

ゼロから紐解くパーセプトロンの実装

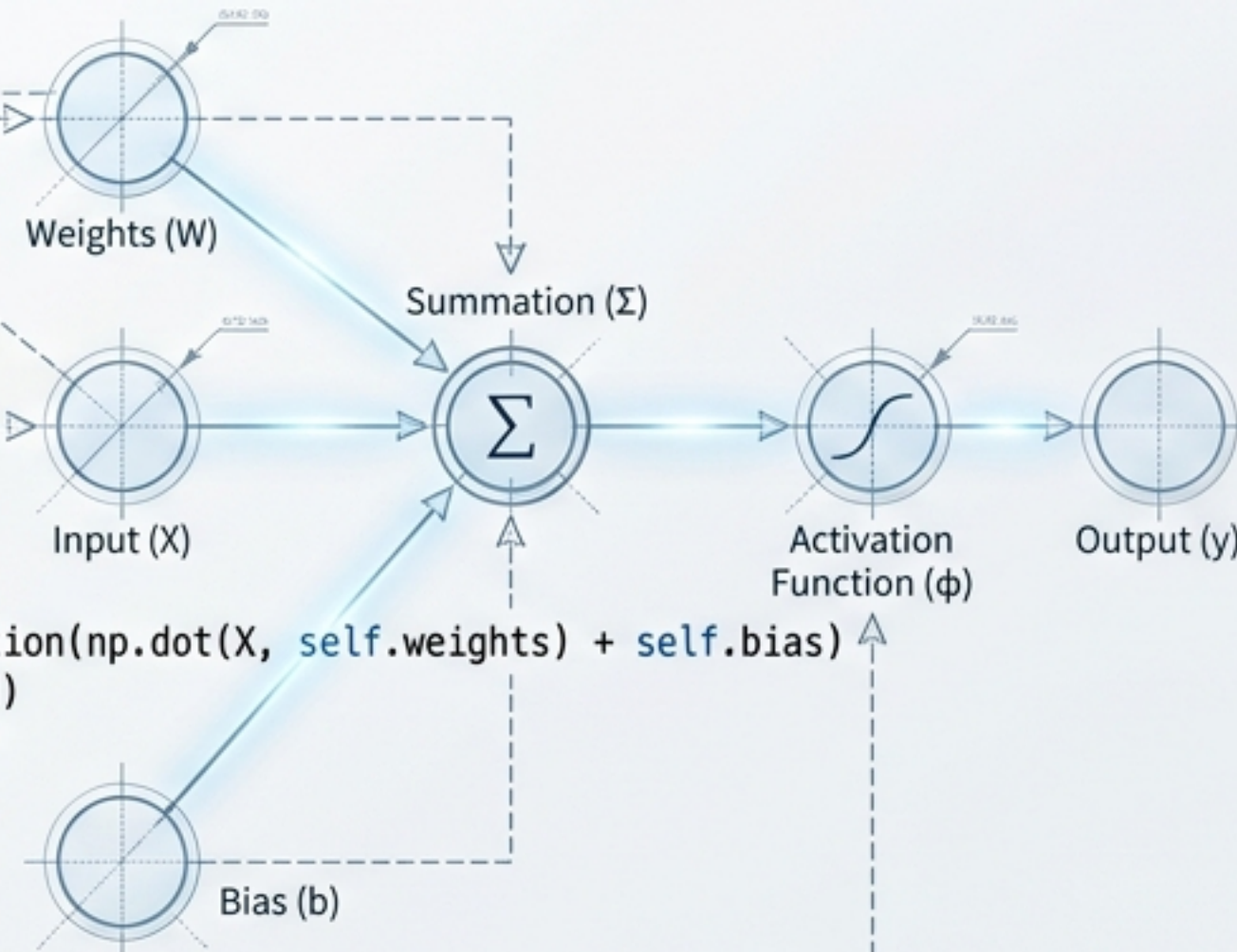
2019年秋期 応用情報技術者試験（問3）Pythonコード・ブロック完全解剖

Concept to Code: Neural Network Fundamentals

```

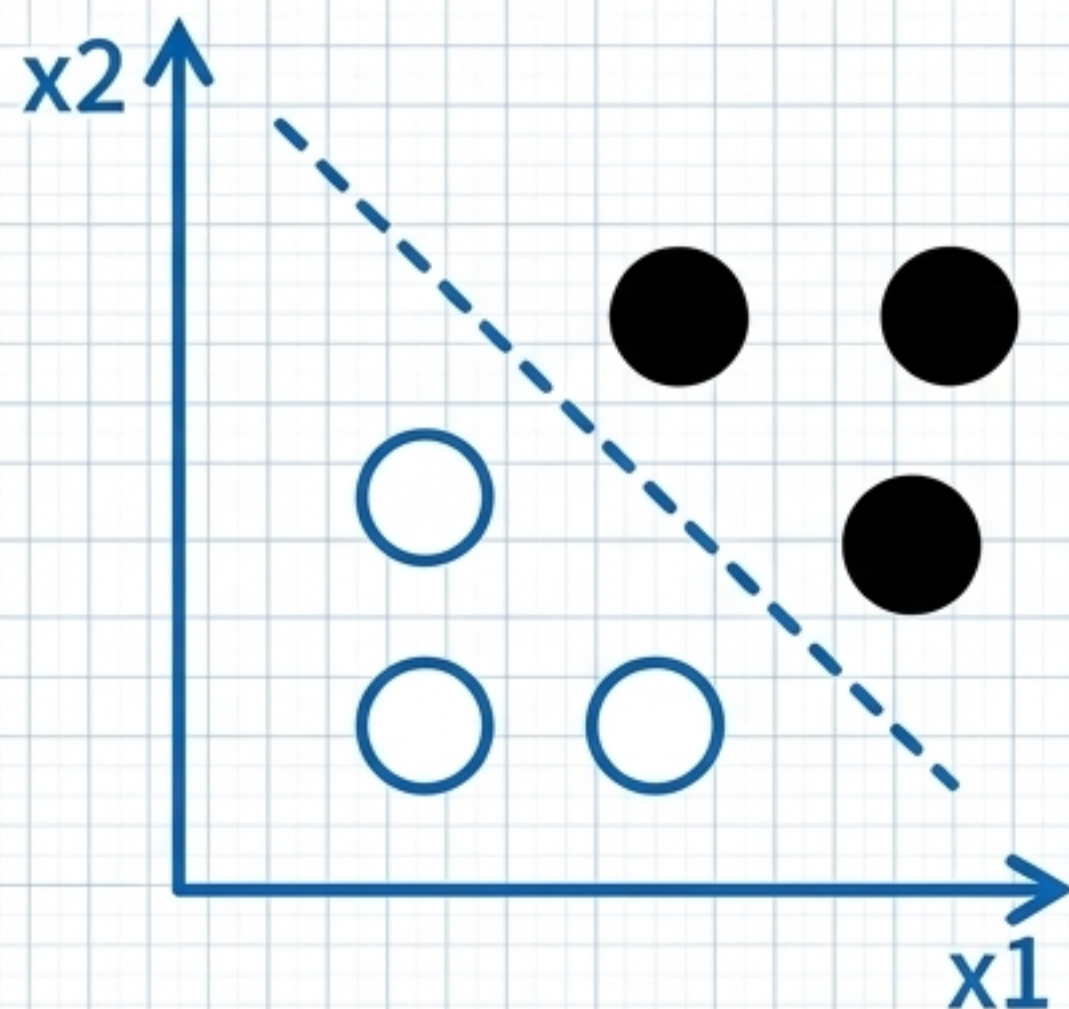
1 class Perceptron:
2     def __init__(self, learning_rate=0.01, n_iters=1000):
3         self.lr = learning_rate
4         self.n_iters = n_iters
5         self.weights = None
6         self.bias = None
7
8     def fit(self, X, y):
9         # Initialize parameters
10        n_samples, n_features = X.shape
11        self.weights = np.zeros(n_features)
12        self.bias = 0
13
14        for _ in range(self.n_iters):
15            # Update rule
16            y_predicted = self.activation_function(np.dot(X, self.weights) + self.bias)
17            update = self.lr * (y - y_predicted)
18            self.weights += np.dot(X.T, update)
19            self.bias += np.sum(update)
20
21        def activation_function(self, x):
22            # Step function
23            return np.where(x >= 0, 1, 0)

