A The Imbalanced Tree Model Proofs

A.1 Proof of Theorem 5.1

Lemma A.1. *The abstract tree mode has polynomial runtime if and only if all critical nodes pick the right child.*

Proof of Lemma A.1. Necessity comes from the structure of the model and any picking of left child will immediately result in exponential search space. For sufficiency, by the assumption, the depth of the tree model is bounded by $O(\text{poly}(n))$, and the final sub-search space with all right child picking is also $O(\text{poly}(n))$. Then we have the total runtime is $O(\text{poly}(n)) + O(\text{poly}(n)) = O(\text{poly}(n))$. Q.E.D.

Proof of Theorem 5.1. By Lemma A.1, we have

$$\begin{aligned}\lim_{n\to\infty} \Pr(\text{model has poly runtime}) &= \lim_{n\to\infty} p^{\Theta(\log(\text{poly}(n)))} \\ &\leq \lim_{n\to\infty} p^{C\cdot\log(\text{poly}(n))} && \text{(for some constant } C) \\ &= 0 && (0 < p < 1 \text{ as assumption}).\end{aligned}$$

As a conclusion, $\lim_{n\to\infty} \Pr(\text{tree mode has exponential runtime}) = 1$. Q.E.D.

A.2 Proof of Theorem 5.2

Proof. We can assume there exists at least one non-critical node on the plan (if all tree nodes are critical, we can augment an extra non-critical node to the right child of the deepest critical node without affect other properties of the tree model). Let u be the shallowest non-critical node and d be the depth of node u. We set the restart time $t^{\mathcal{A}}$ to be the size of the left subtree of d. $t^{\mathcal{A}}$ is poly(n) by the definition of the model. Let $q = p^d(1-p)$. So the expected runtime

$$\begin{aligned}\lim_{n\to\infty} l^{\mathcal{A}} &= \lim_{n\to\infty} (d + O(\text{poly}(n))) \sum_{k=0}^{\infty} q \cdot (1-q)^k \cdot (k+1) \\ &= \lim_{n\to\infty} (d + O(\text{poly}(n))) \cdot \frac{1}{q} \\ &= O(\text{poly}(n)).\end{aligned}$$

By the result of Luby et al. [1993], $l^{\mathcal{A}}_{univ} = O(l^{\mathcal{A}} \log(l^{\mathcal{A}})) = O(\text{poly}(n))$. Q.E.D.

B Network Architecture and Training details

B.1 Network Architecture

A single input tensor of board states has shape $[4 \times H \times W]$ and a batch of board states can have different heights and widths. For each batch, we take the maximum H and W of all state tensors as the batch height and width, and zero-pad the empty cells.

The network consists of a single convolution block followed by 16 residual blocks.

The convolutional block applies the following modules:

1. A convolution of 128 filters of kernel size 3×3 with stride 1
2. 2D batch normalization [Ioffe and Szegedy, 2015]
3. A ReLU nonlinearity

Each residual block applies the following modules sequentially to its input:

1. A channel-wise dropout layer with probability 30% of a channel to be zeroed.
2. A convolution of 128 filters of kernel size 3×3 with stride 1

3. 2D batch normalization
4. A Relu nonlinearity
5. A channel-wise dropout layer with probability 30% of a channel to be zeroed.
6. A convolution of 128 filters of kernel size 3×3 with stride 1
7. 2D batch normalization
8. A skip connection that adds the input to the block
9. A Relu nonlinearity

The output of the residual tower is then fed into two independent heads for computing the policy and value. Both heads contain an extra residual block followed by a fully connected layer. The policy head outputs a vector of size 4 and the value head outputs a single scalar.

B.2 Training details

We use the AdamW optimizer [Loshchilov and Hutter, 2017] with weight decay 0.01 and an initial learning rate 0.001. We train the network with supervised training data for 200 epochs. The last 50 epochs use a learning rate of 0.0001. We set the batch size to 256. The whole training procedure took around 70 hours to finish on 5 Tesla V100 GPU cards.