Automated Mechanism Design via Neural Networks

Weiran Shen, Pingzhong Tang , Song Zuo

Presented By Shaily Mishra

Problem Addressed

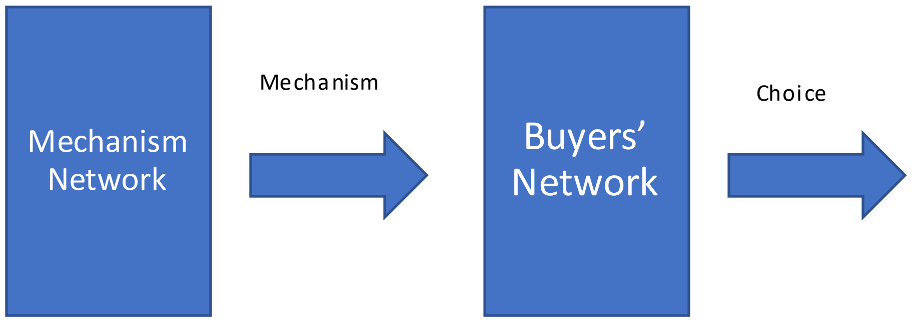
REPRESENTATION

EXACTNESS

DOMAIN DEPENDENCE

Goal

- Multi item, single buyer
- Regardless of game theoretic assumptions
- Without any assumptions on valuations
- Exact IC , IR
- Almost optimal
- Any objective

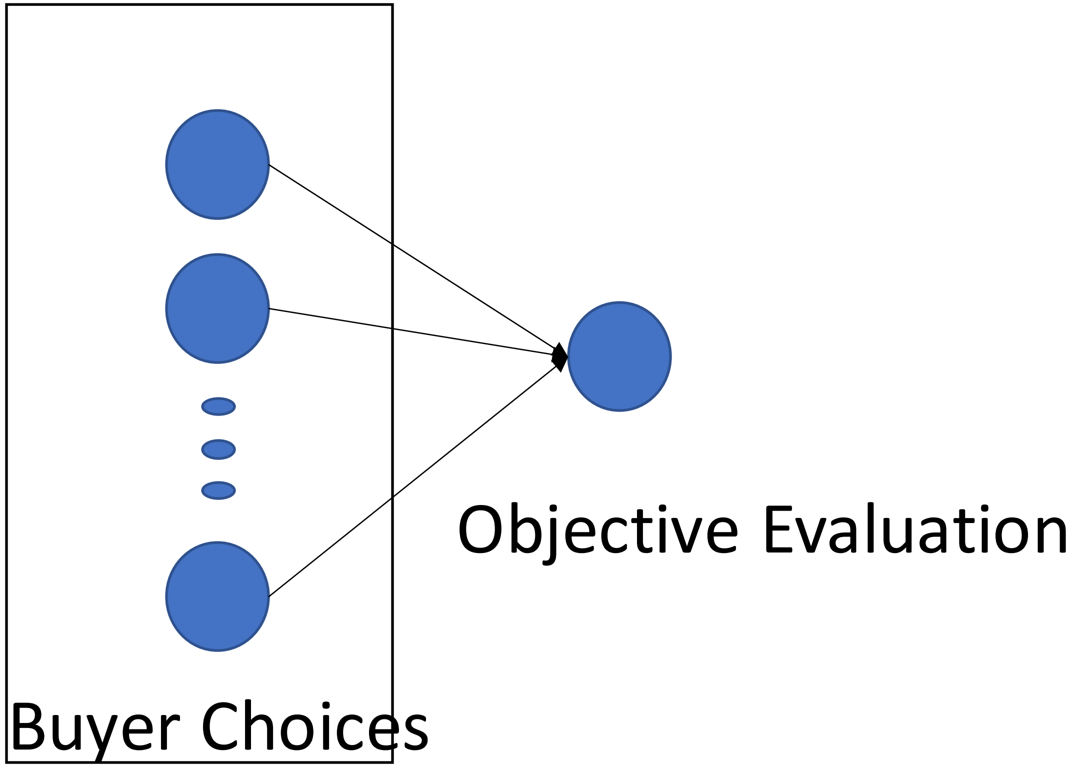


Solution

- Mechanism as menu
- Example of two item menu
  - {(0,0),0}
  - {(0,1),2}
  - {(1,0),3}
  - {(1,1), 5}
- Exact IR - always has a choice of exit