```



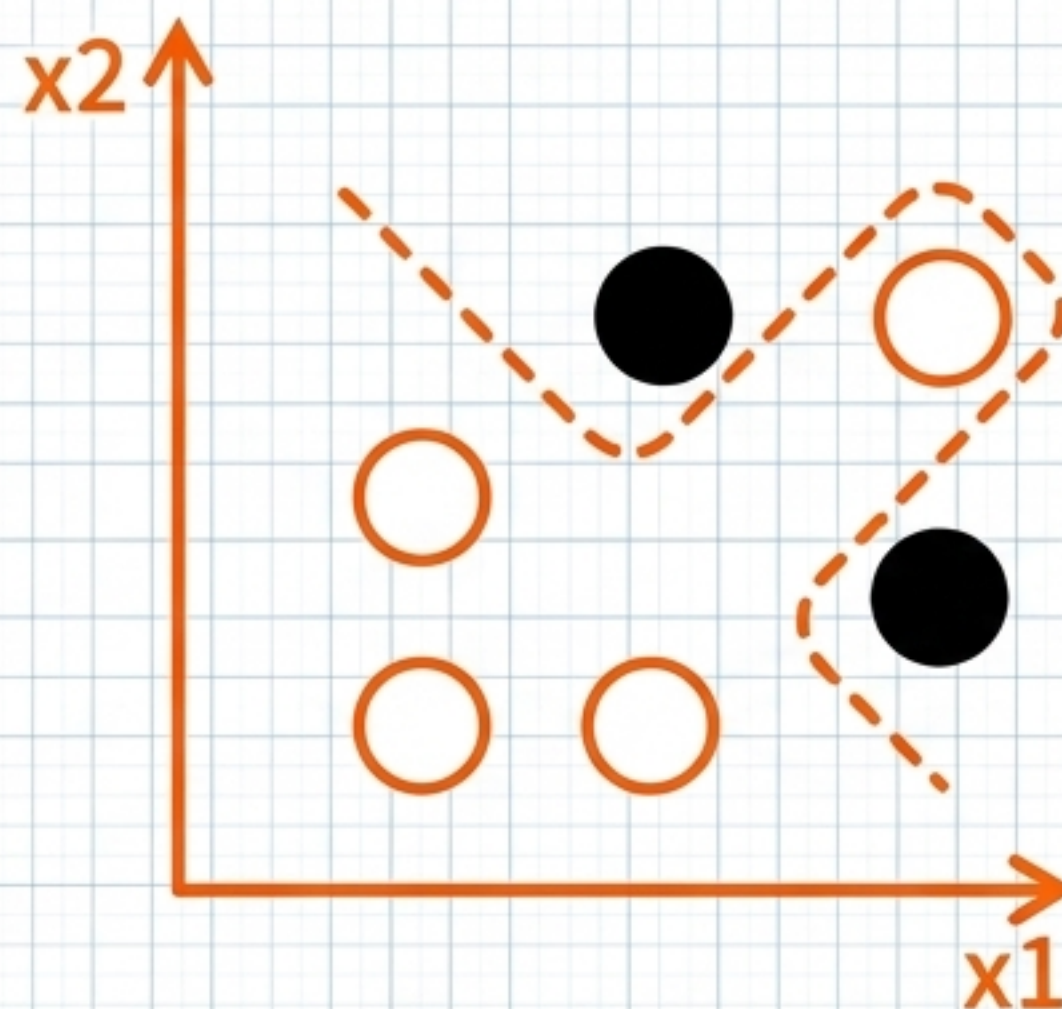
前提知識：分類問題における線形と非線形の境界線

線形問題（1本の直線で分割可能）



AND, OR, NAND, NOR

非線形問題（直線の限界）

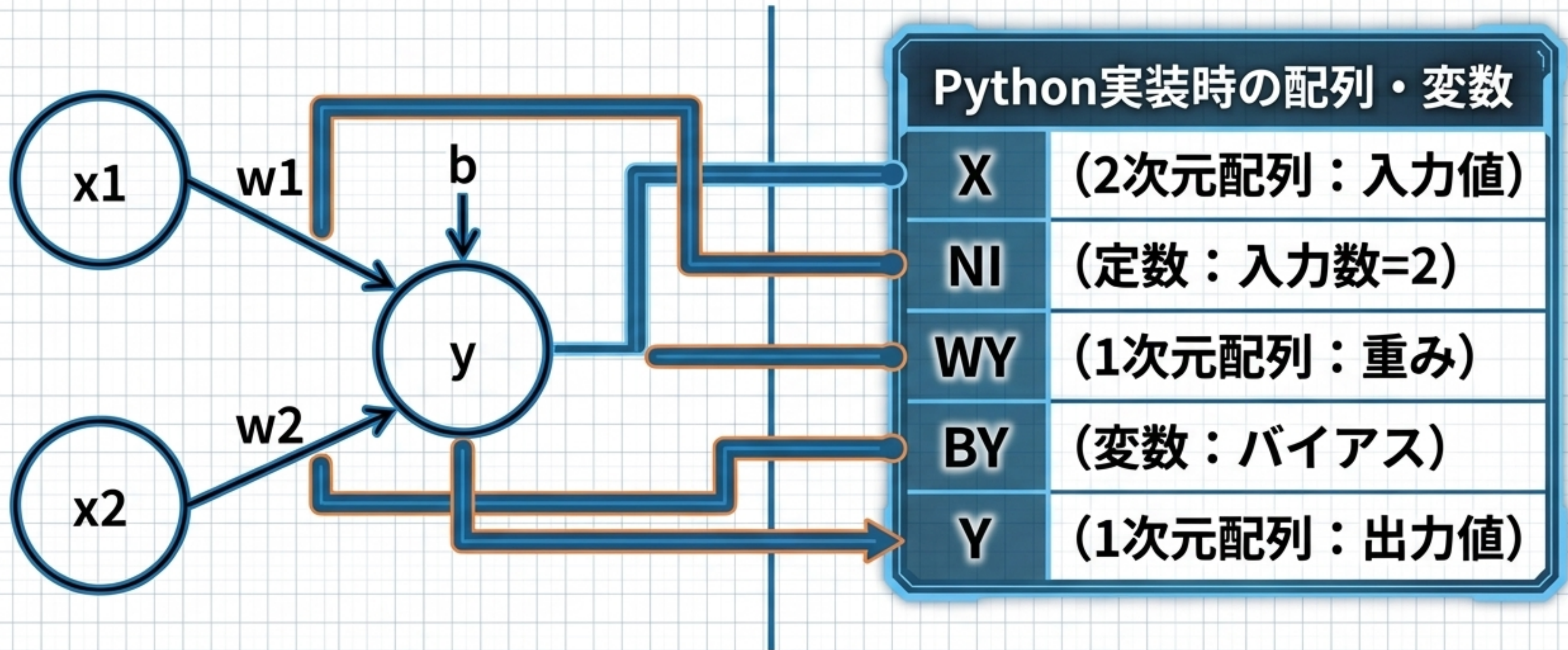


XOR (排他的論理和), EQ (同値)



パーセプトロンの目的は、この境界線を数学的・プログラムの的に引くことにある。

単純パーセプトロンの設計図（数式から変数へ）



ノード演算式： $y = x_1w_1 + x_2w_2 + b$

