**面向国产可重构计算芯片的机载智能算法适配研究**

项目实施说明报告

单位名称（公章）：

目 录

一、项目名称 1

二、技术要求 1

（一）总体要求 1

（二）功能要求 1

（三）性能要求 1

三、项目进度 2

四、质量控制办法 2

五、项目实施计划 2

（一）机载智能算法研究 2

1、传统蒙特卡洛树搜索算法结构研究 2

2、使用蒙特卡洛树搜索的应用结构研究 4

3、基于蒙特卡洛树搜索的智能决策算法结构研究 10

（二）可重构架构的研究与适配场景 22

1、机载智能计算单元设计需求分析 22

2、可重构计算研究现状 25

（三）针对机载算法的异构可重构架构适配研究 32

1、蒙特卡洛树搜索结构适配性 32

2、支持乘加算子的多层级动态重构的计算架构 33

3、支持其他算子的多层级动态重构的计算架构 38

（四）机载智能计算单元研究方法及可行性分析 46

1、研究方法和技术路线 46

2、实验手段 46

六、交付技术状态 47

插图和附表清单

图 1 AlphaGo中的MCTS搜索示意图 5

图 2 MCTS的四个阶段（从左至右依次为搜索、扩展、模拟、反向传播） 7

图 3 AlphaGo中监督学习策略网络结构示意图 8

图 4 AlphaGo价值网络结构示意图 9

图 5 AlphaGo神经网络输入特征 9

图 6复杂搜索空间 12

图 7蒙特卡洛树搜索算法整体结构 15

图 8蒙特卡洛树搜索算法详细步骤的伪代码 16

图 9 AlphaGo中的策略网络和价值网络 18

图 10 AlphaGo中的神经网络组成 19

图 11 AlphaGo Zero中的自博弈过程 20

图 12 AlphaGo Zero中的改进 21

图 13空战智能决策算法中的网络结构 22

图14 几款具有代表性的可重构处理器：(a) ReMUS[6] (b) ADRES[3] (c) LSSD[7] (d) ParallelXL[10] 28

图15 几款面向AI的可重构计算架构：(a) FlexFlow[17] (b) SCALEDEEP[19] (c) Ganax[21] (d) Bit Fusion[20] 29

图 16 MCTS并行方法 32

图 17 正向推理和反向传播时过程示意图 34

图 18 面向神经网络运算的可重构计算阵列设计 35

图 19 配置信息的片上高速缓存结构 35

图 20 精度可配置乘加单元设计示意图 37

图 21 高能效近似乘加单元设计 37

图 22 异构编译流程图 39

图 23 支持灵活配置的可重构阵列和存储系统 40

图 24 自动化配置工具示意图 40

图 25 配置信息压缩示例 41

图 26 配置信息压缩示例 42

图 27 核心计算部分映射示例 44

图 28 分支控制结构优化流程 44

图 29 BN层处理流程 45

图 30 研究方法路线图 46

# 一、项目名称

面向国产可重构计算芯片的机载智能算法适配研究

# 二、技术要求

## （一）总体要求

面向国产可重构计算芯片的机载智能算法适配研究项目针对基于强化学习的智能任务决策算法在机载嵌入式环境中的高效运行需求，基于具备高性能功耗比的国产可重构计算芯片研究算法的适配优化技术，重点针对智能任务决策算法的神经网络计算特性研究可重构计算芯片适配方法、针对算法动态升级需求研究基于可重构技术的模型更新方法。

## （二）功能要求

* 面向国产可重构计算芯片的机载智能算法适配支持强化学习类智能决策算法的机载移植适配，提供面向智能决策算法的高性能神经网络智能计算资源；
* 面向国产可重构计算芯片的机载智能算法适配支持多种神经网络结构，包括GRU结构和Dense结构；
* 面向国产可重构计算芯片的机载智能算法适配支持多种神经网络算子，包括Const，MatMul，BiasAdd，Tanh，Mul，Add，Sigmoid，Split，Sub，ExpandDims，Shape，GreaterEqual，Cast，Less，Range，ListDiff，GatherV2，Prod，Pack，Transpose，Reshape，Maximum等；
* 面向国产可重构计算芯片的机载智能算法适配支持智能决策算法的参数动态可重构，支持参数的独立更新；
* 面向国产可重构计算芯片的机载智能算法适配支持智能决策算法的模型动态可重构，支持模型的静态/动态重构；
* 面向国产可重构计算芯片的机载智能算法适配支持智能决策算法的固化，支持上电后自动加载运行。

## （三）性能要求

* 面向国产可重构计算芯片的机载智能算法适配支持神经网络并行处理，处理能力不低于2TOPS/W，芯片功耗不高于1W；
* 面向国产可重构计算芯片的机载智能算法适配支持多种处理精度，包括INT8、FP16；
* 面向国产可重构计算芯片的机载智能算法适配对110\*512\*512\*512\*10规模的神经网络处理时间不大于1ms。

# 三、项目进度

以合同签订时间为T0，在合同周期内的研制进度要求如下：

T0+1个月：根据甲方提出的技术要求，乙方完成面向国产可重构计算芯片的机载智能算法适配方案设计；

T0+3个月：完成智能决策算法计算流程分析，完成基础算子设计与实现，通过中期评审；

T0+6个月：完成合同规定的全部研究内容，通过甲方的验收评审。

# 四、质量控制办法

* 乙方需编制项目研制方案，并对项目研制过程进行监督和协调；
* 乙方需根据研制内容各个节点要求，严格考核，确保合同要求的执行；
* 乙方需严格进行技术状态管理，对相关技术文件严格审查，确保技术文件可追溯性和现行有效性；
* 甲方组织专家开展设计评审和验收评审，对方案设计和成果进行严格把关；
* 乙方需组织各级设计人员进行加技术攻关、技术协调等工作；

# 五、项目实施计划

## （一）机载智能算法研究

### 1、传统蒙特卡洛树搜索算法结构研究

蒙特卡洛树搜索(Monte Carlo Tree Search)并不是一种"模拟人"的算法。而是通过随机的对游戏进行推演来逐渐建立一棵不对称的搜索树的过程。可以看成是某种意义上的强化学习，当然这一点学界还有一些争议。

蒙特卡洛树搜索大概可以被分成四步。选择(Selection)，拓展(Expansion)，模拟(Simulation)，反向传播(Backpropagation)。

MCTS的算法分为四步，第一步是节点挑选，就是在树中找到一个最好的值得探索的节点，一般策略是先选择未被探索的子节点，如果都探索过则选择UCB值最大的子节点。第二步是节点拓展，就是在前面选中的子节点中走一步创建一个新的子节点，一般策略是随机自行一个操作并且这个操作不能与前面的子节点重复，服从均匀分布。第三步是节点模拟，就是在前面新拓展出来的节点开始进行模拟，直到到达搜索结束状态，这样可以收到这个拓展出来的节点的得分是多少，该分值可通过网络进行计算。第四步是反向传播，就是把前面拓展出来的节点得分反馈到前面所有父节点中，更新这些节点的Q值（分值）和访问次数N，方便后面计算UCB值。

MCTS算法详细过程如下。

（1）节点挑选

对于挑选节点，选用Upper Confidence Bounds（UCB）的方法进行筛选，公式如下：

 (1)

其中为节点估计的值（比如评估的分值，或者胜率等等），*N*为该子节点的父节点被访问的次数，为该子节点被访问的次数。由公式可看出，若父节点被访问次数增多，或者该子节点被访问次数较少，或者子节点本身分值较高，则该子节点被访问的概率就会增大。*C*为可调参数*，C*越大就越偏向于广度搜索，*C*越小就越偏向于深度搜索。

从根节点开始，根据UCB算法算出每个子节点的分值，并进行排序，在我方行进阶段选择分值最大的子节点，在敌方行进阶段选择分值最小的子节点，直到底部没有子节点时停止。

如果一个节点下面还有未探索的子节点，那么先拓展下面的子节点就可以了，如果没有子节点，那么就用UCB算法来得到下一个子节点，然后便利下直到有未探索的节点可以探索。

（2）节点拓展

在节点选择阶段结束时，在选中的子节点中以均匀分布的规则创建若干子节点的过程为节点拓展

选择均匀分布的方式拓展子节点，如遇到已拓展的节点，则重新随机生成其他子节点，更新信息并返回。

（3）节点模拟

模拟是一个移动的序列，从当前节点开始，到终端节点结束。为了让子节点得到一个初始的评分。让游戏随机进行，直到得到一个游戏结局，这个结局将作为该子节点的初始评分。一般使用胜利/失败来作为评分，只有1或者0。该过程可用网络直接计算reword。

（4）反向传播

反向传播是从叶节点（模拟开始）到根节点的遍历。模拟结果被传送到根节点，并更新反向传播路径上每个节点的统计信息。反向传播保证每个节点的统计信息能够反映该节点所有后代的模拟结果。

反向传播模拟结果的动机是为了更新反向传播路径上所有节点v（包括模拟开始的节点）的总模拟收益Q(v)和总访问次数N(v)。

Q(v)：总模拟收益是节点v的一个属性，最简单形式的就是所有考虑的节点的模拟结果之和。

N(v)：总访问次数是节点v的另一个属性，代表这个节点有多少次出现在反向传播路径上。

每个访问过的节点都需要更新这两个量。如随机找一个节点，这个节点的统计信息反映了它多大可能是最佳下一步（总模拟收益），以及它被访问的频率（总访问次数）。收益高的节点是接下来探索的优秀候选节点，同时那些访问次数低的节点也是需要接下来继续探索的节点（因为它没有被探索完全）。把前面计算的这个分值反馈到“相关的”的节点上了，这个相关的意思是从根节点开始一直到这个拓展节点经过的所有节点，他们的分值和访问次数都需要更新。

（5）终止条件

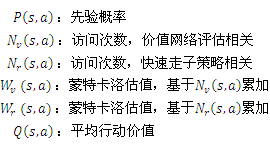
为了避免程序无限的搜索下次，需定义搜索终止条件，如搜索时间、搜索深度、迭代次数等。

MCTS算法主要解决Combinatorial Game的问题，在博弈论中，把zero-sum、perfect information、deterministic、discrete、sequential的游戏统称为Combinatorial Game，例如围棋、象棋等。

即该应用场景必须是能分出输赢（不能同时赢）、游戏的信息是完全公开的（不像打牌可以隐藏自己的手牌）、确定性的（每一个操作结果没有随机因素）、顺序的（操作都是按顺序执行的）、离散的（没有操作是一种连续值）。其中，信息对称（Perfect information / Fully information）是指游戏的所有信息和状态都是所有玩家都可以观察到的，因此双方的游戏策略只需要关注共同的状态即可，而信息不对称（Imperfect information / Partial information）就是玩家拥有只有自己可以观察到的状态，这样游戏决策时也需要考虑更多的因素。还有一个概念就是零和（zero-sum），就是所有玩家的收益之和为零，如果我赢就是你输，没有双赢或者双输，因此游戏决策时不需要考虑合作等因素。

### 2、使用蒙特卡洛树搜索的应用结构研究

AlphaGo将基于卷积神经网络的策略网络和价值网络与蒙特卡洛树搜索算法相结合，实现了一种异步的策略和价值MCTS算法（APV-MCTS）。搜索树中的每个节点s包含了若干的边*(s,a)*，其中a表示当前节点的一种合法动作。每个边*(s,a)*包含以下属性：



APV-MCTS算法执行时，在独立搜索线程上并行执行多个模拟过程，每个过程同样分为四个阶段，下图所示：

搜索（a）：每个模拟过程的第一阶段，开始于搜索树的根节点，终止于叶节点，记终止时刻为L。对于之前的任一时刻 t < L，根据以下公式选择动作：

|  |  |
| --- | --- |
|  |  |

公式中使用了PUCT算法的一个变体，，其中是个常量。该搜索控制策略初始倾向于先验概率高的走子，后期将倾向于平均行动价值高的走子。



扩展（b）：当访问次数超出一定阈值，即*Nr*(*s*,*a*)>*nthr*时，将新状态*s’*=*f*(*s,a*)加入搜索树中。新参数中访问次数和蒙特卡洛估值均初始化为0，先验概率*P*(*s,a*)由策略网络*Pα* (•*|s’*)计算得到。通过动态调整阈值*nthr*，使新节点的添加速率和GPU的计算速率相匹配。



评估（c）：叶节点在首次访问时会被加入一个评估队列，等待价值网络*Vθ*(*sL*)的计算。同时模拟过程的第二阶段——快速走子将会从叶节点开始，直至游戏结束。对于任一时刻t>=L ,使用快速走子策略为双方选择动作。最后根据游戏比分计算模拟结果*zt*。为了最小化计算时间，设定策略网络和价值网络mini-batch=1。



回溯（d）：对于搜索树中的每一个时刻*t*<=*L*，为快速走子策略相关属性添加一个虚拟损失，假装某节点失败了*nv1*场比赛，从而避免其他并行线程探索相同路径。在模拟过程结束后，恢复虚拟损失，并回溯路径上的每个节点，根据模拟结果更新快速走子策略相关属性。另外，当价值网络对叶节点的评估结束时，开启一个新的回溯过程，更新价值网络相关属性。每次更新后都要根据公式（2）重新计算平均行动价值



|  |  |
| --- | --- |
|  |  |

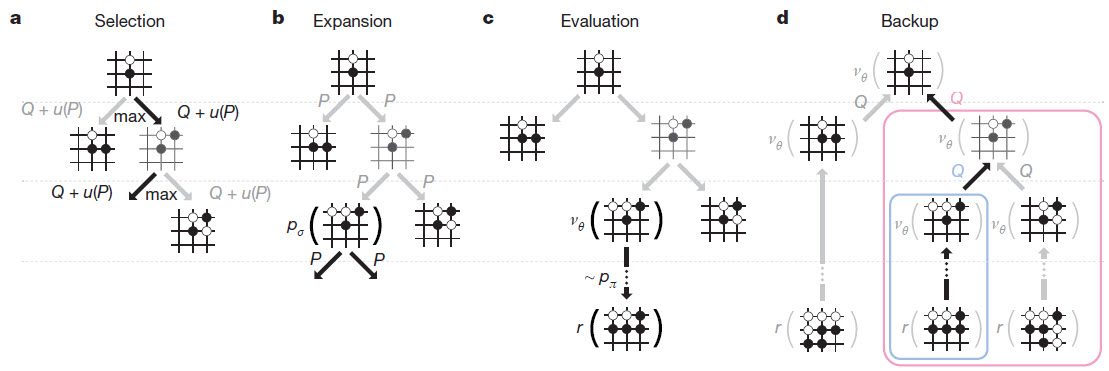


图 1 AlphaGo中的MCTS搜索示意图

经过若干次模拟过程后，AlphaGo选择访问次数最高的节点作为对局中的实际走子动作，这样可以减少意外情况对走子的影响。下一时刻，将搜索树中被选中的子节点作为新的根节点，保留子树的全部信息并舍弃搜索树的其他部分。在对手下棋过程中，AlphaGo会持续进行模拟过程，不断的扩大搜索树，并使估值更准确。此外，当某个节点的max(*Q*(*s,a*))<-0.8时，AlphaGo会放弃此次模拟，判定本次模拟过程的结果是失败。

AlphaGo算法的两大核心组件是蒙特卡洛树和神经网络。

#### 2.1 AlphaGo中的蒙特卡洛树搜索

蒙特卡洛树搜索（Monte-Carlo Tree Search）是一种启发式的搜索策略，能够基于对搜素空间的随机抽样来扩大搜索树，从而分析围棋这类完美信息博弈游戏中每一步应该采取哪个动作才能够创造最好的机会。它结合了随机模拟的一般性和树搜索的准确性。需要说明的是，蒙特卡洛树搜索并不是只有一种算法，而是一类算法。其中最流行的算法之一就是UCT（upper confidence bounds for trees）算法，根据多次模拟的统计规律，选择胜率最高的节点作为下一步动作。