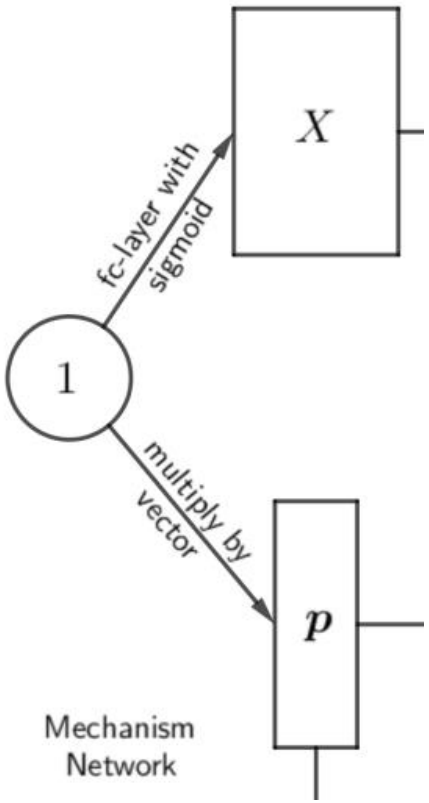
Solution – Buyers Network

- Maps mechanism to strategy
- Encoding of buyer's utility function at different valuations function
- Each node, corresponds to different valuation function, and outputs the menu item that gives maximum utility
- Expected objective is calculated using probabilities of these valuations function, and their choice result
- Generate by assumption or learn from previous data
- Exact IC



Experimental Setup – Mechanism Network

- Allocation matrix *X (mxk)*
- M = #items, k = menu size
- Payment vector ***p*** *(kx1)*
- Row i of X, gives allocation of item i
- Column of X represents menu item




```
For m = 3 , k = 5

-------Allocation Matrix------------------------
tensor([[0., 1., 1., 1., 0.],
        [1., 1., 1., 0., 0.],
        [0., 1., 0., 1., 0.]])
-------Price-------------------------
tensor([0.8172, 0.1233, 0.6187, 0.4816, 0.0000],
```

Experimental Setup –Buyer Network

- Assuming : additive valuation function and independent values

```
for m = 3 and k = 5
Valuations for all 3 items are ~ U(0,1)
For each item , discretize the interval to di discrete values
Let d1 = 2, d2 = 3, d3 = 4
 discrete values for item 1 = {0.5,1}
 discrete values for item 2 = {0.25,0.5,1}
 discrete values for item 3 = {0.125,0.25,0.5,1}
Creating m m-dimensional tensors V1,...,Vm, each of size d1 × d2 × · · · × dm
i.e. Creating 3 3-dimensional tensors V1,V2,V3, each of size 2 × 3 × 4
```

V1

```
tensor([[[0.5000, 0.5000, 0.5000, 0.5000],
         [0.5000, 0.5000, 0.5000, 0.5000],
         [0.5000, 0.5000, 0.5000, 0.5000]],

        [[1.0000, 1.0000, 1.0000, 1.0000],
         [1.0000, 1.0000, 1.0000, 1.0000],
         [1.0000, 1.0000, 1.0000, 1.0000]]])
```

