED DIAMOND) DUE TO JOINT TRAINING

RUNTIME IMPROVEMENTS OVER BASELINE NETWORK DUE TO CLASSIFYING THE MAJORITY OF SAMPLES AT EARLY EXIT POINTS (NO COMPUTATION PERFORMED FOR HIGHER LAYERS)

AS $T$ VALUES INCREASE, MORE SAMPLES EXIT AT THE HIGHER EXIT BRANCHES
FUTURE WORK

- Automatically find the threshold values $T$ for each exit branch
- Investigate alternative confidence measures other than softmax entropy (e.g., OpenMax, GANs)
- Dynamically adjusting the weight of loss based on individual samples
  - Easier samples have more weight at lower branches
  - Harder samples have more weight at higher branches
- Introduce a mechanism to exit a percentage of samples at earlier points in the network
- Jointly training these exit points improves accuracy which allows additional samples to exit early
- Achieve a factor of 2-4x speedup compared to baseline single network for our test case
- BranchyNet implementation written in Chainer and open source: https://gitlab.com/htkung/branchynet
Thanks for your attention!
Comments and Questions?
Table: Selected performance results for BranchyNet on the different network structures. The BrachyNet rows correspond to the knee points (denoted as green stars in the previous slides).

<table>
<thead>
<tr>
<th>Network</th>
<th>Acc. (%)</th>
<th>Time (ms)</th>
<th>Gain</th>
<th>Thrshld. T</th>
<th>Exit (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LeNet</td>
<td>99.20</td>
<td>3.37</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-LeNet</td>
<td>99.25</td>
<td>0.62</td>
<td>5.4x 0.025</td>
<td>94.3, 5.63</td>
<td></td>
</tr>
<tr>
<td>AlexNet</td>
<td>78.38</td>
<td>9.56</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-AlexNet</td>
<td>79.19</td>
<td>6.32</td>
<td>1.5x 0.0001, 0.0565.6, 25.2, 9.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LeNet</td>
<td>99.20</td>
<td>1.58</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-LeNet</td>
<td>99.25</td>
<td>0.34</td>
<td>4.7x 0.025</td>
<td>94.3, 5.63</td>
<td></td>
</tr>
<tr>
<td>AlexNet</td>
<td>78.38</td>
<td>3.15</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-AlexNet</td>
<td>79.19</td>
<td>1.30</td>
<td>2.4x 0.0001, 0.0565.6, 25.2, 9.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>