MCTS包括四个阶段：选择、拓展、模拟和反向传播。如下图所示。

（1）选择（Selection）

从根节点出发，检查每个节点，选择出一个急需被拓展的节点。

对于被检查的节点，有三种可能：

1. 该节点所有可行动作都已经被拓展；
2. 该节点有可行动作未被拓展。
3. 该节点游戏已经结束（例如已经连成五子的五子棋局面）。

对于这三种可能，分别采取以下操作：

1. 对于所有子节点都已经被拓展的节点，基于UCB公式，选择最大节点作为新的根节点继续进行检查；
2. 如果存在未被拓展的子节点，选择一个还未拓展的动作A，执行阶段2；
3. 游戏已经结束，则执行阶段4。

（2）拓展（Expansion）

根据阶段1中选择的动作A，游戏到达新的状态Snew，将新节点Snew添加到搜索树中，之后执行阶段3。

（3）模拟（Simulation）

为了使新节点得到一个初始评分，从节点Snew开始让游戏随机进行，直到得到一个游戏结局，以此作为Snew的初始评分，之后执行阶段4。

（4）反向传播（Backpropagation）

从整个树的根节点到当前节点路径上的全部节点，都要根据当前节点的评分进行更新，将新的评分添加到各自的累积评分中。



图 2 MCTS的四个阶段（从左至右依次为搜索、扩展、模拟、反向传播）

#### 2.2 AlphaGo中的神经网络

（1）快速走子网络

AlphaGo的快速走子策略网络是一个线性softmax策略，基于能够使棋盘达到状态*s*的'response' pattern和关于状态*s*可能采取的动作*a*的'non-response' pattern进行计算。其中'response' pattern是一个围绕到达状态*s*的动作的菱形区域，包含12个点的棋子颜色、自由度等信息；'non-response' pattern是一个以动作*a*为中心的3×3区域。使用从人类对局中选出的800万个位置数据（棋面+下一步动作）对快速走子策略网络进行训练，通过随机梯度下降最大化对数似然函数值。面对一个空白棋盘，网络可以在每个CPU线程上达到每秒完成1000次模拟过程。



（2）监督学习策略网络

AlphaGo监督学习策略网络*Pσ*是一个13层的卷积神经网络，使用KGS数据集中挑选的2940万个位置数据进行训练和测试。对于每轮训练，设置mini-batch=m，从训练数据集中随机抽取，通过随机梯度上升使人类走子的对数似然函数最大化，即：



|  |  |
| --- | --- |
|  |  |

步长初始值为0.003，每经过8千万次的训练的值减半。





图 3 AlphaGo中监督学习策略网络结构示意图

（3）强化学习策略网络

AlphaGo中，强化学习策略网络*Pρ*基于策略网络*Pσ*进行训练。每次迭代同时进行n场对局，当前网络*Pρ*的对手记为*Pρ-*，其参数*ρ-*从自博弈对手池中随机选取，以避免过度拟合。初次迭代时，*ρ=ρ-=σ*，每500轮迭代后，将当前网络参数加入对手池。每次对局结束后，分别从两个选手的角度评估对战结果，并复盘比赛计算参数调整梯度：



|  |  |
| --- | --- |
|  |  |

基准*v*(*sit*)由价值网络计算得到，在第一轮训练中初始化为0。通过强化学习训练，策略网络的输出将更加倾向于赢得比赛而不是预测人类棋手的走子。

（4）价值网络

对于围棋这类完美信息博弈游戏，理想情况是可以通过一局基于完美策略的对局模拟，选择下一步的动作。但现实是没人拥有完美策略，于是用策略网络来代替。为了解决策略网络模拟时间过长的问题，AlphaGo建立价值网络，使。

为了避免过度拟合，人工构建若干对局。首先，随机选取一个时刻；然后在时刻*t*=1,2,…,*U*-1，使用监督学习策略网络*Pσ*选择各时刻的动作，在时刻*U*，从棋盘的合法位置随机选取一个作为动作*aU*,在时刻*t*=*U*+1,…,*T*,,使用强化学习策略网络选择各时刻的动作；最后根据上述对弈情况判断对局结果。每局比赛仅有（*sU*+1,*zU*+1）加入数据集用于训练。



价值网络的结构跟策略网络基本相同，仅多了两个全连接层，如图4所示。价值网络通过随机梯度下降使预测价值更加接近实际观测价值：

|  |  |
| --- | --- |
|  |  |



图 4 AlphaGo价值网络结构示意图

（5） 网络参数

所有的棋盘状态都被处理成19×19的特征平面，原始的棋盘信息依据围棋规则进行预处理，得到表征棋子颜色、自由度、捕获等不同信息的特征平面。所有特征表现为选手相关，即当前选手棋子和对手棋子，而不是黑棋和白棋。另外，特征平面的值为整数的，被拆分成若干个二进制平面，例如用8个独立的特征平面分别表示1自由度、2自由度、……、不小于8自由度。网络参数如图10所示，策略网络只是用上半部分48个特征平面，价值网络使用全部的49个特征平面。

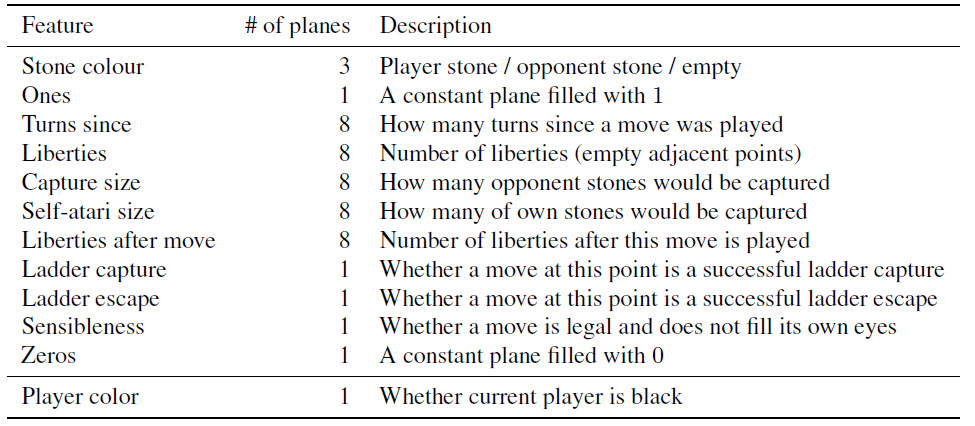


图 5 AlphaGo神经网络输入特征

### 3、基于蒙特卡洛树搜索的智能决策算法结构研究

针对空战中我机的智能决策问题，在双机空战的场景，需要在空战过程中对作战态势进行分析，选择消灭对方保存自己的攻防策略，设计优势函数，以赢取最大空战优势期望为目标，建立空战双方机动决策模型，并引入蒙特卡洛树搜索方法和强化学习方法对问题进行求解，以保证我机在空战中取得优势占位，取得空战博弈中的最终胜利。

#### 3.1空战智能决策建模

空战智能决策的实质就是在当前战场环境下，选择合适的基本机动动作，使得空战优势函数达到最大，从而赢得空战的胜利。研究空战智能决策问题需要对空战智能决策问题进行建模。

（1）基本机动动作库

空战机动决策集合主要分为“典型战术动作决策集模型（典型战术动作库）”与“基本机动动作决策集模型（基本机动动作库）”。设空战过程中机动动作状态空间为E={*E1*，*E2*，…,*E9*}，分别对应9种飞行状态。整个空战过程就是敌我合理选择基本机动动作，所以整个空战过程可以描述为一个多叉树上从根节点到叶节点的一条路径。空战就是敌我双方依次在空战多叉树上进行机动动作选择。

选择相应的机动动作之后，需要根据机动动作对应的控制量U={φ, u, n}求解出作战飞机的状态量X＝｛*x*，*y*，*z*，*v*，*γ*，*ψ*｝，其中*φ*表示滚转角，*u* 表示油门位置，*n*表示过载，*x*、*y*、*z*的空间坐标，*v*表示速度，*γ*、*ψ*表示航迹倾角和航迹偏角。

式中：*x*、*y*、*z*表示速度在3个坐标轴上的分量。则联立作战飞机运动学方程：

因此,可根据作战飞机控制量*U*得到其状态量*X*。

（2）空战优势函数

对作战飞机机动动作的选择通过依据当前空战态势构建对应的空战优势函数来完成，使我机对敌机的空战优势达到最大。针对空战优势函数，对角度优势函数、距离优势函数、速度变化优势函数、高度优势函数建模，将这４种函数值加权求和得到空战优势函数。

1. 角度优势函数

角度优势描述了两机的相对方位，综合考虑目标角和进入角，得到角度优势函数*Sa*。

1. 距离优势函数

相对距离在空战过程中是一个十分重要的参数。我机距离优势函数的计算，需要考虑包括最大攻击距离、最佳攻击距离（不可逃逸区）和最小攻击距离在内的机载武器攻击包线，得到距离优势函数*Sr*

1. 速度变化优势函数

对于近距空战而言，使用速度变化进行衡量更有意义，速度变化越激烈的战机越不容易被敌方瞄准打击，需要考虑过载，设计速度变化优势函数*Sv*。

1. 高度优势函数

导弹射程随两机之间的高低位置而受影响：两机携带有同型导弹，位于高位的飞机，其导弹的最大发射距离要大于位于低位的飞机。并且，位于高位的飞机可以利用高度优势进行俯冲，快速进入攻击区并退出战斗。因此依据相对高度设计高度优势函数*Sh*。

综上，得到空战决策过程的优势函数为：

*S*=*W1Sa*+*W2Sr*+*W3Sv*+*W4Sh*

式中： *W1*,*W2*,*W3*,*W4*分别为４种优势函数在空战优势中的权系数，要求ΣW*i*=1, 在不同作战态势下，空战优势函数各参数的重要程度不同。

（3）空战优势

根据空战优势函数，空战态势与进入角、视线角、相对距离、过载、相对高度等有关。若当前时刻为*t*决策时刻，则*t*决策时刻对应的空战态势为*O*(*t*)；相应的，根据空战态势*O*(*t*)，可以得到此空战态势下的空战优势*S*(*t*)；机动动作的状态空间为*E*={*E1，E2，…，E9*}，通过智能决策算法在第*t*决策时刻选择的机动动作*A*(*t*)。

假设已知*t-*1决策时刻的空战态势*X*(*t-1*)以及机动动作*A*(*t-*1)，现在需要对*t*决策时刻的机动动作进行决策。传统方法主要以试探法为主，在*t*决策时刻分别选取不同的机动动作，估计相应的空战态势，得到空战优势*S*(*t*)，选择使得空战优势*S*(*t*)最大的机动动作*Ei*作为该时刻的机动动作。

由于求解未来*ｋ*步所有决策过程的空战优势期望，将需要罗列9k种可能动作，随着探索步数*k*的增加，使用穷举方法不现实。空战机动动作选择是通过试探未来的机动动作来确定下一步决策的空战优势概率。蒙特卡洛树搜索（Monte Carlo Tree Search, MCTS）是一种平衡历史收益和未来开发的树搜索策略。其基本原理是：首先随机选择机动动作策略，然后通过预期收益来更新原来选择策略的价值，设定随机策略选择的概率，与先前计算出的策略价值成正比，通过大量反复模拟，让最好的策略出现。因此，使用蒙特卡洛树搜索的方式来进行求解。

#### 3.2蒙特卡洛树搜索在智能决策中应用

蒙特卡洛树搜索（Monte Carlo Tree Search，MCTS），是一类树搜索算法的统称，可以较为有效地解决一些探索空间巨大的问题，例如AlphaGo用的是蒙特卡洛树搜索算法的一种，并且一般的围棋算法都是基于MCTS实现的。

3.2.1蒙特卡洛树搜索在智能空战中应用分析

AlphaGo在人工智能领域取得了划时代意义的成绩，围棋智能几乎长期被人类视为人工智能挑战象牙塔上的明珠，2015年3月，AlphaGo以4比1的悬殊比分战胜了韩国围棋大师李世石，而AlphaGo用的正是蒙特卡洛树搜索算法在围棋算法中的应用，相对于传统蒙特卡洛树搜索算法，在计算树节点的价值中增加了神经网络。

（1）场景分析

当树的层数比较浅的时候，可以穷举计算每个节点输赢的概率，基本思路是这样的，从树的叶子节点开始，如果是是本方回合就选择max的，如果是对方回合就是min的，实际上这也是假设对方是聪明的也会使用min-max算法这样在博弈论里面就达到一个纳什均衡点。

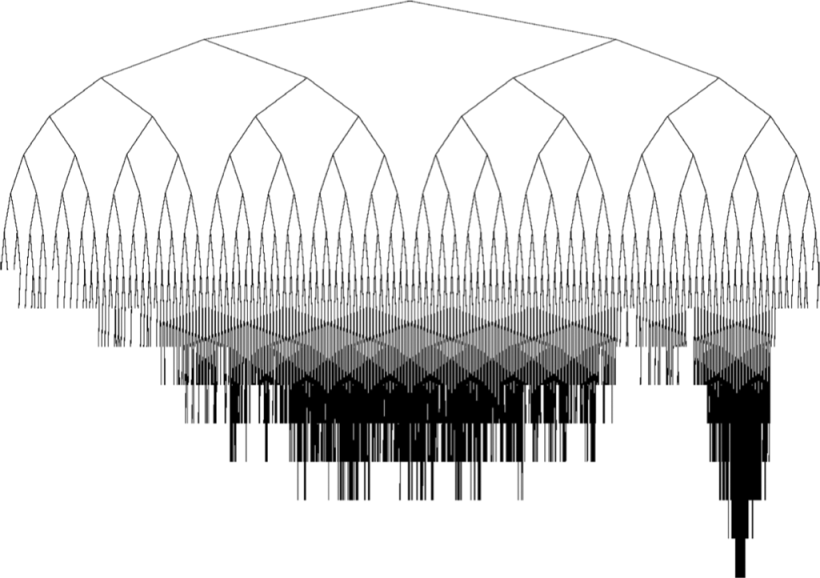


图 6复杂搜索空间

搜索空间比较小的时候的可以使用min-max策略，当搜索空间变大时就不在适用如图6所示。以围棋为例，蒙特卡洛树搜索算法要解决的问题是这样的，把围棋的每一步所有可能选择都作为树的节点，第零层只有1个根节点，第1层就有361种下子可能和节点，第2层有360种下子可能和节点，这是一颗非常大的树，如图6所示，要在每一层树节点中搜索出赢概率最大的节点，也就是下子方法。MCTS要解决的问题是搜索空间足够大，不能计算得到所有子树的价值，这是需要一种较为高效的搜索策略，同时也得兼顾探索和利用，避免陷入局部最优解。MCTS实现这些特性的方式有多种，例如经典的UCB（Upper Confidence Bounds）算法，就是在选择子节点的时候优先考虑没有探索过的，如果都探索过就根据得分来选择，得分不仅是由这个子节点最终赢的概率来，而且与这个子节点玩的次数成负相关，也就是说这个子节点如果平均得分高就约有可能选中（因为认为它比其他节点更值得利用），同时如果子节点选中次数较多则下次不太会选中（因为其他节点选择次数少更值得探索），因此MCTS根据配置探索和利用不同的权重，可以实现比随机或者其他策略更有启发式的方法。

蒙特卡洛树搜索是一种基于树数据结构、能权衡探索与利用、在搜索空间巨大仍然有效的搜索算法。蒙特卡洛树搜索的使用场景需要是搜索空间巨大，因为搜索空间如果在计算能力以内，其实是没必要用MCTS的，而真正要应用上还需要有其他的假设：场景必须是零和（能分出输赢不能同时赢）、信息对称（双方的信息是完全公开的，不像打牌可以隐藏自己的手牌）、确定性的（每一个操作结果没有随机因素）、顺序的（操作都是按顺序执行的）、离散的（没有操作是一种连续值）。所谓零和，就是参与者的收益之和为零，如果我赢就是你输，没有双赢或者双输，因此决策时不需要考虑合作等因素。信息对称是指双方的所有信息和状态都是对方都可以观察到的，因此双方的策略只需要关注共同的状态即可。在双方多轮博弈后，双方会选择一个纳什均衡点，对于MCTS适用于特定的场景，MCTS也是一种找到逼近纳什均衡的搜索策略。