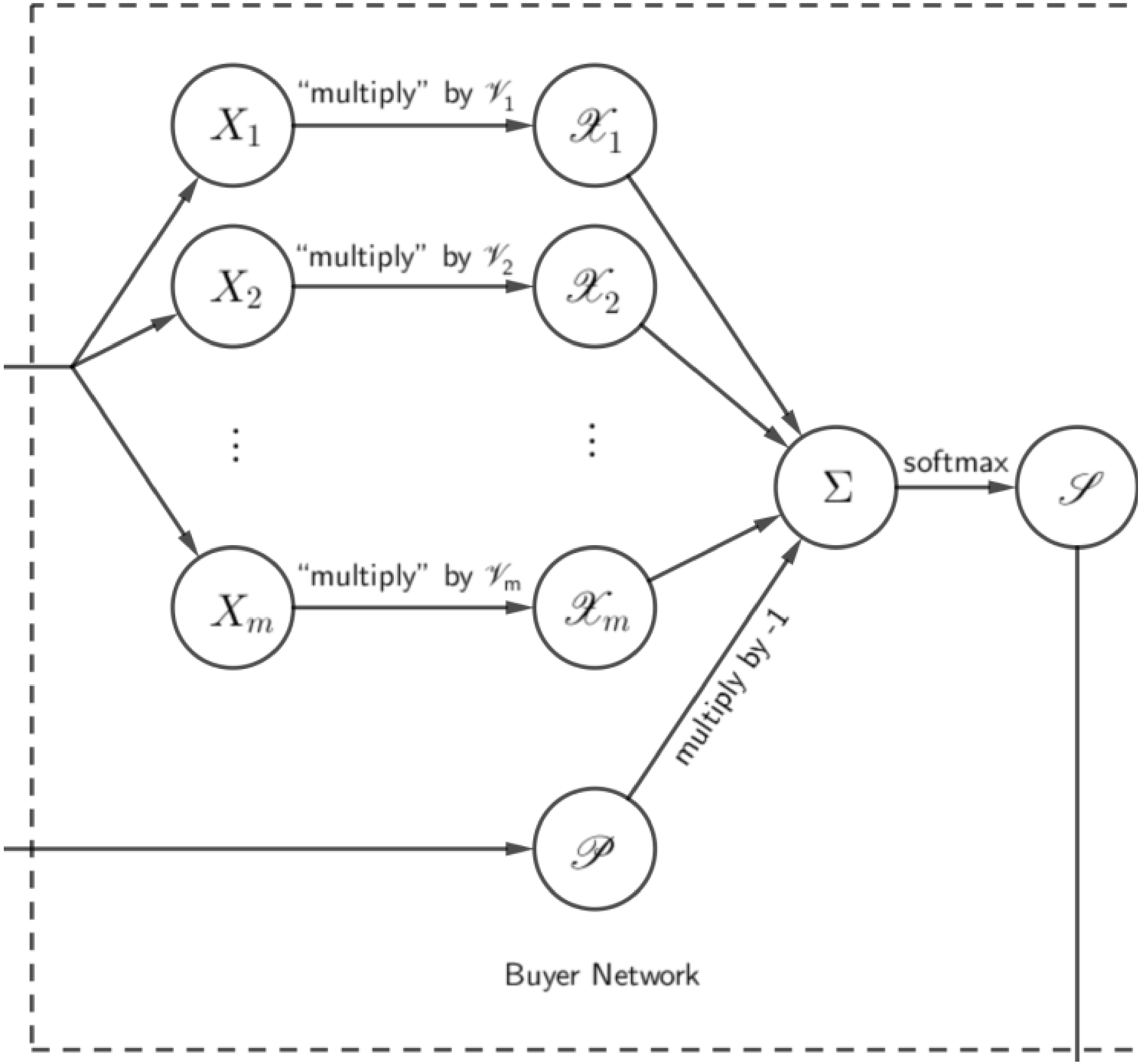
V2

```
tensor([[[0.2500, 0.2500, 0.2500, 0.2500],
         [0.5000, 0.5000, 0.5000, 0.5000],
         [1.0000, 1.0000, 1.0000, 1.0000]],

        [[0.2500, 0.2500, 0.2500, 0.2500],
         [0.5000, 0.5000, 0.5000, 0.5000],
         [1.0000, 1.0000, 1.0000, 1.0000]]])
```

V3

```
tensor([[[0.1250, 0.2500, 0.5000, 1.0000],
         [0.1250, 0.2500, 0.5000, 1.0000],
         [0.1250, 0.2500, 0.5000, 1.0000]],

        [[0.1250, 0.2500, 0.5000, 1.0000],
         [0.1250, 0.2500, 0.5000, 1.0000],
         [0.1250, 0.2500, 0.5000, 1.0000]]])
```

Buyer's Network



Allocation x valuation

Mutiply each Xi(kx1) with Vi(d1×d2×...×dm) => Yi (d1×d2×...×dmxk)