この計算を、プログラムの配列演算としてどう実装するか？

コード解剖 1：外側のループとバイアスの初期化

```
1 def simple_perceptron(X, Y):  
2     # 入力パターンの数(NC)だけループ  
3     for out in range(NC):  
4         ytemp = BY # 設問3 空欄[ア]
```



空欄[ア] = BY

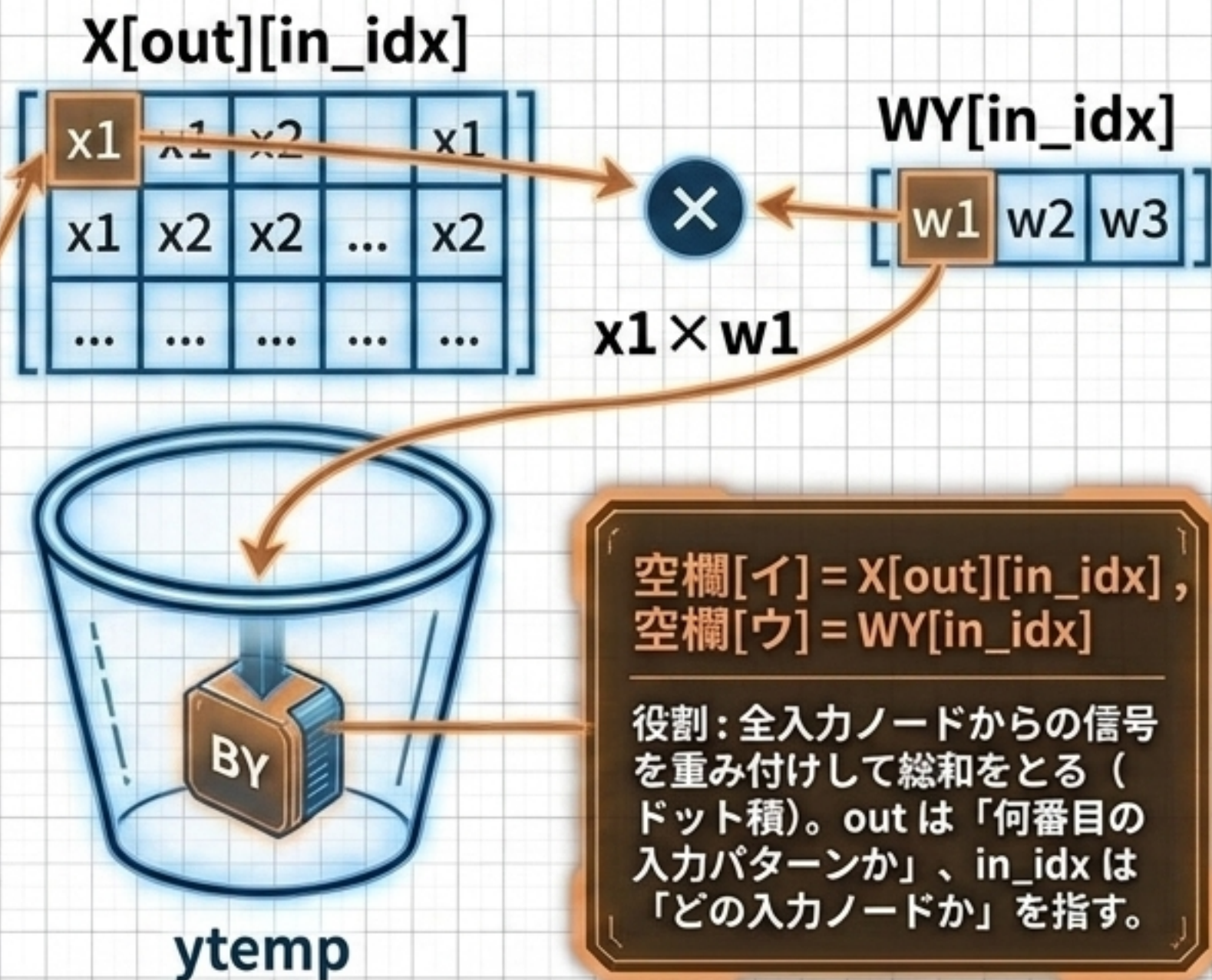
役割：各入力パターンの計算を始める際、出力ノードのベースラインとなる「バイアス値」を最初にセットする（数式 $y = \dots + b$ の b に該当）。