对于空战场景，敌我双方都试图占据优势态势，击败对方，更不存在合作的可能，因此空战场景是一场零和博弈；敌我双方的机动动作和状态信息的都可以被双方的机载探测设备获取，各机在获取自身状态的同时，执行机动动作时必然要考虑对方的机动动作和状态信息，实际上，双方在决策机动动作时都同时考虑了一个空战态势，因此双方的对战场景是信息对称的；飞机在执行机动动作和火控解算时，当飞控指令或火控指令下达时，飞机的飞控系统或火控系统会严格执行相应的指令，动作的执行是确定性的，不存在随机性；飞机的机动作是根据执行严格按指令下达的顺序执行，不会存在当前动作未执行完，而执行下一条指令；在空战智能决策中建立了机动动作库，各个动作之间彼此是相互的独立，机动动作编号转换为机动指令，因此各个机动动作是离散的。通过以上分析可知，通过对空战智能决策场景的建模，将空战的决策离散为9种基本机动动作，而空战场景是一场敌我双方信息对称的零和博弈，因此，蒙特卡洛算法完全适用于空战场景。

（2）黑盒优化

所谓优化就是根据给定的数据集找到更好的选择，例如机器学习就是典型的优化过程，但一些机器学习算法如LR、SVM、DNN都不是黑盒，而是根据数学公式推导通过对函数求导等方式进行的优化。如果能把问题描述成一个函数或者凸优化问题，那么通过数学上的求导就可以找到最优解，这类问题并不需要用到MCTS等搜索算法，但实际上很多问题例如围棋就无法找到一个描述输赢的函数曲线，这样就无法通过纯数学的方法解决。

无法找到一个描述输赢的函数曲线这类问题统称为黑盒优化（Black box optimization）问题，不能假设知道这个场景内部的函数或者模型结构，只能通过给定模型输入得到模型输出结果来优化。例如多臂老虎机（Multi-arm Bandit）问题，有多台老虎机可以投币，但不知道每一台输赢的概率，只能通过多次投币来测试，根据观察的结果预估每台机器的收益分布，最终选择认为收益最大的，这种方法一般会比随机方法效果好。

黑盒优化的算法也有很多，例如进化算法、贝叶斯优化、MCTS也是，而这些算法都需要解决如何权衡探索和利用（Exploration and Exploitation）的问题。如果只有一个投币，那么当前会选择期望收益最高的老虎机来投（Exploitation），但如果有一万个投币，我们不应该只投一个老虎机，而应该用少量币来探索一下其他老虎机（Exploration），说不定能跳过局部最优解。

对于空战问题，进行空战智能决策建模后，实际上将其转化为了如何根据当前的空战态势，在机动动作库中选择最为合理的机动动作，使得我机在空战中取得优势函数，从而赢得空战。在决策时，不能只根据当前的状态决策机动动作，而是进行多步探索，考虑当前的决策对将来的影响。若进行9步探索，将最多有99中机动动作组合，这是非常大的搜索空间，而对于一个树状的搜索结构，难以进行函数建模，或者将其转化为凸优化问题，因此对于文中的空战智能决策问题是一个黑盒优化问题。需要说明的是，一般黑盒优化问题，需要进行大量的探索、模拟，例如进化算法、贝叶斯优化，对于蒙特卡洛树搜索算法，虽然在节点的探索和利用上做了权衡，对于满树的搜索空间进行了缩减，但本质上依然是黑盒优化问题，对智能计算单元的算力有一定的要求。

3.2.2蒙特卡洛树搜索算法分析

通过前文分析可知，蒙特卡洛树搜索是一种基于树数据结构、能权衡探索与利用、在搜索空间巨大仍然比较有效的搜索算法。蒙特卡洛树搜索算法主要包括选择（Selection）、扩展（Expansion）、模拟（Simulation）和反向传播（Backpropagation）四个阶段。

（1）UCB算法

蒙特卡洛树搜索算法中通过UCB（Upper Confidence Bounds）算法来权衡子节点的探索和利用（exploitation和exploration），具体原则是在选择子节点的时候优先考虑没有探索过的子节点，如果都探索过就根据得分来选择，因此MCTS根据UCB算法来探索和利用不同的权重，可以实现比随机或者其他策略更有启发式的方法。UCB算法如公式XX所示。

其中v'表示当前树节点，v表示父节点，Q表示这个树节点的累计quality值，N表示这个树节点的visit次数，C是一个常量参数（可以控制exploitation和exploration权重）。

这个公式的意思是，对每一个节点求一个值用于后面的选择，这个值有两部分组成，左边是这个节点的平均收益值（越高表示这个节点期望收益好，越值得选择，用于exploitation），右边的变量是这个父节点的总访问次数除以子节点的访问次数（如果子节点访问次数越少则值越大，越值得选择，用于exploration），因此使用这个公式是可以兼顾探索和利用的。

（2）算法原理

蒙特卡洛树的基本思路就是这样的，通过不断的模拟得到大部分节点的UCB值，然后下次模拟的时候根据UCB值有策略得选择值得利用和值得探索的节点继续模拟，在搜索空间巨大并且计算能力有限的情况下，这种启发式搜索能更集中地、更大概率找到一些更好的节点，可以分为4个步骤包括选择（Selection）、扩展（Expansion）、模拟（Simulation）和反向传播（Backpropagation）：

1. 选择（Selection）

Selection就是在树中找到一个最好的值得探索的节点，一般策略是先选择未被探索的子节点，如果都探索过就选择UCB值最大的子节点。

1. 扩展（Expansion）

Expansion就是在前面选中的子节点中走一步创建一个新的子节点，一般策略是随机自行一个操作并且这个操作不能与前面的子节点重复。

1. 模拟（Simulation）

Simulation就是在前面新Expansion出来的节点开始模拟游戏，直到到达游戏结束状态，这样可以收到这个Expansion出来的节点的得分是多少。

1. 反向传播（Backpropagation）

第四步是Backpropagation，就是把前面expansion出来的节点得分反馈到前面所有父节点中，更新这些节点的quality value和visit times，方便后面计算UCB值。

蒙特卡洛树搜索算法整体结构的伪代码如下图所示。

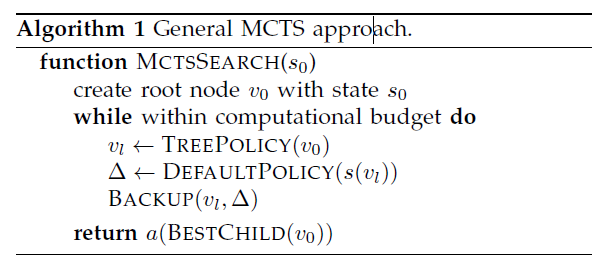


图 7蒙特卡洛树搜索算法整体结构

其中，TREE\_POLICY就是实现了Selection和和Expansion两个阶段，DEFAULT\_POLICY实现了Simulation阶段，BACKUP实现了Backpropagation阶段，基本思路和上文描述的步骤相同。

蒙特卡洛树搜索算法详细步骤的伪代码如下图所示。

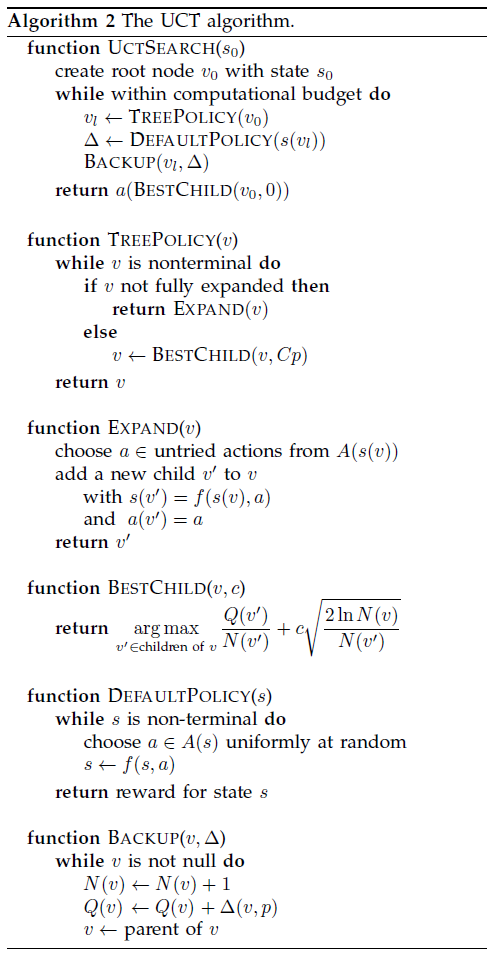


图 8蒙特卡洛树搜索算法详细步骤的伪代码

为了实现蒙特卡洛树搜索算法的功能，对于各个节点的数据结构，应包含父节点和子节点属性，还以及用于计算UCB值的访问次数visit times（N）和各个节点的UCB值quality value（Q值）。为了关联各个节点状态，需要为每个节点绑定一个状态对象。各个需要实现增加节点、删除节点等功能，还有需要提供函数判断子节点的个数和是否有空闲的子节点位置。

对于TREE\_POLICY其实现了Selection和和Expansion两个阶段，检查如果一个节点下面还有未探索的子节点，那么先expansion下面的子节点就可以了，如果没有未探索子节点（即该节点的所有的子节点均已展开），那么就用BESTCHILD函数（其实也就是UCB算法）来得到下一个子节点，然后便利下直到有未探索的节点可以探索。

对于BESTCHILD函数就是根据节点的状态获取quality value（Q值）和visit times（N），然后计算出UCB值，然后比较返回UCB值最大的子节点。

EXPAND函数实现稍微复杂，实际上是在当前节点下，选择一个未被执行的动作（action）来执行即可，策略可以随机选，如果有随机选中之前已经执行过的则重新选。

因此,TREE\_POLICY方法就是根据是否有未探索子节点和UCB值（也就是权衡和exploration和exploitation）后选出了expansion节点，然后就是用DEFAULT\_POLICY来模拟剩下的游戏了。

对于DEFAULT\_POLICY方法，可以调用随机策略，一直玩到最后得到一个reward值即可，当然对于AlphaGo等其他游戏可能需要实现一个更优的快速走子网络实现类似的功能。

对于BACKUP实现了把前面DEFAULT\_POLICY方法计算的reward反馈到“相关的”的节点上，这个相关的意思是从根节点开始一直到这个expansion节点经过的所有节点，这些节点的quality value（Q值）和visit times（N）都需要更新。

（2）空战智能决策算法对蒙特卡洛树算法的改进

对于基于UCB的蒙特卡洛树搜索算法，空战智能决策算法在比照AlphaGo的做法下，对于基于UCB的蒙特卡洛树搜索算法进行了改进。

首先，AlphaGo每个节点可选动作太多了，selection阶段不能像前面先遍历完所有子节点再expansion，这里是用改进的UCB算法来找到最优的需要expansion子节点，算法基本类似也是有控制exploration/exploitation的常量C并且与该子节点visit times成反比。而在空战智能决策算法中，设计的机动动作库中有9种基本动作，每个节点依然可以遍历所有的子节点。

其次，进行expansion时不会像前面这样直接random选择任意的action，而是这里也考虑到exploration/exploitation，一般前30步根据visit times成正比来选择，这样可以尽可能得先探索（根节点加入了狄利克雷分布保证所有点都经过），后面主要是根据visit times来走了。这一点AlphaGo和空战智能决策算法都使用。

第三，新版AlphaGo Zero去掉了基于handcraft规则的rollout policy（即DEFAULT\_POLICY），也就是快速走子网络，以前是必须有快速走子直到完成比赛才能得到反馈，现在是直接基于神经网络计算预估的winer value概率值，然后平均得到每个子节点的动作状态value也就是Q值。这一点在空战智能决策算法中仿照了AlphaGo Zero的做法，通过神经网络来评估各个选择节点的Q值，从而不必进行多步随机仿真模拟，才能给出选择节点的得分。

#### 3.3 空战智能决策中的神经网络结构

**3.3.1 AlphaGo中的神经网络分析**

围棋由于它巨大的搜索空间并且很难评估棋盘中落子的位置和决定下一步的落子，长期以来被视为人工智能处理经典游戏中最大的挑战。AlphaGo阐述了一种新的方法去处理围棋，这种新的方法使用“价值网络”去评估棋盘中已落子的位置，“策略网络”去选择下次落子的位置，如图9

所示。AlphaGo中有两条法则可以有效地减少搜索空间的大小。一是搜索树的深度太深可以用价值网络评估来解决；二是搜索树宽度太宽的问题可以用策略网络来解决，它是基于状态而得出动作的一个概率分布。

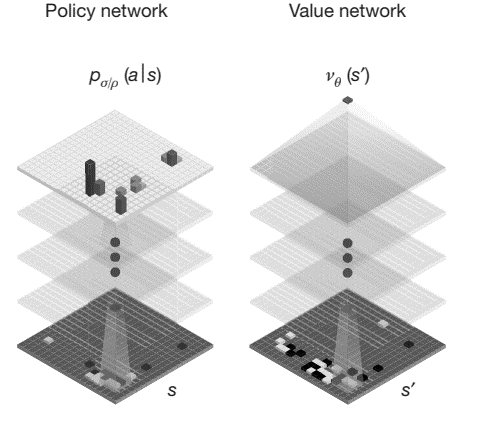


图 9 AlphaGo中的策略网络和价值网络

AlphaGo中的神经网络由四部分组成，分别为SL policy network、Rollout policy、SL policy network、Value network,如下图所示。

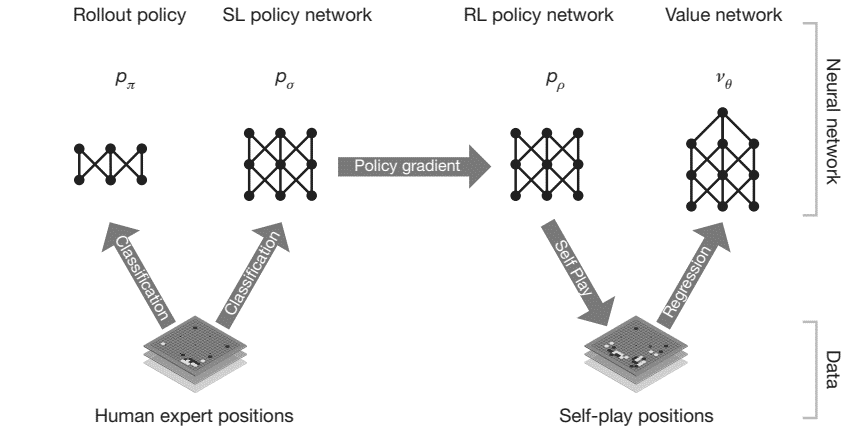


图 10 AlphaGo中的神经网络组成

1. SL policy network

用人类棋手里面的对弈记录进行训练，输入为48张图片，以输出为人类棋手的落子为目标，进行训练。SL policy network采用卷积神经网络和ReLu激活函数，最后一层使用softmax输出落子概率。输入策略网络中的状态是一个棋盘状态的表达。

1. Rollout policy

将很复杂的卷积神经网络去掉就能够得到一个快速走子网络。它与SL policy network的功能一样，两者间的不同是：Rollout policy网络简单，落子更快，准确率较低。rollout policy是一个网络结构简单的神经网络，也是使用softmax作为输出，能够达到24.2%的精确度，决策一步棋仅需2 μs，而SL网络需要3 ms。这个网络主要用于后面蒙特卡洛树的模拟仿真。