i.e. Yi (2×3×4×5), gives us the value of allocated item i for different valuation

```
Y1
tensor([[[[0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000]],

         [[0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000]],

         [[0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000],
          [0.0000, 0.5000, 0.5000, 0.5000, 0.0000]]],


        [[[0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000]],

         [[0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000]],

         [[0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000],
          [0.0000, 1.0000, 1.0000, 1.0000, 0.0000]]]])
```

```
Y2
tensor([[[[0.2500, 0.2500, 0.2500, 0.0000, 0.0000],
          [0.2500, 0.2500, 0.2500, 0.0000, 0.0000],
          [0.2500, 0.2500, 0.2500, 0.0000, 0.0000],
          [0.2500, 0.2500, 0.2500, 0.0000, 0.0000]],

         [[0.5000, 0.5000, 0.5000, 0.0000, 0.0000],
          [0.5000, 0.5000, 0.5000, 0.0000, 0.0000],
          [0.5000, 0.5000, 0.5000, 0.0000, 0.0000],
          [0.5000, 0.5000, 0.5000, 0.0000, 0.0000]],

         [[1.0000, 1.0000, 1.0000, 0.0000, 0.0000],
          [1.0000, 1.0000, 1.0000, 0.0000, 0.0000],
          [1.0000, 1.0000, 1.0000, 0.0000, 0.0000],
          [1.0000, 1.0000, 1.0000, 0.0000, 0.0000]]],


        [[[0.2500, 0.2500, 0.2500, 0.0000, 0.0000],
          [0.2500, 0.2500, 0.2500, 0.0000, 0.0000],
          [0.2500, 0.2500, 0.2500, 0.0000, 0.0000],
          [0.2500, 0.2500, 0.2500, 0.0000, 0.0000]],

         [[0.5000, 0.5000, 0.5000, 0.0000, 0.0000],
          [0.5000, 0.5000, 0.5000, 0.0000, 0.0000],
          [0.5000, 0.5000, 0.5000, 0.0000, 0.0000],
          [0.5000, 0.5000, 0.5000, 0.0000, 0.0000]],

         [[1.0000, 1.0000, 1.0000, 0.0000, 0.0000],
          [1.0000, 1.0000, 1.0000, 0.0000, 0.0000],
          [1.0000, 1.0000, 1.0000, 0.0000, 0.0000],
          [1.0000, 1.0000, 1.0000, 0.0000, 0.0000]]]])
```