パーセプトロンのコードを解剖し、変数の役割を視覚的に理解する。

コード解剖 2：内側のループと入力の重み付け

```
1 def simple_perceptron(X, Y):  
2     # 入力パターンの数(NC)だけループ  
3     ytemp = BY # 設問3 空欄[ア]  
4     # 入力ノードの数(NI)だけループ  
5     for in_idx in range(NI):  
6         ytemp += X[out][in_idx] * WY[in_idx]
```



パーセプトロンのコードを解剖し、変数の役割を視覚的に理解する。

コード解剖 3：活性化関数 (ステップ関数) による分類

```
1  
2 # ステップ関数 (活性化関数)  
3 if ytemp > 0: # 設問3 空欄[エ]  
4     Y[out] = 1  
5 else:  
6     Y[out] = 0
```



空欄[エ] = > 0

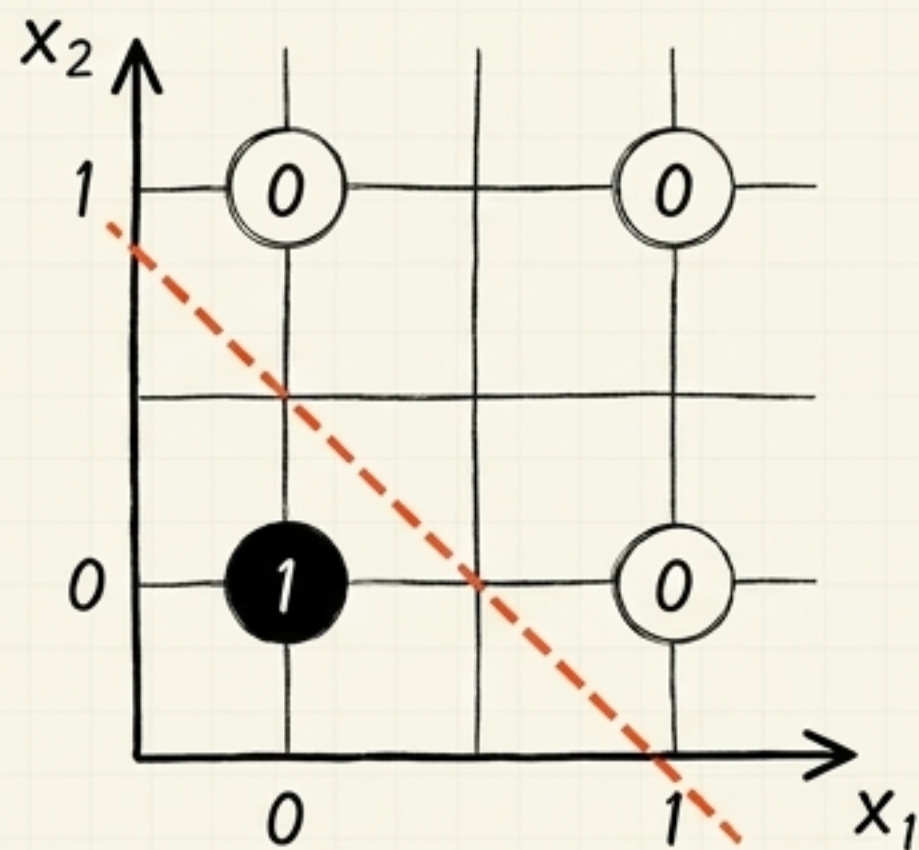
役割：計算された総和が閾値（0）を超えたか判定し、最終的な出力値（0または1）を決定する。ここで線形境界線を越えたかどうかの最終ジャッジが下される。