1. RL policy network

最开始由SL policy network复制而来，两者开始对弈，在对弈的结果里面我们依据最终的胜负来修正权重。优化之后再更新对手的权重，再接着对弈，两者共同优化。RL policy network的网络结构是和SL policy network的网络结构一模一样，并且RL网络权重的初始化是拷贝SL网络的参数。使其自我博弈，随着自我博弈的进行，依据最终是否胜利RL网络就会慢慢进化。RL policy network随机选择之前版本的自己（RL policy network）进行对弈。用这种从对手池中选择对手的方式能够更稳定地训练，防止过拟合。最终胜利的话，奖励为+1，输了的话，奖励为-1。之前也说过，这种自我对弈出来的RL policy network其实就具备很高的围棋水平了。

1. Value network

用RL policy network自我对弈得到的棋局数据来训练价值网络。输入是一个棋面，输出是这个棋面的胜率。使用RL policy network强大的策略网络来评估价值函数。价值网络的架构与策略网络的架构是非常相似的，只是将其输出变为一个单一的值，而不是策略网络中的动作分布概率。用回归的方法训练神经网络，以最小化均方差做随机梯度下降。误差来自价值网络的预测的输出和这盘局相应的最终奖励输出。为了防止过拟合等问题，他们从强化学习自我博弈的不同的棋局里面选出30万把不同位置的棋盘来做训练。

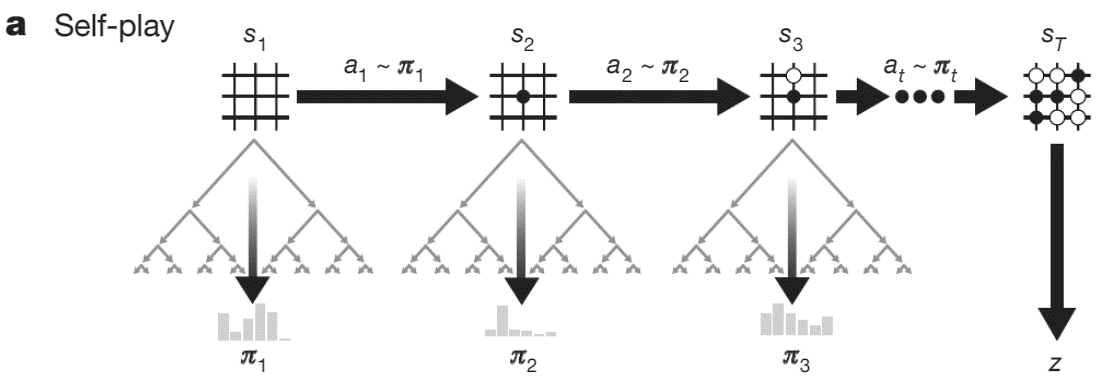
AlphaGo中的树搜索使用深度神经网络来评估棋局和选择动作。这些神经网络是利用人类专家的动作通过监督学习训练而来，然后通过强化学习来进行自我对弈。这里，引入了一种完全独立的强化学习算法，不需要采用人类数据或是基于游戏规则的引导或领域知识。AlphaGo成为自己的老师：通过训练神经网络来完成AlphaGo的动作选择的预测和对弈的胜者。这个网络还提高了树搜索的能力，使得能够在下一手中有更高质量的落子选择和更强的自我对弈能力。通过自我博弈，不断进化的程序是AlphaGo Zero，如下图所示。

图 11 AlphaGo Zero中的自博弈过程

与AlphaGo不同的是：

第一、完全独立地采用自我对弈强化学习的方法来训练，从刚开始的随机博弈开始就没有任何的监督或使用人工数据。

第二、只采用棋盘上的黑白走子作为输入特征。

第三、用了单个神经网络，而不是分离的策略和估值网络。

第四、采用了只依赖一种神经网络的更简单的树搜索策略，来评估棋盘和简单的走子，而不是用是采用蒙特卡洛快速走子。

AlphaGo Zero的神经网络使用强化学习通过自我博弈得到的。每一个盘面状态s，按照神经网络fθ的引导执行一次MCTS，如下图所示。MCTS搜索输出每一步可行棋的概率π。。MCTS搜索得出的概率所选择的棋步通常比直接用神经网络算出的概率p所选的棋步强；MCTS可以看作是一个强大的策略提升算子。带使用改进的基于MCTS策略来选择棋步，然后将胜者作为价值的一个样本，可以看做是一个强大的策略评估算子。强化学习算法的主要想法就是在策略迭代过程中重复使用这些搜索算子：神经网络参数的更新使得其所预测的可行棋概率分布和盘面价值更接近用蒙特卡洛方法提升过后的概率分布和盘面价值(π,z);这些新参数用于下一轮自我博弈使得搜索也更强。

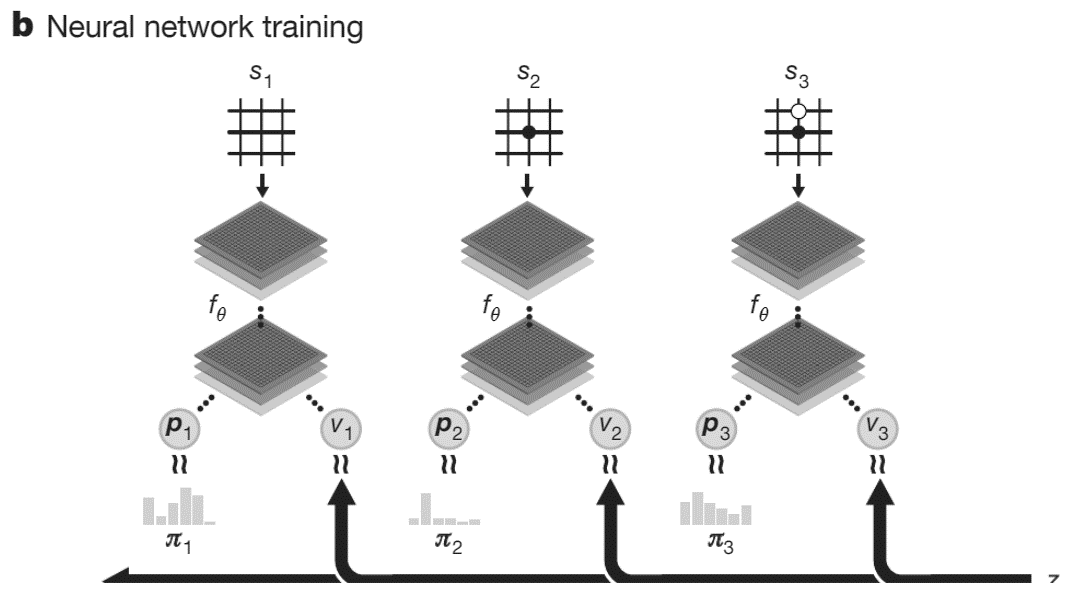


图 12 AlphaGo Zero中的改进

**3.3.2空战智能决策的神经网络分析**

对于空战智能决策算法向智能计算单元的移植，不需要考虑空战智能决策算法的训练网络过程，只需考虑前向推理网络过程。由于在空战智能决策的建模过程中，建立了机动动作库，将飞机的机动动作离散化为9种基本动作，也就是说空战智能决策的结果将在9种基本动作中产生。相对于围棋棋面，最多可能有361种可能的落子位置，空战智能决策的动作选择空间较小，且可选取机动动作数量在每一决策步中固定。因此相对于AlphaGo中的网络结构有如下简化：

1. 无需考虑训练过程

AlphaGo中的SL policy network用人类棋手里面的对弈记录进行训练，而在空战智能决策算法向智能计算单元的移植过程中，只需要考虑前向推理，不需要考虑训练过程。

1. 不需要考虑策略网络

AlphaGo中的RL policy network用于决策蒙特卡洛树中分支中的落子位置，在空战智能决策算法中，只有9种基本动作，搜索树中各个节点只有9个叶子节点，节点数较少，可以完全展开。

1. 无需快速走子网络

在AlphaGo Zero中，通过价值网络来评估棋盘和简单的走子，而不是用是采用蒙特卡洛快速走子。空战智能决策算法也采用AlphaGo Zero中的策略，用价值网络Value network替代快速走子网络Rollout policy。

**3.3.3空战智能决策算法中的网络结构**

空战智能决策算法中的价值网络（Value network）结构是深度神经网络（Deep Neural Networks，DNN），包括4个全连接层，输入层和两个隐藏层的激活函数为ReLu激活函数，输出层的激活函数为Softmax,其网络结构如下图所示。

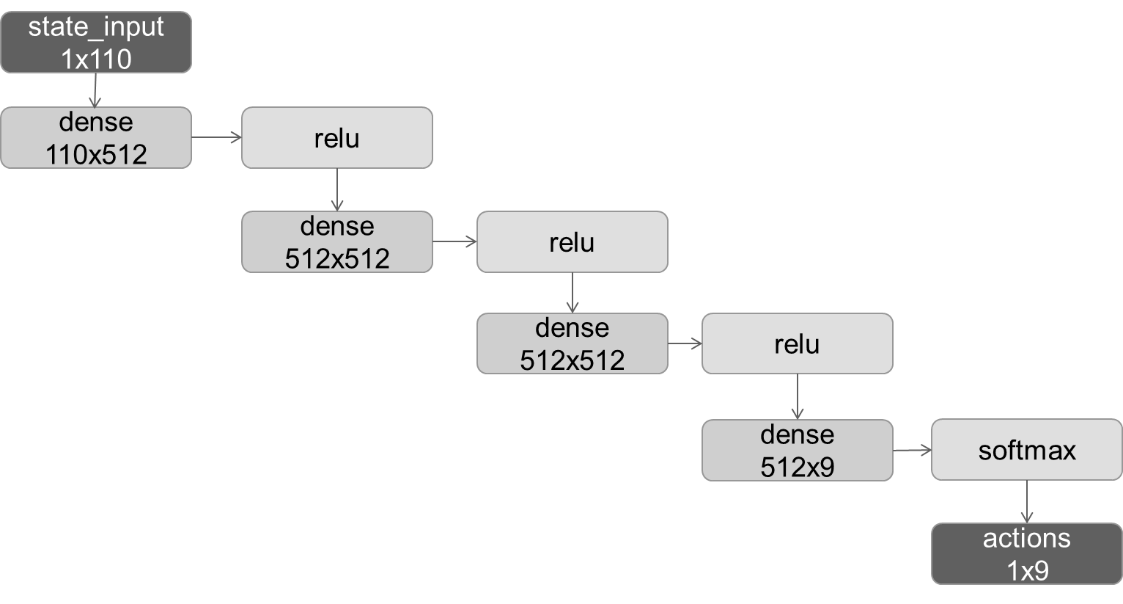


图 13空战智能决策算法中的网络结构

## （二）可重构架构的研究与适配场景

### 1、机载智能计算单元设计需求分析

设计需求首先分析蒙特卡洛树搜索算法的具体运算需求。以AlphaGo算法为例，AlphaGo算法的计算主要包括蒙特卡洛树搜索和神经网络模型计算，而DeepMind团队在用AlphaGo算法和人类旗手以及其他棋类算法比赛时，使用的主要的处理器为CPU、GPU以及Google自主研发的TPU处理器。因此可以根据算法运行时使用的处理器规模衡量算法复杂度。

截止目前，DeepMind团队先后在Nature上发表了两篇文章，分别是最初版的AlphaGo和最新版的AlphaGo Zero。但是DeepMind团队对AlphaGo的更新是逐步进行的，根据AlphaGo的战绩，可以分为四个版本。

（1）AlphaGo Fan，AlphaGo的第一个公开版本，因为在2015年10月以5:0的比分战胜欧洲围棋冠军樊麾而得名。该版本AlphaGo使用48个CPU和8个GPU进行计算，分40个搜索线程；同时，DeepMind团队尝试了分布式结构，使用1202个CPU和176个GPU进行计算，同样分为40个搜索线程。经测试，分布式结构的棋力仅有微小提升。

（2）AlphaGo Lee，非公开版本，2016年3月以4:1的比分打败世界围棋冠军李世石。根据相关信息，该版本使用分布式结构，基于48个TPU进行计算而不是GPU。

（3）AlphaGo Master，2017年1月和人类顶尖高手对战取得了60:0的战绩。该版本初步将策略网络和价值网络合并为一个网络，仅使用一台设备，4个TPU进行计算。根据Google最初公开的数据，TPU的计算性能是GPU的15倍以上。

（4）AlphaGo Zero，2017年11月，以100:0打败AlphaGo Lee, 89:11打败AlphaGo Master。该版本更进一步简化了输入特征信息，同样使用一台设备，4个TPU进行计算。但是在训练过程中，使用了60个GPU进行计算，及19个CPU作为参数服务器。

从上面4个版本的算法对比可以看出，如果使用CPU和GPU进行算法计算，顺利运行算法需要50个左右CPU处理器和10个左右的GPU处理器。而在算法训练中，一般需要几十个GPU处理器。算法描述中没有具体给出所用的CPU和GPU处理器型号，但是根据对相关论文的调研和研究，在利用GPU进行神经网络训练时，通常使用Nvidia的GTX系列GPU，该系列GPU专为大型服务器而设，其功耗非常大，单个GPU的运行时功耗通常在100W以上。

通过AlphaGo算法运行环境配置，可知蒙特卡洛树搜索算法的运算需求，主要涵盖支持MCTS搜索过程的运算需求，以及神经网络处理部分的运算需求。

（1）MCTS搜索过程运算需求

MCTS搜索过程是较为典型的决策树运算过程。决策树算法的特点是根据不同的条件进行判断和分支选择，并且有非常多的条件规则和分支选择规则。在机载应用中，决策树算法的树生成过程和树剪枝过程，即决策树的学习过程，通常在地面服务器中进行，只将优化后的树形结构移植到机载环境中执行。但在某些情况下，同样需要在机载计算系统中实时进行决策树算法的动态生成和调整。

决策树结构可以和if-then规则对应，对于if-then形式的规则，体现在处理器中的操作包括：

1）从存储空间（外部存储器或者内存）中获取规则条件；

2）根据规则条件，从规则数据库中搜索相应规则；

3）依据规则条件，执行逻辑判断，按照规则结果进行分支选择；

因此，决策树算法运行时的计算需求涉及两个方面，一方面是处理器执行数据读写和执行分支跳转类指令的能力；一方面是计算机中的数据库管理能力，反映在硬件的存储器性能和操作系统的数据库系统或文件系统支持。

其中，处理器执行能力，依赖于处理器架构，并受编译器能力影响。处理器的处理能力一方面在很大程度上依赖流水线处理技术，将指令处理流水化，以充分利用运算器和存储器的资源，加快处理速度。而在执行跳转类指令时，由于跳转类指令会破坏流水线，导致流水中断。另一方面，跳转类指令往往需要跨地址进行数据寻址和指令寻址，会破坏存储器中指令和数据的时间局部性和空间局部性，导致取指和取数据延迟增加。因此，处理器的指令处理瓶颈，往往在跳转类指令。处理器执行跳转类指令的能力，是影响决策类算法计算速度的重要指标。除处理器的执行能力外，编译器对决策类算法的计算效率也有影响。决策类算法通常采用高级语言实现，编译器将其编译为机器语言，编译过程中对高级程序语言的优化方式、优化水平，影响生成的机器语言的执行效率，最终反映在算法的指令规模和处理器执行各条指令的效率。

CPU处理器架构核心在于设置了运算器、控制器和存储器，利用专门的控制器控制指令的执行，同时设置了大量的数据寄存器和状态寄存器进行指令的控制和运行。因此，CPU架构特性加上指令集的设置，使其能够较好的支持控制类指令的执行。

另一方面，决策类算法通常需要构建决策树，或者生成if-then规则来存储整个决策算法，而决策树自身的结构复杂度，或者由决策树生成的规则的规模，都非常高。在决策类算法执行过程中，根据每一步的输入条件，需要从众多的分支或规则中搜索满足条件的规则。因此，计算系统对决策树的存储和管理能力，以及对大量规则的管理能力，直接影响每一步的决策效率。具体的，若用树形结构存储决策树，则内存空间大小能否满足决策树的存储规模需求，是影响决策树搜索速度的最大限制。若内存空间不足，则需将决策树的树形结构存储至外部存储器，而外部存储器的访存速度相比于内存，平均会下降一至两个数量级。因此，从内存空间的指标，决策树算法对计算系统的存储空间，尤其是内部存储空间的能力需求较大。若用规则形式存储决策树，则需用文件系统或数据库系统存储和管理众多的规则，规则的存储和管理方式，以及规则查询方法，直接影响规则搜索的效率。因此，计算系统是否具备高效的数据库管理能力，是决策规则能否快速查找的关键。