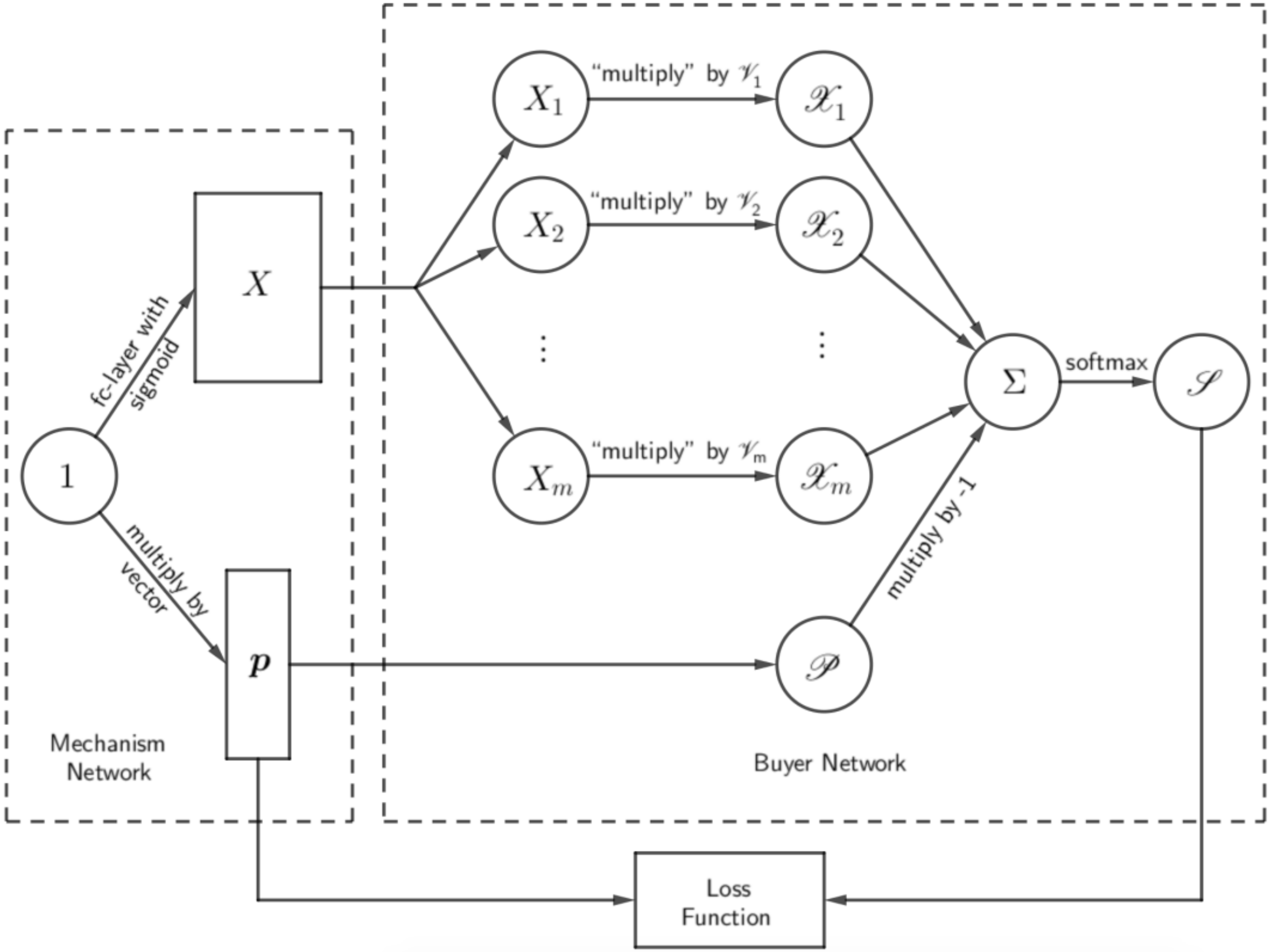
```
Y3
tensor([[[[0.0000, 0.1250, 0.0000, 0.1250, 0.0000],
          [0.0000, 0.2500, 0.0000, 0.2500, 0.0000],
          [0.0000, 0.5000, 0.0000, 0.5000, 0.0000],
          [0.0000, 1.0000, 0.0000, 1.0000, 0.0000]],

         [[0.0000, 0.1250, 0.0000, 0.1250, 0.0000],
          [0.0000, 0.2500, 0.0000, 0.2500, 0.0000],
          [0.0000, 0.5000, 0.0000, 0.5000, 0.0000],
          [0.0000, 1.0000, 0.0000, 1.0000, 0.0000]],

         [[0.0000, 0.1250, 0.0000, 0.1250, 0.0000],
          [0.0000, 0.2500, 0.0000, 0.2500, 0.0000],
          [0.0000, 0.5000, 0.0000, 0.5000, 0.0000],
          [0.0000, 1.0000, 0.0000, 1.0000, 0.0000]]],


        [[[0.0000, 0.1250, 0.0000, 0.1250, 0.0000],
          [0.0000, 0.2500, 0.0000, 0.2500, 0.0000],
          [0.0000, 0.5000, 0.0000, 0.5000, 0.0000],
          [0.0000, 1.0000, 0.0000, 1.0000, 0.0000]],

         [[0.0000, 0.1250, 0.0000, 0.1250, 0.0000],
          [0.0000, 0.2500, 0.0000, 0.2500, 0.0000],
          [0.0000, 0.5000, 0.0000, 0.5000, 0.0000],
          [0.0000, 1.0000, 0.0000, 1.0000, 0.0000]],

         [[0.0000, 0.1250, 0.0000, 0.1250, 0.0000],
          [0.0000, 0.2500, 0.0000, 0.2500, 0.0000],
          [0.0000, 0.5000, 0.0000, 0.5000, 0.0000],
          [0.0000, 1.0000, 0.0000, 1.0000, 0.0000]]]])
```

```
Summation
tensor([[[[0.2500, 0.8750, 0.7500, 0.6250, 0.0000],
          [0.2500, 1.0000, 0.7500, 0.7500, 0.0000],
          [0.2500, 1.2500, 0.7500, 1.0000, 0.0000],
          [0.2500, 1.7500, 0.7500, 1.5000, 0.0000]],

         [[0.5000, 1.1250, 1.0000, 0.6250, 0.0000],
          [0.5000, 1.2500, 1.0000, 0.7500, 0.0000],
          [0.5000, 1.5000, 1.0000, 1.0000, 0.0000],
          [0.5000, 2.0000, 1.0000, 1.5000, 0.0000]],

         [[1.0000, 1.6250, 1.5000, 0.6250, 0.0000],
          [1.0000, 1.7500, 1.5000, 0.7500, 0.0000],
          [1.0000, 2.0000, 1.5000, 1.0000, 0.0000],
          [1.0000, 2.5000, 1.5000, 1.5000, 0.0000]]],


        [[[0.2500, 1.3750, 1.2500, 1.1250, 0.0000],
          [0.2500, 1.5000, 1.2500, 1.2500, 0.0000],
          [0.2500, 1.7500, 1.2500, 1.5000, 0.0000],
          [0.2500, 2.2500, 1.2500, 2.0000, 0.0000]],

         [[0.5000, 1.6250, 1.5000, 1.1250, 0.0000],
          [0.5000, 1.7500, 1.5000, 1.2500, 0.0000],
          [0.5000, 2.0000, 1.5000, 1.5000, 0.0000],
          [0.5000, 2.5000, 1.5000, 2.0000, 0.0000]],

         [[1.0000, 2.1250, 2.0000, 1.1250, 0.0000],
          [1.0000, 2.2500, 2.0000, 1.2500, 0.0000],
          [1.0000, 2.5000, 2.0000, 1.5000, 0.0000],
          [1.0000, 3.0000, 2.0000, 2.0000, 0.0000]]]])
```