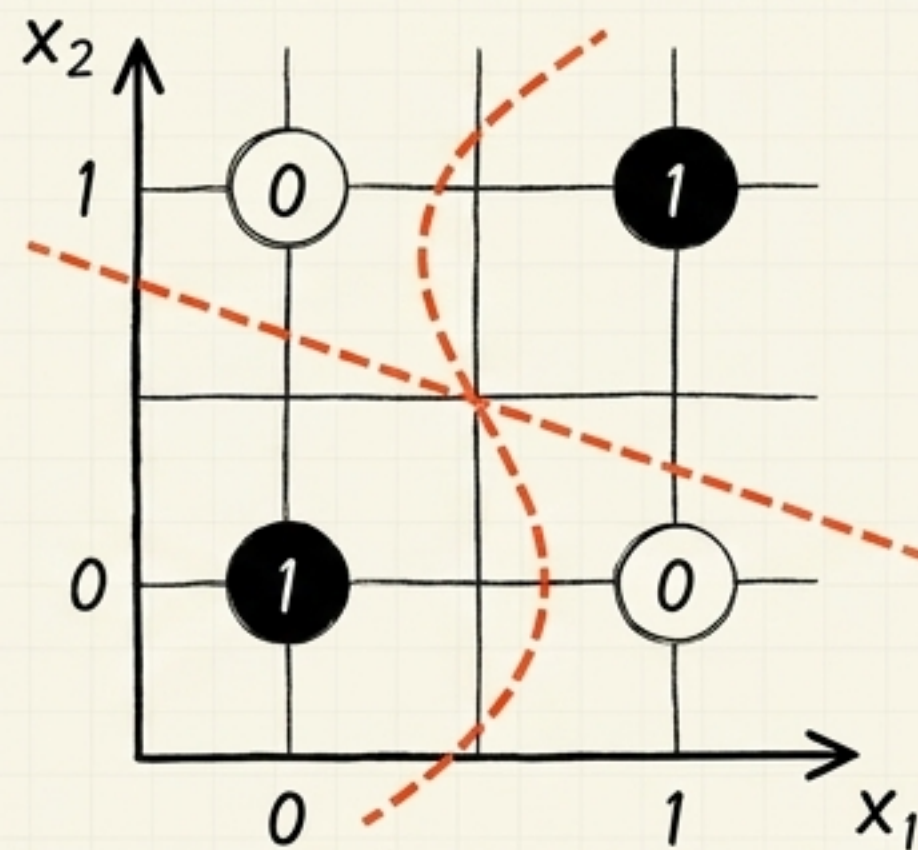
パーセプトロンのコードを解剖し、変数の役割を視覚的に理解する。

限界の露呈：なぜ単純パーセプトロンでは「同値(EQ)」が解けないのか？

否定論理和 (NOR)



同値 (EQ)