综上所述，决策树类算法对计算系统的能力需求，主要包括两个方面：计算能力，尤其是处理大量复杂的分支跳转类控制指令的能力；存储能力，尤其是对大量离散化数据的高效管理能力。

（2）神经网络运算需求

神经网络算法的两大特性即计算密集性和存储密集性。计算密集性体现在神经网络算法的三个方面：（1）计算规模十分庞大，如卷积神经网络算法中较为简单的AlexNet算法，其乘加运算的次数就有几十亿之多；（2）计算指令占整个算法指令数的比例非常高，在神经网络算法中，基于神经网络的简单且标准的网络模型和结构，算法的绝大部分操作都是卷积运算，而卷积运算仅包括乘法和加法运算。（3）深度神经网络算法在执行前向计算时，计算指令分布密集，由深度神经网络的结构，决定了网络的较低层都是标准卷积运算，中间夹杂的诸如dropout等概率选择，也可以转换为乘法运算；同时网络的较高层可能会有一层或多层全连接运算，其运算核心也是乘法和加法。在整个网络进行前向过程中，几乎没有控制类指令。因此，神经网络算法的典型计算密集性特性，使得该类算法十分适合利用具备大规模并行结构的处理器实现。

神经网络算法的存储密集性体现在算法模型的规模。一方面，通过对多种不同应用和不同架构的深度神经网络算法分析可知，深度神经网络算法模型的规模普遍较大，较简单的网络模型，其参数数量都在百万以上，而在计算机中存储时，若采用浮点数存储，则需百兆以上的存储空间。而深度神经网络算法复杂度不断增加，其模型规模必将不断增大，这对计算系统的存储资源提出了很大的需求。另一方面，通过对深度神经网络算法执行过程分析可知，深度神经网络算法执行中，需要频繁访问存储空间以获取卷积核、特征图等矩阵形式的数据，同时需要将每一步运算中产生的新特征图以及更新的卷积核参数写回至存储空间中。因此在网络运行过程中，处理器与存储器之间将会进行频繁且巨量的数据传输，这对处理器内部数据总线的带宽，提出了非常高的需求。

综上所述，神经网络算法对计算系统的能力需求，主要包括两个方面：计算能力，尤其是处理大规模并行化的乘加运算的能力；存储能力，尤其是高速存储空间大小以及处理器与存储器之间的数据总线带宽。

基于上述分析，机载智能计算单元，需要设计高性能CPU主控单元来支持MCTS搜索过程的处理，并设置面向神经网络运算加速的专用协处理器来支持神经网络的快速处理。综合考虑已有技术成熟度，选用高性能CPU主控，多核CGRA联协处理神经网络的方式，进行机载智能计算单元设计。

### 2、可重构计算研究现状

#### 2.1、可重构计算研究意义

近年来，人工智能技术已在各类民用场景中发挥着愈发重要的作用，被广泛应用于机器视觉、语音识别、自动驾驶等各个领域。信息化和智能化也正在重塑未来战争的形态，军事智能成为竞争的焦点。为适应智能化战争的需求，重大装备（如各型地基雷达、机载雷达、无人作战平台等）需要具备高灵敏、高性能、强实时、智能化的信息处理能力，以面对现代战场环境的高度复杂性、动态性和不确定性，不断获得情报信息。这就要求核心信息处理芯片必须兼具高密度的智能计算能力，具备高能效、可编程、高灵活、自适应等特点。

高密度计算与人工智能算法各具特点。有一些神经网络算法运算规整、对计算精度要求高，数据速率高、数据量大，一般采用16位/32位定点计算；也有些算法在数据处理中，算法灵活多变、动态范围较大，需要采用单精度和双精度浮点计算。而人工智能算法具有内生容错性，权重模型普遍采用8位定点，有些场景下甚至可以被压缩为1位；神经网络模型具有较大冗余性，通过剪枝等方法处理后，运算不再规整。而另一方面，神经网络与智能算法又紧密联系，在一个完整的人工智能应用中，这二者缺一不可。以蒙特卡洛树搜索算法为例，节点拓展与节点选择评估就有着明显的差别，节点拓展的运算较为复杂但是并行度的要求并不高，而节点选择评估需要用到神经网络算法，有着极为大量的卷积操作，但是操作却又单一，只有乘加操作，这就要求硬件计算平台必须兼顾这两类差异明显的计算任务。此外，由于军事环境具有边界不确定性和信息不完整性的特点，人工智能算法的训练数据集必然无法充分反映应用环境的情况，这就需要信息系统具有自适应、自学习的能力，通过应用过程中的增量学习，提高针对实际环境的信息处理效果。这就要求硬件计算平台能够支持人工智能算法的在线训练。

回顾近三十年信息处理芯片的发展历史，其性能得益于集成电路工艺的进步和计算架构的演进。当前，集成电路制造工艺已进入“后摩尔时代”，工艺进步带来的性能提升已非常有限，探索新型计算架构成为唯一选择。近年来人工智能芯片的发展历程以充分说明了这一趋势。针对神经网络算法，传统的CPU器件(如Intel Xeon E5)能效只有0.2~1GOPS/W，而采用可重构架构的神经网络计算芯片（如清华大学Thinker芯片）的能效则可达到TOPS/W量级。

面向重大装备的复杂数据处理和智能计算需求，研究动态可重构的多模态智能处理器，将能够提升处理器件效能和硅资源利用率，解决支持领域单一的问题，有助于实现重大装备的网络化、小型化、一体化和智能化。

根据设计特点分析，机载智能计算单元采用由高性能粗粒度可重构计算阵列集成得到的可重构处理器作为协处理器提供神经网络加速处理功能。

#### 2.2、国内外研究现状

传统的以冯·诺依曼架构为基础的通用处理器具有极强的灵活性，但是其性能和功耗并不理想。相对的，专用计算执行速度快，功耗低。但代价是灵活性和扩展性很差。高密度神经网络算法的计算量和数据量都十分庞大，而且处于快速演进之中。使用通用处理器或者完全定制化的专用电路都没有办法同时满足性能、能效和灵活性要求。可重构计算（Reconfigurable Computing）将软件的灵活性和硬件的高效性结合在一起，在性能、功耗和灵活性等关键指标之间取得更好的平衡，因此更适合处理这些任务。

在2000年前后，国际上可重构计算的研究出现了一次热潮，大量可重构计算器件被设计和应用，如PACT公司的XPP-III[1]、IPFlex公司的DAPDNA-2[2]、IMEC的ADRES处理器[3]等。在国内，西北工大、中科大和清华大学等也同步开展可重构计算架构的研究[4][5][6]。这些研究充分证明了可重构架构在高密度计算领域的优越性。2015年，《国际半导体技术路线图》（ITRS报告）将粗粒度可重构架构视为未来最有希望的可编程计算架构。2017年，美国国防部高级研究计划局（DARPA）发起了“电子复兴计划”，大力布局“软件定义硬件”研究计划，研究可重构架构。目前，动态可重构架构已经成为处理器体系结构领域最受关注的研究热点。近年 ISCA、MICRO、HPCA等体系结构顶级会议上相关研究工作层出不穷[7][8][9][10]。例如，威斯康辛大学麦迪逊分校和高通公司联合提出的LSSD架构[7]能同时支持近似计算、卷积、流处理、深度学习等多种领域，并且取得接近每个领域专用加速器的性能和能效；斯坦福大学提出的Plasticine架构[8]支持并行编程模型Parallel Patterns，提升了可重构架构的可编程性；以色列理工学院提出的可重构架构[9]支持计算阵列在线程内交互，避免了不必要的读写存储器操作；康奈尔大学提出的ParallelXL架构[10]支持动态任务生成和调度，并实现了动态负载均衡。这些工作体现了可重构架构在跨领域应用、多任务并行和多模态处理上的巨大潜力。

随着基于神经网络的人工智能算法的崛起，通用处理器不具备相适应的性能和能效，神经网络计算芯片的研究出现了井喷[11][12][13][14][15][16][17][18][19][20][21][22]。其中，中科院计算所开发的DianNao[11][12][13][14]系列是属于指令驱动的专用处理器（Application-Specific Instruction set Processor，ASIP），其计算单元配置非常灵活，但是每一步操作都需要精确的指令来控制存储器访问和具体的计算单元的操作类型，导致架构与片外存储交互频繁，使得性能和能效往往受限于总线带宽。随后，空间计算架构逐渐成为神经网络加速器的主流。这是由于空间计算架构提供了大量计算资源和丰富的互连，以及相对简洁的控制机制，非常适合神经网络的计算模式，如麻省理工的Eyeriss[15]，比利时鲁汶大学的Envision[16]。在空间计算架构之后，神经网络算法的高速演进又推动了神经网络计算可重构化的趋势。为了适应日新月异的网络拓扑结构、网络层类型和网络压缩技术，ISCA、HPCA等顶级会议上有大量的神经网络计算工作均采用可重构空间计算架构。其中，FlexFlow[17]和SCNN[18]采用可变的数据流，来适应不同网络的负载和稀疏度，以减少访存，提高能效；Bit fusion[20]采用bit brick技术来实现计算单元位宽可变能力；Ganax[21]有效结合SIMD和MIMD两种工作模式，取得了比Eyeriss更高的性能和能效；SCALEDEEP[19]和HyPar[22]则同时支持神经网络推理和训练。

虽然现有工作已经具备远超通用处理器的能效和高于专用计算器件的灵活度，但是，对于树搜索和神经网络融合的任务来说，仍然有关键性不足。首先，多模态处理能力不足，无法同时适应节点搜索和神经网络的不同位宽计算，以及张量类型数据和时间序列数据同时处理等需求。第二，该领域大部分工作还不具备成熟的片上学习能力。第三，在编译技术上，目前还没有面向高密度计算与人工智能融合的统一编译方法。因此，本课题将设计一种高密度计算和人工智能融合共存的多模态处理器架构，并开发配套的异构编译方法。该架构支持资源复用和多层级动态重构，通过异构的计算阵列，满足不同位宽和计算类型的需求，同时实现树搜索算法、神经网络推理和训练的高效执行。



图14 几款具有代表性的可重构处理器：(a) ReMUS[6] (b) ADRES[3] (c) LSSD[7] (d) ParallelXL[10]

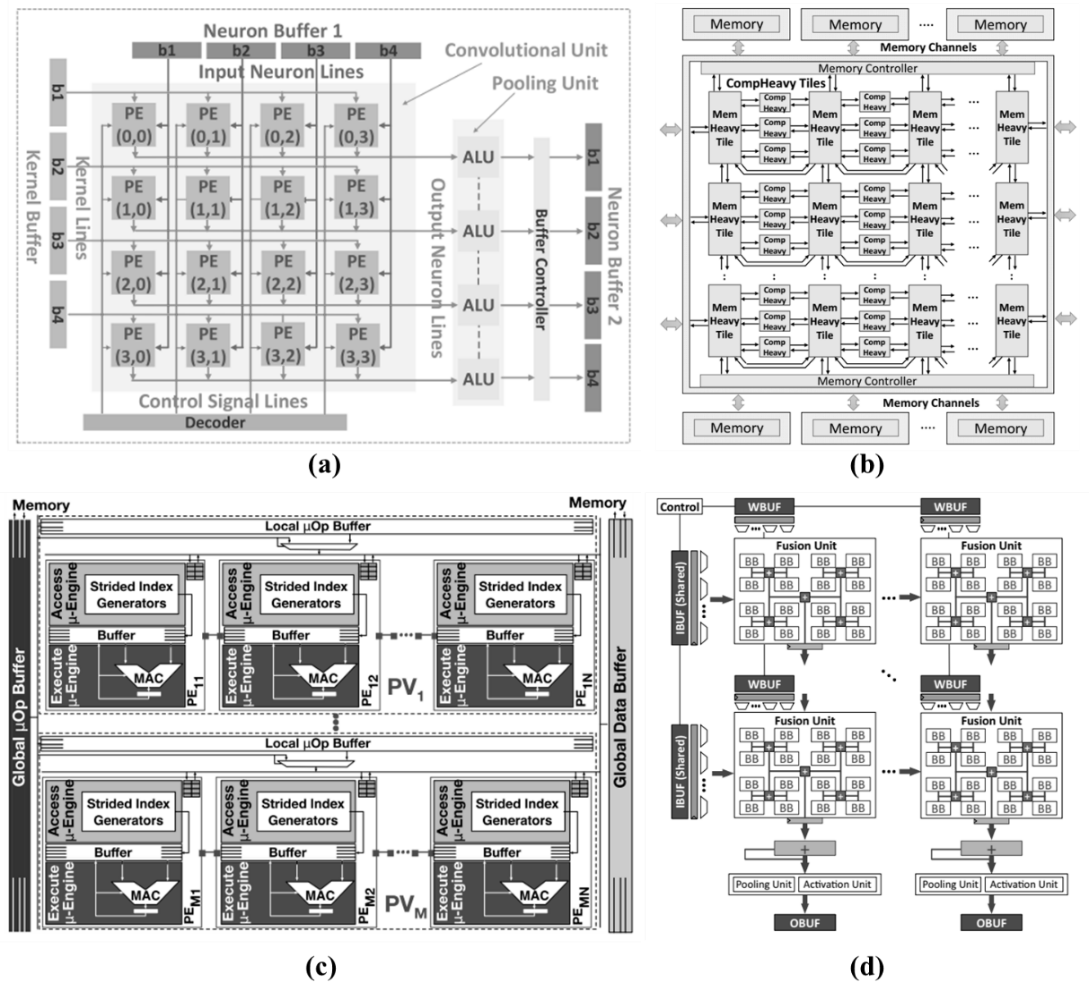


图15 几款面向AI的可重构计算架构：(a) FlexFlow[17] (b) SCALEDEEP[19] (c) Ganax[21] (d) Bit Fusion[20]

[参考文献]

1. XPP-III Processor Overview White Paper”, www.pactxpp.com.
2. “DAPDNA-2 Dynamically Reconfigurable Processor”, http://www.ipflex.com/
3. Mei B, Vernalde S, Verkest D, et al. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix[C]//International Conference on Field Programmable Logic and Applications. Springer, Berlin, Heidelberg, 2003: 61-70.
4. 杨博涵. 针对多媒体图像处理的可重构处理元设计[D]. 西北工业大学, 2005.
5. 罗赛. 可重构计算系统体系结构研究与实现[D]. 中国科学技术大学, 2006.
6. Liu L, Deng C, Wang D, et al. An energy-efficient coarse-grained dynamically reconfigurable fabric for multiple-standard video decoding applications[C]//Proceedings of the IEEE 2013 Custom Integrated Circuits Conference. IEEE, 2013: 1-4.
7. Nowatzki T, Gangadhan V, Sankaralingam K, et al. Pushing the limits of accelerator efficiency while retaining programmability[C]//2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2016: 27-39.
8. Prabhakar R, Zhang Y, Koeplinger D, et al. Plasticine: A reconfigurable architecture for parallel patterns[C]//2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2017: 389-402.
9. Voitsechov D, Port O, Etsion Y. Inter-thread Communication in Multithreaded, Reconfigurable Coarse-grain Arrays[C]//2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2018: 42-54.
10. Chen T, Srinath S, Batten C, et al. An architectural framework for accelerating dynamic parallel algorithms on reconfigurable hardware[C]//2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2018: 55-67.
11. Chen T, Du Z, Sun N, et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning[C]//ACM Sigplan Notices. ACM, 2014, 49(4): 269-284.
12. Chen Y, Luo T, Liu S, et al. Dadiannao: A machine-learning supercomputer[C]//Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2014: 609-622.
13. Liu D, Chen T, Liu S, et al. Pudiannao: A polyvalent machine learning accelerator[C]//ACM SIGARCH Computer Architecture News. ACM, 2015, 43(1): 369-381.
14. Du Z, Fasthuber R, Chen T, et al. ShiDianNao: Shifting vision processing closer to the sensor[C]//ACM SIGARCH Computer Architecture News. ACM, 2015, 43(3): 92-104.
15. Chen Y H, Krishna T, Emer J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks[J]. IEEE Journal of Solid-State Circuits, 2017, 52(1): 127-138.
16. ENVISION: A 0.26-to-10TOPS/W Subword-Parallel Computational Accuracy-Voltage-Frequency-Scalable Convolutional Neural Network Processor in 28nm FDSOI.
17. Lu W, Yan G, Li J, et al. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks[C]//2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2017: 553-564.
18. Parashar A, Rhu M, Mukkara A, et al. Scnn: An accelerator for compressed-sparse convolutional neural networks[C]//2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2017: 27-40.
19. Venkataramani S, Ranjan A, Banerjee S, et al. Scaledeep: A scalable compute architecture for learning and evaluating deep networks[C]//Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA). 2017: 13-26
20. Sharma H, Park J, Suda N, et al. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks[C]//Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2018: 764-775.
21. Yazdanbakhsh A, Samadi K, Kim N S, et al. Ganax: A unified mimd-simd acceleration for generative adversarial networks[C]//Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2018: 650-661.
22. Song L, Mao J, Zhuo Y, et al. HyPar: Towards Hybrid Parallelism for Deep Learning Accelerator Array[C]//International Symposium on High-Performance Computer Architecture (HPCA). 2019.

