Supplementary Note: Comparison of AlphaZero with Global-Pooling vs. AlphaZero

This supplement reports a sanity-check comparison between AlphaZero using Wu's global-pooling method ("AlphaZeroGP") and our baseline AlphaZero. AlphaZeroGP was trained jointly on Connect 4, Gomoku, and Othello. Note that AlphaZeroGP is a simplified test implementation and not a verbatim reimplementation of Wu's KataGo; we did not perform an exhaustive hyperparameter search or architecture tuning for the pooling heads.

Specifically, AlphaZeroGP's neural network architecture includes the following:

- To identify the game, one channel per game (spatial size H×W) is appended to the input stack, setting the channel for the current game to 1 and the others to 0.
- An initial convolutional layer is followed by a stack of 16 residual blocks (Res16). The baseline AlphaZero uses six residual blocks (Res6). AlphaZeroGP Res16 has 18.9M parameters, which is comparable to AlphaViT L8, AlphaViD L5, and AlphaVDA L5.
- The policy head applies a 1×1 convolution to produce spatial logits. In addition, it performs global pooling on a single-channel map and uses a linear layer to produce the logit for the "pass" action. Because the number of channels and pooled statistics are fixed, this head accommodates different board sizes.
- The value head averages convolutional feature maps into per-channel scalar values, followed by fully connected layers, and outputs a scalar evaluation of the board state. This averaging also allows flexible use across different board sizes.

Results and Analysis

In this supplement, AlphaZeroGP Res16 and AlphaZero Res6 denote AlphaZeroGP with 16 residual blocks and AlphaZero with 6 residual blocks. AlphaZeroGP Res16 performed worse than AlphaZero Res6 in the multi-game scenario (Connect 4, Gomoku, and Othello), as shown in Figure S1. This weaker performance may be attributable to the relatively simple design of the global-pooling policy head and the averaging-based value head. Because these heads reduce rich convolutional feature maps to simple statistics (e.g., global averages or maxima), they may fail to capture the diverse and complex patterns needed to discriminate between different game rules. In addition, the strong inductive bias

of convolutional (ResNet) architectures toward local spatial patterns likely hinders generalization across fundamentally different games.

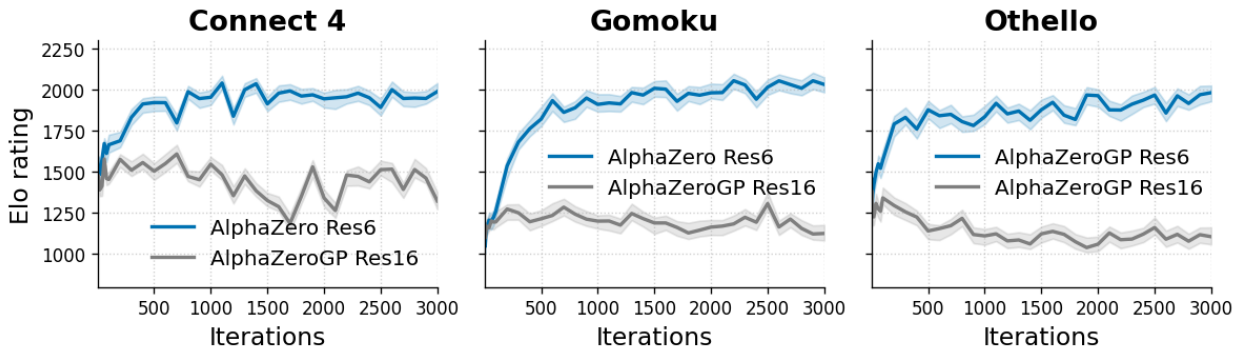

Figure S1. Elo rating progression over training iterations for AlphaZeroGP Res16 and AlphaZero Res6 on Connect 4, Gomoku, and Othello (horizontal axis: training iterations; vertical axis: Elo rating). The shaded bands show 95% confidence intervals estimated by bootstrapping.

Note

These results do not necessarily suggest a refutation of global pooling or Wu's KataGo. AlphaZeroGP is a simplified evaluation implementation, and we did not perform a systematic hyperparameter search or exhaustive architecture tuning for the pooling heads (i.e., there may be better ways to encode game identity or different residual-block hyperparameters such as depth and channel widths). Therefore, we do not draw definitive conclusions about the general effectiveness of convolutional networks with global pooling (or the KataGo architecture) from this experiment.