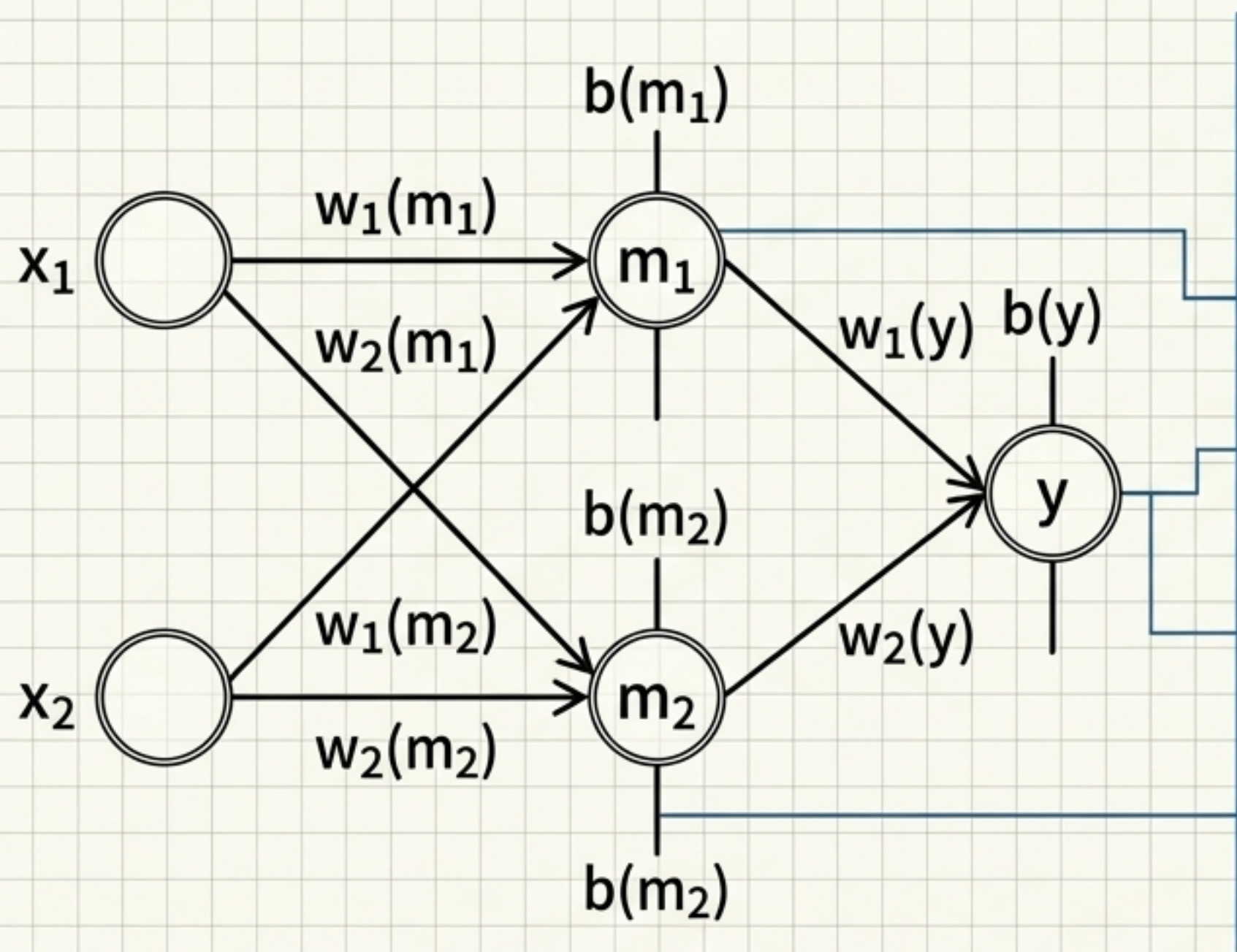
単純パーセプトロン（直線1本）の限界。

2入力の同値 (EQ) や排他的論理和 (XOR) は非線形問題であり、1本の直線では分類できない。（設問4）

解決策 = 空間を歪めるため、「層 (Layer)」を追加する。



3層パーセプトロンの設計図 (新たな変数の導入)