## （三）针对机载算法的异构可重构架构适配研究

本课题致力于完成对机载智能算法蒙特卡洛树搜索算法以及其配套需要使用的深度神经网络的整体计算架构进行适配性研究。基于对决策树的等智能终端应用中采用的神经网络的多样性、共通性分析，实现算子与运行模式的分离，加速其中并行逻辑，优化串行执行逻辑的数据通道，从而提高处理速度，同时构建动态可配置机制，降低芯片面积，提高能效比。主要包含以下两个模块。

### 1、蒙特卡洛树搜索结构适配性

MCTS 的并行方法主要分为三种：

图示

描述已自动生成

图 16 MCTS并行方法

1. **叶并行 (leaf parallelization)**, 即在叶节点扩展时进行并行。在遇到叶节点时，同时执行多次模拟 (simulation)，然后使用多次模拟的结果来代替之前的结果即可。
2. **根并行 (root parallelization)**, 即直接使用进程或线程创建多个不同的树，在不同的树中同时执行搜索。
3. **树并行 (tree parallelization)**, 即多个线程在同一个树中进行并行，每个线程在树的不同部分执行搜索。搜索过程中需要对树的一些线程加锁，一般方法分为全局锁（global mutex）即仅利用一个线程来更新和遍历树，其他线程用来并行模拟(simulation)；以及局部锁(local mutex)，多个线程都可以进入树的搜索，但是经过节点时会对该节点加锁，这个节点的计算完成后才会将该节点的锁解开。

在当前场景下，叶节点的并行并不适用。树并行相对复杂和繁琐，涉及到多线程之间的通信等待，并行效果也不一定好，推荐使用根并行的方法。根并行方法也分为single-run和multiple-runs。其中 single-run，每个线程采用不同的随机种子各自分别搜索一棵树，到底一定一时间后，将结果合并。multiple-runs同样是每个线程采用不同的随机种子各自分别搜索一棵树，但每隔一段时间，会将搜索结果合并，其中访问次数的加和需要除以线程数，这样会有更好的效果。

实际上无论是single-run方式还是multiple-runs方式，和单线程进行同样的次数搜索，尽管并行的方法会进行一些重复的搜索，看起来总的有效搜索次数其实低于单线程的方法，但是在一些实验中反而效果更好。主要原因可能是单线程的搜索更容易进入局部最优，而多线程的搜索能从某种程度上规避这一点。

在该问题中，无论是采用single-run，multiple-runs的方法，都需要对搜索次数或时间以及并行的线程数进行重新测定，以满足最后的效果要求。同时，如果是multiple-runs还需要对什么时候进行同步效果更好进行进一步测算研究。

### 2、支持乘加算子的多层级动态重构的计算架构

**2.1 乘加算子计算架构适配**

根据前一节中对可重构的介绍，该研究内容属于动态重构基础理论，对整个项目起到基础支撑作用。信息处理和神经网络任务中计算量和数据量十分庞大，因此需要研究支持资源复用和多层级动态重构的计算架构，通过对可重构计算单元及其阵列结构进行动态重构和扩展，实现计算操作、数据处理流程、计算规模均可变的硬件加速电路，分别从多个维度提高芯片的计算能效和灵活性。

一方面，研究计算电路的多层级动态重构。在运算部件层面研究可配置的核心部件，使其具有位宽扩展能力；在计算单元层面，研究计算单元内部资源复用机制，通过重构数据通道实现多算子支持；在单元阵列层面，设计高位宽和低位宽两种计算单元融合的异构阵列，并研究阵列重构机制，适配不同访存比，优化访存行为，提高能效。其中，高位宽计算单元的主要目标是灵活通用，可以兼容32位浮点、32位定点以及16位定点计算，且可以实现多种计算类型；而低位宽计算单元的主要目标是高能效，主要负责执行神经网络中16位到1位的定点计算。

另一方面，研究动态重构的配置机制，包括异构阵列的配置信息通路和缓存机制，以及动态重构中的配置时间隐藏机制。另外，由于本课题面向的计算任务可能包含大量多层循环，其外层流水核心往往比较庞大，所以需要研究配置信息压缩技术。

以动态重构理论为基础，研究本课题中的关键技术：支持神经网络推理和训练的硬件架构。主要研究既支持神经网络推理又支持片上训练的计算架构。针对正向推理过程，主要探索神经网络运算中数据复用和数据流架构技术，并结合硬件架构特性和网络特性，研究能依据架构属性为每一层网络提供合适数据复用模式的性能分析模型。在设计训练架构时，充分研究后向传播训练算法，考虑如何复用现有的推理硬件结构，合理增加计算逻辑和控制逻辑。其中后向传播算法分为计算特征错误图(error map)和更新权重梯度值(gradient update)，根据运算特点，考虑如何在异构阵列上分配训练的计算任务，尤其针对训练过程中上采样的计算过程，需考虑不同于传统下采样卷积计算时的片上映射模式和数据复用模式。

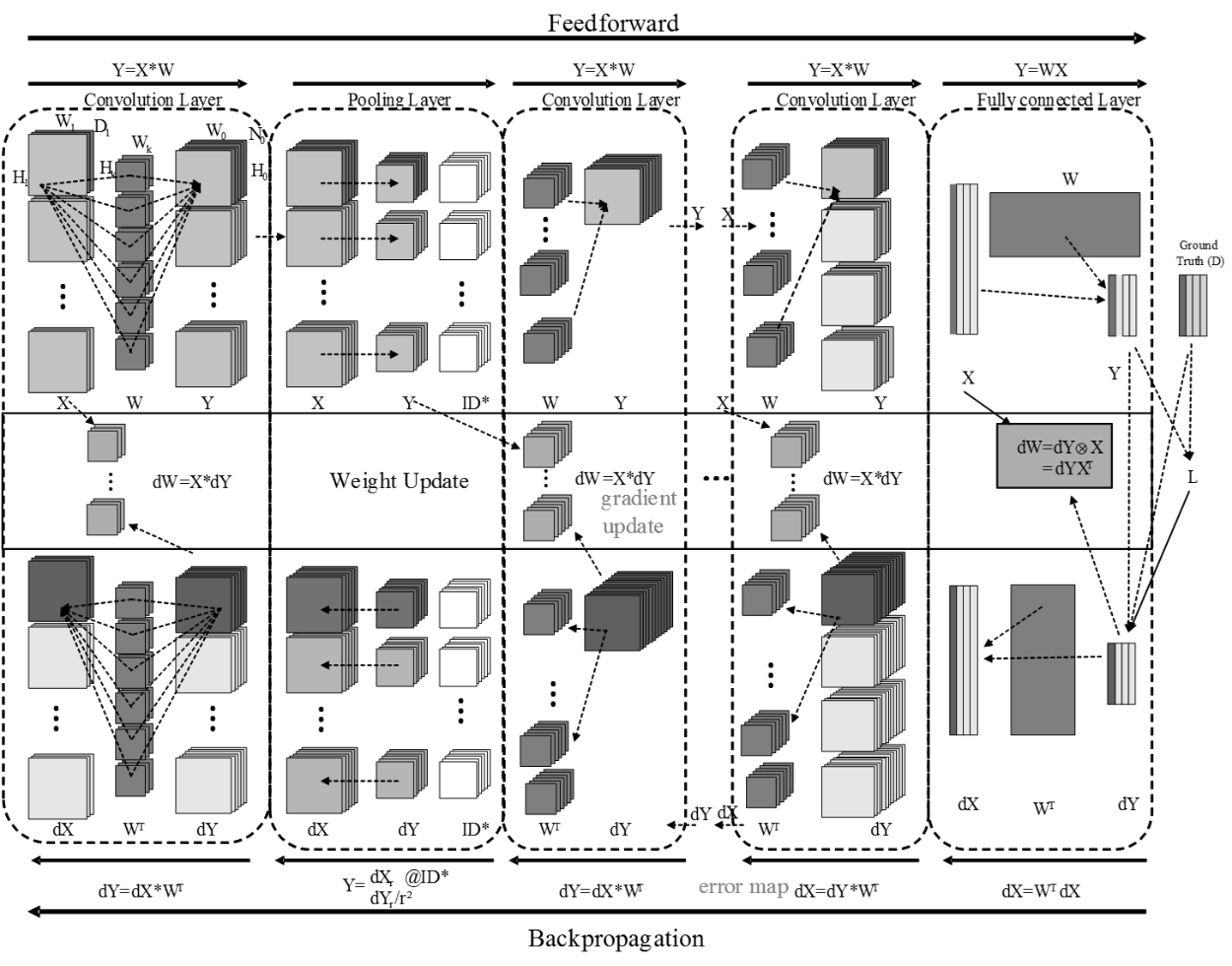


图 17 正向推理和反向传播时过程示意图

**2.2 神经网络运算结果适配性**

根据前一章第3节对基于蒙特卡洛树搜索的智能决策算法结构的分析可知，算法中神经网络部分是由4层全连接网络组合而成，输入维度为110，输出返回值为9个决策机动号相对应的的分值以确定下一次搜索的路线。因此，该算法神经网络部分只包括全连接网络层单元和激活函数单元。具体地，激活函数包括ReLU系列函数以及Softmax函数，全连接网络层单元主要包括大规模矩阵乘加运算。

下图所示是深度神经网络可重构计算的整体架构示意图。因为神经网络的核心运算形式可以等价为矩阵向量乘法，所以本课题采用一种可重构二维计算阵列（PE array）作为核心的计算部件，以提高架构的并行处理能力。阵列内的计算单元互连可以根据网络的不同而进行重构，提高资源利用率。此外，因为不同神经网络（如卷积神经网络、递归神经网络）有各自的特殊的计算，如池化、门控、激活等计算，本课题采用异构的方式设计计算阵列，大部分计算单元只支持核心的乘加计算，少部分计算单元支持更加复杂的计算，提高计算阵列的适应性。此外，阵列的配置信息结构和配置接口也根据硬件架构进行优化，实现运行过程中的动态重构。

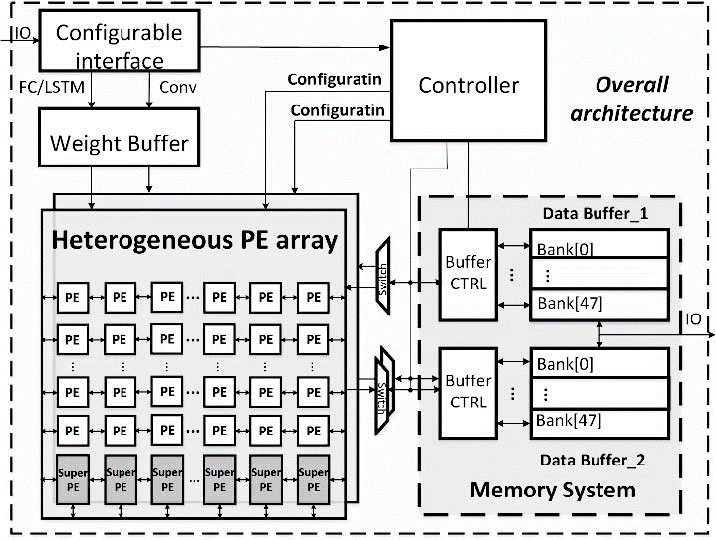


图 18 面向神经网络运算的可重构计算阵列设计

本方案中配置信息的高速缓存结构将可重构处理器近期将要执行的配置信息存储在片上高速缓存中，当可重构处理器执行这些配置时，可以从片上配置信息高速缓存中获得，不需要去外部存储器中读取。由于片上高速缓存的访问延时（一般为几个时钟周期）要远远低于外部存储器，因此配置信息的高速缓存结构可以减小配置信息传输时间对可重构处理器性能的影响。在配置信息的高速缓存中，只存储当前使用的部分配置信息，有利于减小存储在片上的配置信息数据量可以减小配置信息存储器的面积和功耗。在可重构处理器运行过程中，配置信息的高速缓存结构可以实时地为可重构处理器读取新的配置信息，不需要重新初始化可重构处理器，保持了可重构处理器的灵活性。

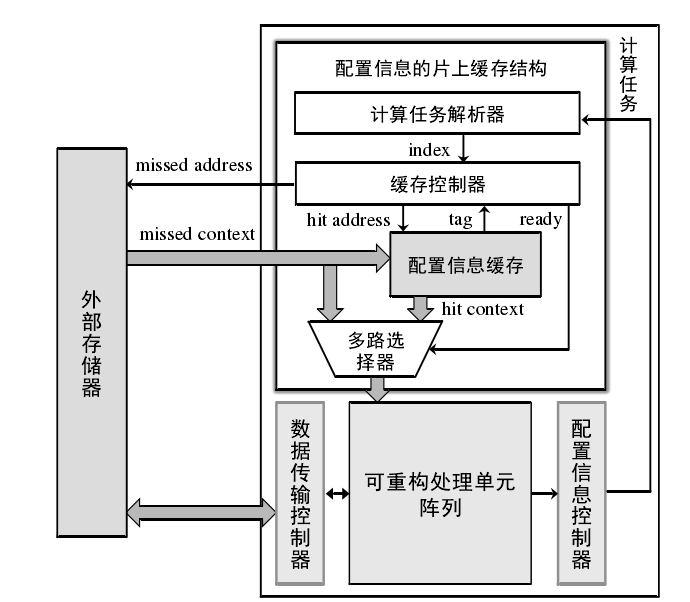


图 19 配置信息的片上高速缓存结构

当可重构处理器执行的配置信息不在片上缓存中，即发生缓存缺失时，需要从外部存储器中读取该配置信息到可重构处理器内部，可重构处理器需要等待配置信息传输完成才能开始执行，这会降低可重构处理器的性能。因此，本课题进一步提出了配置信息的分组预取方法，将配置信息按照不同的计算任务进行分组，对于完成同一个计算任务的一组配置信息进行预取，可以减少配置信息的缓存结构中发生缓存缺失的次数。

配置信息的告高速缓存结构如上图所示，它主要由计算任务解析器、缓存控制器、配置信息缓存和多路选择器四个部分构成，根据配置信息控制器分配的计算任务，计算任务解析器负责解析完成计算任务需要的配置信息，得到配置信息索引（index),发送给缓存控制器。缓存控制器根据配置信息索引，首先查找配置信息缓存的标签位(tag),如果配置信息索引与标签位一致，说明该配置信息己经存储在配置信息缓存中，即在缓存中命中(hit),此时配置信息控制器根据标签位产生该配置信息在缓存中的存储地址（hit address),发送给配置信息缓存模块。