- Each entry (i,j,k,l) :
  - i -> valuation of item 1
  - j -> valuation of item 2
  - k -> valuation of item 3
  - l -> menu item
- Entry for (0,0,0,0) corresponds to (0.5,0.25,0.125) item valuation and allocation (0,1,0) hence the value obtained is 0.25.
- Similarly entry (0,0,0,1) corresponds to (0.5,0.25,0.125) item valuation and allocation (1,1,1), hence the value obtained is 0.875.

```
Taking softmax of utility across menu item
tensor([[[[0.1933, 0.3003, 0.2214, 0.1344, 0.1506],
          [0.1828, 0.3216, 0.2092, 0.1440, 0.1424],
          [0.1614, 0.3648, 0.1848, 0.1633, 0.1257],
          [0.1202, 0.4479, 0.1377, 0.2005, 0.0937]],

         [[0.2063, 0.3205, 0.2363, 0.1117, 0.1252],
          [0.1951, 0.3434, 0.2234, 0.1197, 0.1184],
          [0.1724, 0.3897, 0.1974, 0.1358, 0.1046],
          [0.1286, 0.4791, 0.1472, 0.1670, 0.0780]],

         [[0.2276, 0.3534, 0.2605, 0.0747, 0.0837],
          [0.2153, 0.3789, 0.2465, 0.0801, 0.0792],
          [0.1905, 0.4304, 0.2181, 0.0910, 0.0701],
          [0.1423, 0.5302, 0.1629, 0.1121, 0.0524]]],


        [[[0.1356, 0.3473, 0.2560, 0.1555, 0.1057],
          [0.1271, 0.3688, 0.2399, 0.1651, 0.0990],
          [0.1104, 0.4112, 0.2083, 0.1841, 0.0860],
          [0.0796, 0.4891, 0.1503, 0.2189, 0.0620]],

         [[0.1439, 0.3686, 0.2717, 0.1285, 0.0873],
          [0.1350, 0.3917, 0.2548, 0.1366, 0.0819],
          [0.1174, 0.4373, 0.2216, 0.1525, 0.0712],
          [0.0849, 0.5215, 0.1603, 0.1818, 0.0515]],

         [[0.1573, 0.4028, 0.2969, 0.0852, 0.0579],
          [0.1477, 0.4286, 0.2788, 0.0906, 0.0544],
          [0.1287, 0.4796, 0.2430, 0.1014, 0.0474],
          [0.0935, 0.5742, 0.1765, 0.1214, 0.0344]]]],
```

- Calculate the utility, and then take softmax
- For each value profile, choose the strategy that gives highest weight



- Objective : Maximize revenue

$$\text{Loss} = -\text{Rev} = -\sum_{v\in V} \Pr[v]\,\boldsymbol{p}^T\mathbf{s}(v),$$

Unified Network

Evaluation

- Uniform $[0,c] \times [0,1]$
- Correlated Distribution: Uniform Triangle (Proves theoretical result)
- Restricted Menu Size (Proves theoretical result)
- Unit-Demand Buyer
- Combinatorial Value
- Deterministic Mechanisms