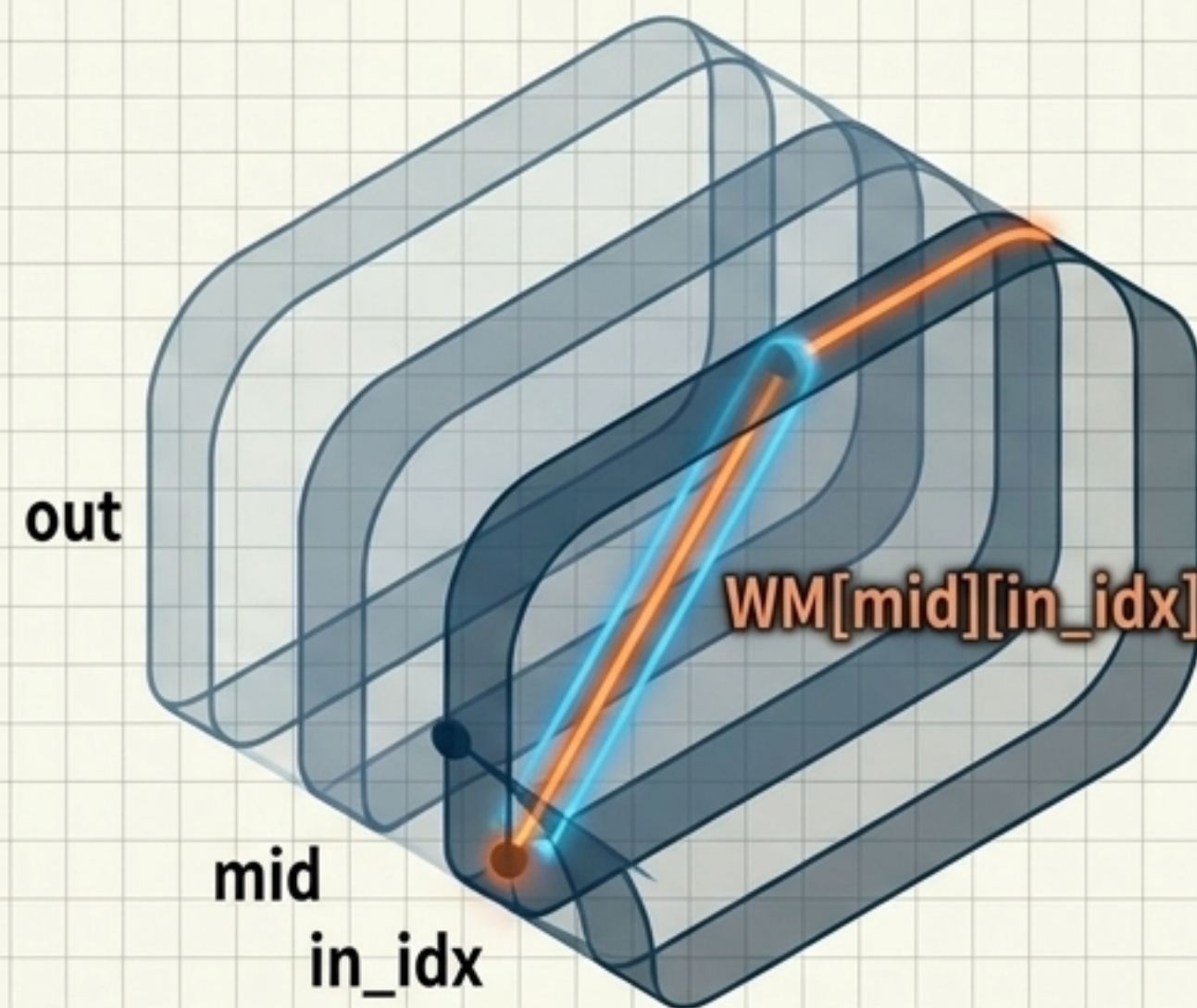


中間層 (Hidden Layer) の追加変数	
NM	(定数: 中間ノード数=2)
M	(2次元配列: $M[out][mid]$)
WM	(2次元配列: $WM[mid][in_idx]$ - 入力から中間への重み)
BM	(1次元配列: 中間層のバイアス)

重要概念: 入力信号を直接判定するのではなく、一度「**隠れた特徴 (m_1, m_2)**」に変換する。このため、**重み WM** が2次元配列に拡張される。

コード解剖 4：中間層の演算（隠れ特徴の抽出）

```
for mid in range(NM):  
    mtemp = BM[mid]  
    for in_idx in range(NI):  
        mtemp += X[out][in_idx]  
        mtemp += X[out][in_idx] * WM[mid][in_idx] # 空欄[オ  
    if mtemp > 0:  
        M[out][mid] = 1 # else 0
```



空欄[オ] = WM[mid][in_idx]

役割: 中間層の演算。単純パーセプトロンと同じドット積の構造を、中間ノードの数 (NM回) だけ並行して実行し、その結果を2次元配列 M にストックする。

コード解剖 5：出力層の演算 (アルゴリズムのフラクタル構造)

```
# 出力層の演算
```

```
for mid in range(NM):
```

```
    ytemp += M[out][mid] * WY[mid] # 空欄[力] * [キ]
```

単純パーセプトロン
公式ブロック

$X[out][in]$

$[in]$

$* WY[in]$

3層出力層
公式ブロック

$M[out][mid]$

$[mid]$

$* WY[mid]$

空欄[力] = $M[out][mid]$, 空欄[キ] = $WY[mid]$
役割: 抽出された中間特徴 (M) を入力として受け取り、最終ノード (y) の出力を計算する。アルゴリズムが同じパターンを繰り返す美しい構造。

アルゴリズムの進化：構造と表現力のトレードオフ

	単純パーセプトロン (Simple)	3層パーセプトロン (3-Layer)
解ける境界線	線形（1本の直線）	非線形（曲線・複数領域）
解決可能な論理演算	AND, OR, NAND, NOR	上記に加え XOR, EQ（同値）に対応
必要なループの深さ	2重ループ（out -> in）	3重ネスト構造（out -> mid -> in）
主要な重み配列の次元数	1次元配列（WY）	2次元配列（WM） + 1次元配列（WY）

結論: 層（ループと配列の次元）を1つ深くすることで、モデルは直線の限界を超え、複雑な論理を獲得する。コードの複雑化は、そのまま**表現力 (Capability)** の拡張を意味している。