用于深度学习的神经网络层数也不断增加，权重系数也不断增加。以ALEXNET 为例，一张图片的数量是150KB，权重数量是120MB; 为了让此类图像识别的计算阵列上运行起来，除了阵列本身需要很高的计算能力之外，高速的数据交互接口设计也是关键。按照上述设计的接口计算（带宽128M/cycle），配合4个256 PE的子阵列，在500MHz的工作频率之下，执行ALENET可以达到33GOPS的执行速度，这相比目前最好的深度学习芯片将有10倍左右的速度提升。

因为神经网络计算过程中对存储容量和带宽有巨大的需求，因此计算架构需要很高的DRAM带宽接口。但是神经网络在运行不同层的时候DRAM带宽需求会发生波动。通常全连接层的带宽需求很高，而卷积层因为有大量权重复用，带宽需求较低。如果DRAM一直保持高带宽，将会造成功耗的浪费。给定带宽下，不同的线宽和电压下的功耗不同，并存在一个功耗极小点。本课题将根据神经网络在架构上的执行时间和访存量，评估每一神经网络层的带宽需求，再调节线宽和电压/频率来最优化DRAM功耗，其中频率是正相关于电压的。本课题将研究可编程的存储接口设计：包括研究物理上如何设计可编程接口，能够对线宽、电压和频率同时调节以降低功耗；并研究动态配置机制以快速调节DRAM带宽，而不影响系统性能。

除了配置的缓存问题与可编程存储接口的设计适配程度，对于运算单元层面来说，则需要通过重构选择不同的计算位宽、计算通道、组合不同的功能模块，提高芯片灵活性。如下图以精度自适应的可重构乘加计算单元为例。为了减小神经网络的规模，在实际使用时常常对网络进行定点化压缩处理，使得网络的数据位宽和权重可以是8比特、4比特、2比特甚至1比特。传统的乘加单元设计采用固定的数据精度（通常为16比特），如果仍执行低比特的计算任务就会造成大量的性能和能耗浪费。精度可配置的计算单元则可以根据需求的不同，充分支持可变位宽神经网络运算，实现硬件资源的最大化利用。并且针对低比特的网络，往往具有更高的计算并行度，提高了计算能效。此外，因为不同神经网络（如卷积神经网络、递归神经网络）有各自的特殊的计算，并且除卷积操作以外还含有全连接操作、池化、门控、激活等计算，运算单元需要复用多种硬件资源，提高芯片的灵活性。因此，需要研究高度灵活的多精度、功能可配的运算单元。

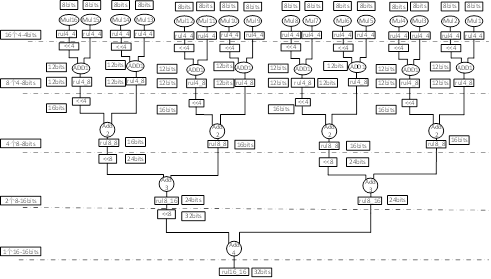


图 20 精度可配置乘加单元设计示意图

然而正如前文所说，在神经网络的层内运算中，除了大量密集计算的特性，神经网络计算还有一个很重要的特点是容错性，即某个神经元输出的微小偏差对输出结果影响不大。这是因为神经网络的输出通常是分类或者识别的结果，并不需要特别精确的数值表示，某个神经元的误差对最终分类/识别结果的影响不大。因为神经网络的容错性，我们利用近似计算技术，通过精简计算单元的实现逻辑来降低功耗，并带来能效上的提升。如下图所示是一个精确的全加器单元(上)和两个近似的全加器单元(下)实现。两个近似单元分别采用了更少的逻辑来实现全加器计算，但是带来了12.5%到37.5%的计算误差。

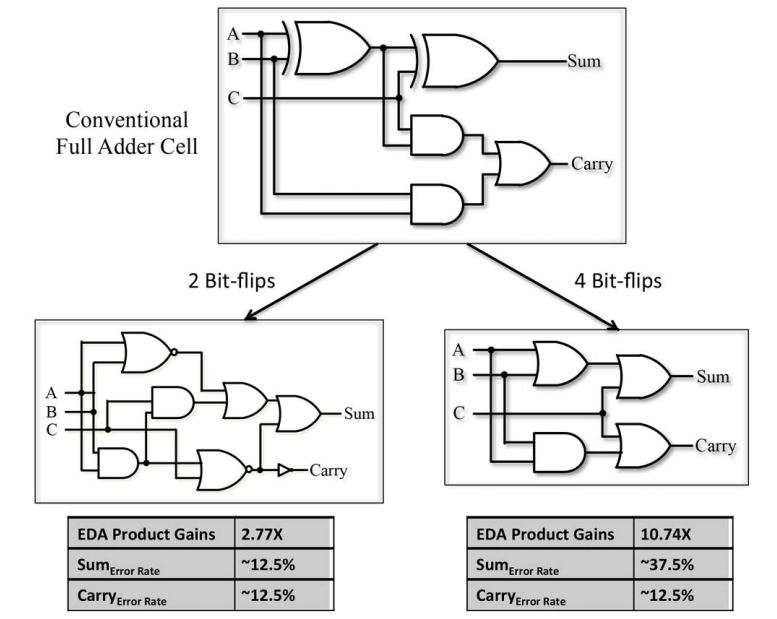


图 21 高能效近似乘加单元设计

本课题将设计一个近似精度可配置的乘加单元。在乘加单元内部设计多个精度控制开关，在计算过程中根据计算任务的需求关闭部分数据通路以节省能耗。同时，设计误差修复的神经网络训练方法，在训练神经网络过程中引入计算单元的近似误差，通过重训练保证网络收敛。因此，网络精度不会应为近似误差造成损失，同时因为不必要的计算逻辑关闭，节省了大量能耗，大大提升了计算能量效率。

### 3、支持其他算子的多层级动态重构的计算架构

**3.1 多模态处理器的融合与编译**

在可重构基础理论研究和关键技术的基础上，研究树搜索和人工智能融合的多模态处理器架构。它不单独针对某一种人工智能算法或者神经网络模型进行优化设计，需要同时具有多任务协同计算能力、多模态数据处理能力和应对算法演进的可扩展能力，并且保证足够高的性能和能效。需要支持三种计算模式，分别是①异构单元并行处理多个独立的计算任务、②异构单元协同处理混合精度的神经网络任务、以及③异构单元协同处理神经网络片上学习任务。

基于上述需求，本课题研究的蒙特卡洛树搜索和人工智能融合的多模态处理器架构，需要围绕核心部分的异构计算阵列，设计一整套包含输入输出、控制、存储、计算等部件的完整可重构计算架构，为了适应树搜索和人工智能任务的大数据量、高并行度、多种计算位宽等特点，保证计算的高灵活性、高性能和高能效，本课题需要充分研究异构计算阵列两种计算单元的最优比例和分布、计算单元的优化结构、有效的互连形式、以及简洁高效的存储组织形式，还需要设计灵活的控制系统，控制高位宽和低位宽计算单元分工合作，形成优势互补，高效地完成任务。

针对编译的方法，本项目与多模态处理器架构互相匹配，研究异构编译方法。为了将输入的节点拓展信息和神经网络任务的描述转化成优化后的架构配置信息，使任务在前面描述的计算架构上正确、高效地执行，需要设计完整的异构编译器。

首先，为了兼容蒙特卡洛树搜索算法应用或者神经网络的层次化描述或者C/C++代码等不同形式的输入，需要设计一个前端，对输入进行一定的预处理，得到统一的计算表达形式。然后，需要开发一个任务划分器，将任务划分为两部分，以分别执行于异构单元阵列上。接着，被划分出来的两部分任务将分别进行优化。对于高位宽算法部分，需要研究基于多面体变换的时空二维映射技术，实现高效的流水线执行。对于低位宽算法部分，需要研究神经网络层间聚类并行技术，实现资源高能效互补利用。同时，还需要研究两部分的分工与通信配置，构建简洁有效的存储系统。并且，需要开发配置信息生成器，将以上过程得到的优化参数和数据流动方式翻译成架构配置信息。配置信息用于输出给控制器，以驱动计算器件执行任务。

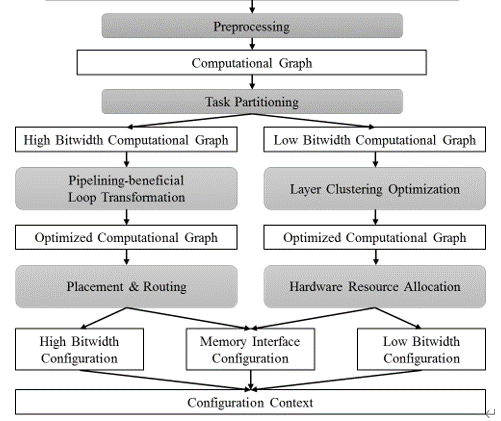


图 22 异构编译流程图

**3.2 深度学习引擎配置工具**

在阵列化可重构层面，运算单元组成的阵列具备可重构属性，进而来适配不同访存比，提高资源利用率，降低能耗。因为不同的卷积神经网络具有不同大小的卷积核，且每个卷积核的通道数不同，当输入特征图和卷积的计算任务映射到一定规格的计算阵列中时，往往会因为阵列尺寸不完全匹配输入特征图大小或者卷积核通道数，使得部分计算单元处于空闲状态，造成资源的浪费。可重构的计算阵列，能够根据任务需求，按照控制信息，灵活地形成不同规格的阵列结构，适应不同种类的算子运算。为实现这一目标，需要研究可重构阵列的互连、存储及配置机制。

不同类型的神经网络层具有不同的计算访存比。卷积层的操作，属于计算密集型操作，无法充分利用片上存储的带宽，但需要大量的计算资源；全连接操作属于存储密集型，运行时需要充分的存储资源和带宽，却只需少量的计算资源。可重构的计算阵列需要根据算子的需求不同，灵活分配不同的计算资源给不同对象。与此相对应的，片上的存储资源也需要和计算资源进行动态配合和重构。

为了在智能任务中获得足够高的精度，神经网络往往需要大量的网络层，每一层中需要大量多通道的卷积核。这使得神经网络的数据量很大，在计算过程中造成了频繁的存储器访问。数据复用，即在计算过程当中对同一数据进行重复的利用，是一种常见的减少存储器重复访问的方法。可重构架构需要支持输入数据复用、输出数据复用以及权重数据复用三种模式，需要研究架构的高效配置机制。

对于深度学习算法的开发者来说，由于各算法框架（Caffe，TensorFlow等）的前端交互和后端实现之间都存在很多区别，更换开发框架成本很高，而开发和交付过程中可能会用到的框架不止一个。从芯片供应商的角度来看，开发一款芯片需要支持多个深度学习框架，每个框架表示和执行工作负载的方式都不一样。所以，需要一个深度学习引擎配置工具，支持多个框架，能够灵活地把深度学习模型部署到可重构计算架构上。

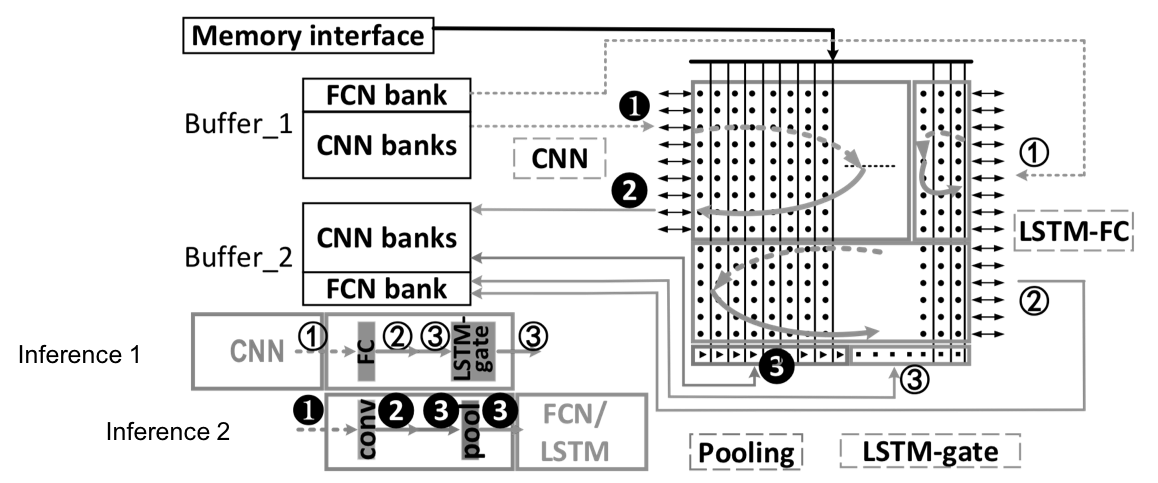


图 23 支持灵活配置的可重构阵列和存储系统

在接受到目标算法中的神经网络部分之后，首先需要对算法中神经网络部分进行详细的分析，主要包括每层神经网络抽象成的具体数学操作，输入、输出、参数的数据类型、大小、维度等。输入一方面来自算法框架的结构描述文件，另一方面来自具体的架构参数如工作频率、计算资源规模和存储带宽等。结构描述文件接口支持多种主流深度学习主流算法框架，工具将其转换成统一的数据流图中间表示方式。详细的分析结果不仅指导硬件开发人员设计CGRA中的功能算子和IP核,也帮助软件开发人员参数格式处理及参数包的构建等功能。工具以硬件资源作为约束，以能量效率为优化目标，构建优化问题，对神经网络的每一层进行的硬件配置方案进行优化。工具最终生成可重构的配置信息，包括对网络每一层的计算资源精度、数据通路和带宽等的硬件配置二进制文件。在线阶段，计算引擎将读取配置工具生成的二进制配置文件，针对网络的不同，动态地重构硬件资源，实现最优的计算效率。



图 24 自动化配置工具示意图

**3.3 其他拟解决的动态重构关键问题**

**1）、多层级动态重构架构的配置信息压缩问题**

在蒙特卡洛树搜索以及神经网络算法中，都有可能出现多层级的嵌套循环流水问题，当对多层循环尤其是非完美循环进行软件流水时，流水核心包含的内容较多，可能会导致动态重构的配置信息量超出存储能力。因此，需要逐层级地提取多层流水中重复的计算过程，得到多层流水核心中的最小重复单元，从而减少配置信息量。在执行时，采用多级索引的方式，相同计算模式可复用同一套配置信息单元，通过拼接各个外层迭代的配置信息元素，来实现完整的计算过程。但是，随着压缩的程度增大，会带来两方面的代价。首先，配置信息所能表达的灵活度会降低，导致更难搜索到从算子到计算阵列的合法映射。其次，索引系统的硬件代价会增加。

确定最优的压缩方案的问题建模如下：

其中，在给定循环和硬件参数的前提下，配置信息量由压缩级数和各个子循环是否压缩(用标记表示：)决定，索引代价由压缩层级决定，代价函数将这两方面的代价加权整合。约束条件是，在压缩方案下可以找到合法映射。