Code

```
class GlobalPool(nn.Module):

    """Aggregate (N,C,H,W) → (N,3C)  [mean, scaled-mean, max]"""

    def __init__(self, board_avg: float = 14.0):

        super().__init__()

        self.board_avg = board_avg


    def forward(self, x):

        N, C, H, W = x.shape
```

```
        mu    = x.mean(dim=(2, 3))                    # (N,C)
        mu_s = ((H - self.board_avg) / 10.0) * mu     # (N,C)
        mx    = x.amax(dim=(2, 3))                    # (N,C)
        return torch.cat([mu, mu_s, mx], dim=1)


class BasicBlock(nn.Module):
    def __init__(self, num_filters):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(num_filters, num_filters, 3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(num_filters)
        self.conv2 = nn.Conv2d(num_filters, num_filters, 3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(num_filters)

    def forward(self, x):
        r = x
        h = self.conv1(x)
        h = self.bn1(h)
        h = F.relu(h)
        h = self.conv2(h)
        h = self.bn2(h)
        h = h + r
        h = F.relu(h)

        return h

class Net(nn.Module):
    def __init__(self, params = Parameters()):
        self.params = params

        super(Net, self).__init__()
```

```
        self.conv1 = nn.Conv2d(self.params.input_channels, self.params.num_filters,
3, stride=1, padding=1)

        self.bn1   = nn.BatchNorm2d(self.params.num_filters)


        self.blocks = self._make_layer(self.params.num_res, self.params.num_filters)


        # policy head

        self.conv_p = nn.Conv2d(self.params.num_filters, self.params.num_filters_p,
1, stride=1)

        self.bn_p   = nn.BatchNorm2d(self.params.num_filters_p)

        self.conv_policy_logits = nn.Conv2d(self.params.num_filters_p, 1, 1,
bias=True)   # 1 logit / location

        self.gpool  = GlobalPool()

        self.pass_fc = nn.Linear(3 * 1, 1)  # 3*1 because we pool a single-channel
map below


        # value head

        self.conv_v = nn.Conv2d(self.params.num_filters, self.params.num_filters_v,
1, stride=1)

        self.bn_v   = nn.BatchNorm2d(self.params.num_filters_v)

        self.fc_v1 = nn.Linear(self.params.num_filters_v, 256)

        self.bn_v1 = nn.BatchNorm1d(256)

        self.fc_v2 = nn.Linear(256, 1)


    def forward(self, x, game_name):

        x = x.view(-1, self.params.k_boards * 2 + 1, self.params.board_x[game_name],
self.params.board_y[game_name])


        N, _, H, W = x.shape

        onehot = torch.zeros(

            (N, self.params.num_game_types, H, W),

            dtype=x.dtype,
```

```
        device=x.device
    )
    if game_name == "Connect4":
        onehot[:, 0, :, :] = 1.0
    elif game_name == "Gomoku":
        onehot[:, 1, :, :] = 1.0
    elif game_name == "Othello":
        onehot[:, 2, :, :] = 1.0
    else:
        raise ValueError(f"Unknown game name: {game_name}")

    x = torch.cat([x, onehot], dim=1)                    # (N, C0+G, H, W)

    h = F.relu(self.bn1(self.conv1(x)))

    h = self.blocks(h)

    N, _, H, W = h.shape

    # policy head
    h_p = F.relu(self.bn_p(self.conv_p(h)))
    spatial_logits = self.conv_policy_logits(h_p).view(N, -1)  # (N, H*W)

    # make 1-channel map to pool for pass bias (reuse logits)
    pass_bias_in = h_p[:, :1]               # (N,1,H,W)
    pooled = self.gpool(pass_bias_in)               # (N,3)
    # (N,3+d) → scalar
    pass_logits = self.pass_fc(pooled)  # (N,1)

    spatial_logits = torch.cat([spatial_logits, pass_logits], dim=1)  # +pass
```

```
        pi_logits      = spatial_logits[:, :self.params.action_size[game_name]]
        p = F.log_softmax(pi_logits, dim=1)


        # value head
        h_v = F.relu(self.bn_v(self.conv_v(h)))       # (N,V_ch,H,W)
        h_v = h_v.mean(dim=(2, 3))
        h_v = F.relu(self.bn_v1(self.fc_v1(h_v)))
        h_v = self.fc_v2(h_v)                         # (N,1)
        v = torch.tanh(h_v)


        return p, v


    def _make_layer(self, blocks, num_filters):
        layers = []
        for _ in range(blocks):
            layers.append(BasicBlock(num_filters))


        return nn.Sequential(*layers)
```