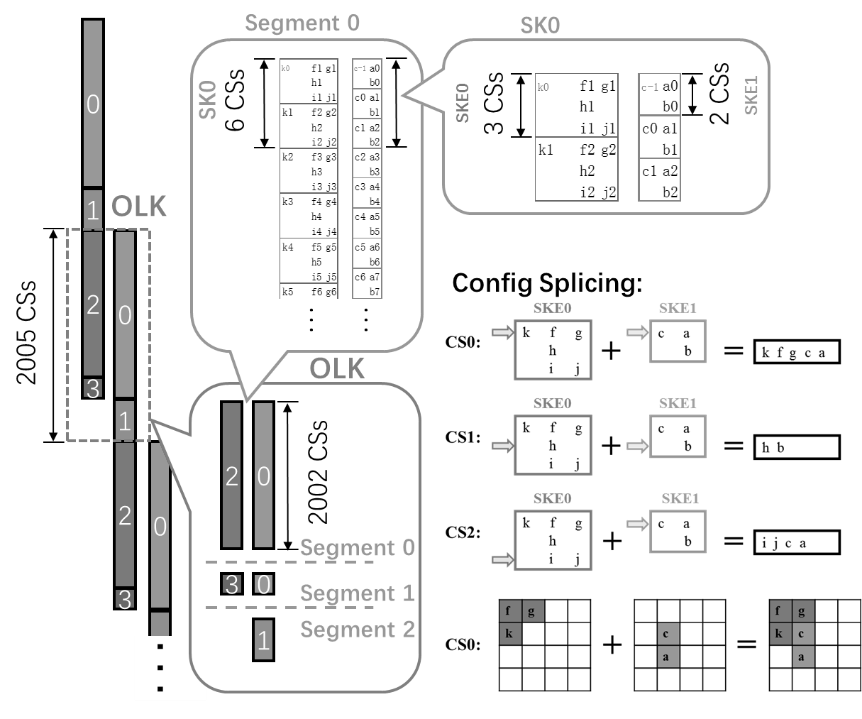


图 25 配置信息压缩示例

**2）、计算任务时空二维映射问题**

考虑到神经网络算法中用到了大量的矩阵乘法，为了提高执行树搜索和人工智能任务的效率，需要把循环中的算子有效地映射到空间计算阵列上，实现并行计算。需要保证算子的执行顺序在时间上合法，并且在空间上有合法的数据通路可以保证数据依赖的实现。

采用基于多面体表示的循环变换可以优化计算任务中的数据流，使得层与层之间的生产与消费良性匹配起来，缩短依赖长度（代表了计算任务中数据存取的时间跨度），从而实现有效的映射。本课题需要考察各种指标的实际影响，对各种代价进行加权求和，得出一个综合代价的模型（其中是要执行的循环，是变换矩阵，是各种代价的权重），然后寻找最优的变换方式使综合代价最小。综上所述，基于多面体的循环变换问题可以被描述为：

其中，为了使变换后的循环不违背数据依赖，就必须要保证对所有的依赖都满足。完成变换后，再将得到的结果映射到时域扩展后的计算阵列上，完成时空二维映射。

图26 (a)(b)(c)是多面体变换的例子。图26 (d)是迭代域和依赖关系，包含了约束条件。图26 (e)是一个包含两个计算单元的阵列，多面体变换结果需要映射到该阵列在时域上的扩展图（如图26(g)所示）

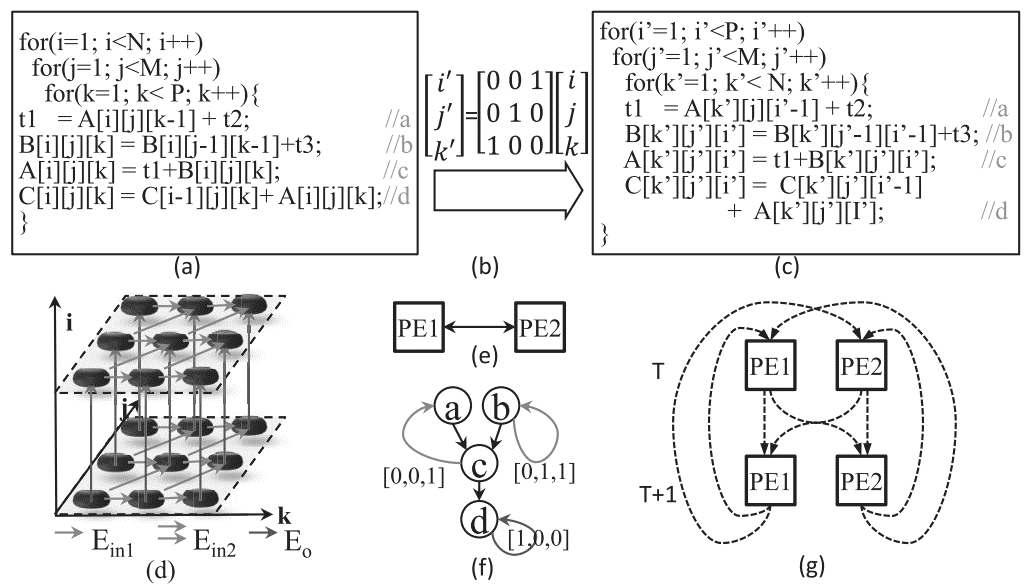


图 26 配置信息压缩示例

**3）、神经网络层间聚类映射问题**

啊神经网络中不同层的访存比（单位访存量可以支持的计算个数）不同，如果顺序执行各个层，计算密集层往往会浪费带宽资源，而访存密集层往往会浪费计算资源。因此，需要将神经网络中的层按照访存比等特征进行聚类，如图27 (a)所示，然后再对计算阵列进行分区，让各个类分别映射到计算阵列上的不同分区实现并行执行，以实现资源利用的互补，提高能效。

在分区工作模式下，同时有多个神经网络实例以流水的方式执行，每一个网络实例的不同阶段（层聚类）是在不同分区上执行的，如图27 (b)所示。平均执行一个神经网络实例的执行时间可以看做是：

其中，是分区编号，是编号为阶段的执行时间，是映射分区的子网络的编号集合。由每个阶段内部各层的执行时间决定，由该卷积层的参数、映射方式以及分配的计算和带宽资源量决定。在给定网络参数和硬件资源量后，影响的是硬件资源的分配方式和各阶段映射方式。需要搜索合理的资源分配和网络映射方式，在需求的资源量不超过可用资源量的条件下，使得最短。该问题的数学描述为如下形式：

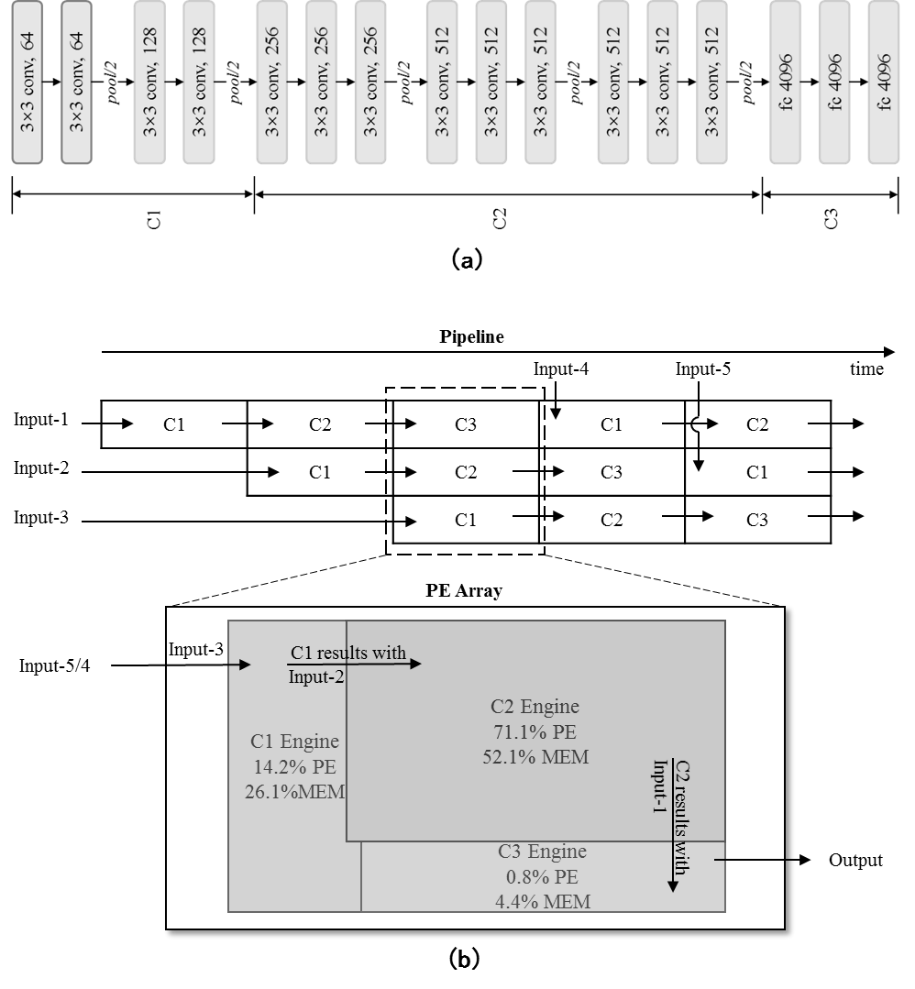


图 27 核心计算部分映射示例

**4）、可重构阵列控制流优化问题**

受空间计算阵列运算面临的一个挑战是如何映射程序中的if-then-else（ITE）分支控制结构。默认的循环映射框架把ITE的所有分支都映射到空间计算阵列上，会造成严重的硬件资源浪费。因此，本课题需要研究如何优化带有ITE结构的数据流图（Data Flow Graph，DFG），采用合法合并、有效消除和触发转换的启发式方法来缩减算子数量，最小化DFG，如图46所示。

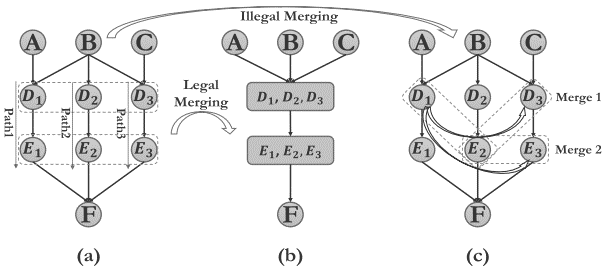


图 28 分支控制结构优化流程

此过程的问题建模如下：

基本定义：是输入DFG，其中节点集合。是选中的算子集合，是冗余节点集合，是需要映射到运算器上的普通操作的节点集合。是输出DFG，表示合并了多分支节点并删除了冗余节点后的循环核心。是表示触发指令的节点集合，是中的节点的预词依赖对集合。

优化目标：最小化的节点个数。

约束条件：（1）合并节点约束：可合并节点必须是分布于同一层级条不同分支路径但是在更新同一个变量的算子。（2）冗余节点：每个冗余节点是一个控制布尔算子（AND,OR,NOT），每个布尔算子的源算子包含一到多个预词。

**5）、片上训练过程中批量归一化的优化问题**

Batch Normalization（BN）完成批量数据的归一化（规范化），它和激活函数层、卷积层、全连接层、池化层一样，也属于网络的一层。在训练过程中，对网络某一层数据进行归一化，可以有效地加快训练速度，并使得网络训练收敛。传统的BN层计算模式如下，首先统计输入的平均值，再计算标准差，最后利用所得的结果进行归一化处理。

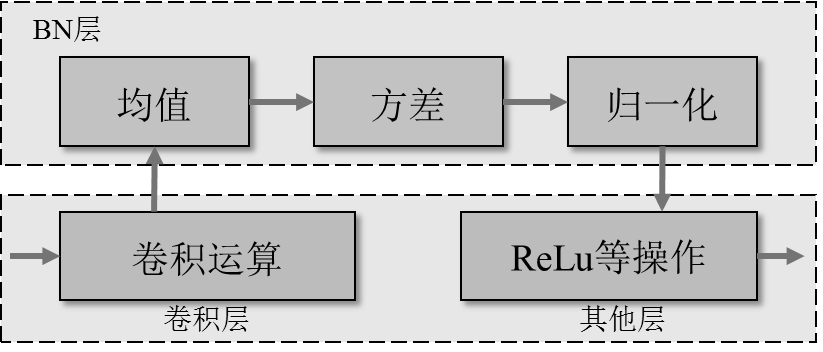


图 29 BN层处理流程

在片上实现传统BN需要遍历当前训练批次的所有输入数据，统计出均值和方差。虽然目前最新研究 Range-BN，避免了复杂度高的方差计算，但是依然需要统计所有输入数据的最值。BN层复杂的计算方式使得处理不仅耗时严重，而且消耗大量计算资源。此外，在训练过程中把BN层作为独立的层接在卷积层之后，这就要求卷积完成后的数据再从片外搬到片上进行BN操作，导致了二次搬运，造成大量的片外访存。如何进一步优化批量归一化算法来近似替代传统的BN操作，使其既能实现数据的归一化处理又能保证模型精度不丢失，同时有利于低功耗、高性能的硬件实现，是实现神经网络训练架构需要关注的重点问题。

## （四）机载智能计算单元研究方法及可行性分析

### 1、研究方法和技术路线

本项目总体研究方法如下图所示。首先对机载智能计算单元和人工智能算法提炼出基本算子，从基本算子构建计算架构，再针对计算架构进行量化分析和理论建模。然后展开硬件的实现，通过和算法模型及软件模拟器的运行结果进行比对，完成硬件功能完备性验证以及性能评测和优化，最后进行系统性集成、流片验证。

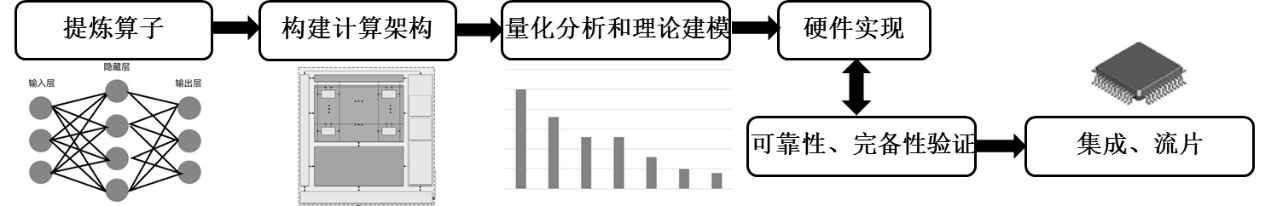


图 30 研究方法路线图

本项目的题技术路线总共分为以下几点。

首先是对算法进行研究，提炼出算子运算特点。分析通用架构的不足，研究多模态异构阵列，支持不同精度的运算，充分复用片上逻辑资源，提高硬件资源利用率。

然后研究架构的可重构特性，使其具备不同层面的可重构属性。在运算单元层面，通过重构使PE 阵列可以根据配置信息灵活处理不同计算任务类型的算子，支持多比特算子运算。在阵列化层面，阵列可重构，进而适配不同访存比，提高资源利用率。

之后设计采用多目标优化算法寻找异构融合架构的最优设计，并采用计算任务动态分配策略实现高效片上资源利用，同时采用异构计算阵列结构，确保各子阵列的并行性及子阵列间处理潜力能够充分发挥，全局控制器根据任务为各子阵列中的计算单元产生配置信息，高效有序地完成任务。

### 2、实验手段

用高级语言实现架构性能分析模型，分析不同架构的预期性能。性能分析模型的输入是架构设计参数、神经网络描述以及蒙特卡洛树搜索算法，进行cycle 级别的性能仿真后得到性能评估结果。

用高级语言实现架构功耗分析模型，评估不同架构的平均功耗。功耗分析模型的输入是架构设计参数，内嵌一个基本单元功耗库，可以对架构进行功耗分析，实现高能效设计。

随后是任务的基于高层次综合的映射问题，任务的映射分三步骤:前端输入模块将输入的神经网络的层次化描述脚本或者其他通用机器语言转为中间表示形式；中段优化模块对不同类型的神经网络层级实现优化求解；后端输出模块根据实际的硬件参数，调整配置信息的格式。

# 六、交付技术状态

见表1。

表 1 交付技术状态

|  |  |  |  |
| --- | --- | --- | --- |
| 交付类别 | 交付内容 | 交付数量 | 交付形式 |
| 软件类 | 面向国产可重构计算芯片的机载智能算法软件 | 1套 | 软件代码 |
| 文档类 | 项目总体方案设计报告 | 1份 | 电子文档  （光盘形式） |
| 面向国产可重构计算芯片的机载智能算法适配算法软件详细设计与测试报告 | 1份 |
| 工作总结报告 | 1份 |