A Proofs of Lemma 2-4

In Sec. 3, we study the geometric structure of optimal contracts by establishing four lemmas. While some of them have been stated and proved for linear contracts [32], and the first two lemmas, at least, are not surprising, we give formal proofs of these properties of the principal's utility and optimal contracts in the general case. An important property of the principal's utility function u^p that we state in Lemma 2 is that u^p can be a discontinuous function.

Lemma 2. *The principal's utility function u^p can be discontinuous on the boundary of linear pieces.*

Proof. For a contract $\boldsymbol{f}$ on the boundary of two neighboring linear pieces μ_i^p and $\mu_{j\neq i}^p$, the agent is indifferent between action a_i and a_j given $\boldsymbol{f}$: $\mu_i^a(\boldsymbol{f}) = \mu_j^a(\boldsymbol{f})$. The principal's utility

$$\begin{aligned}
\mu_i^p(\boldsymbol{f}) &= \mathbb{E}_{o\sim p(\cdot|a_i)}[v_o] - \mathbb{E}_{o\sim p(\cdot|a_i)}[f_o] && (9)\\
&= \mathbb{E}_{o\sim p(\cdot|a_i)}[v_o] - c(a_i) - (\mathbb{E}_{o\sim p(\cdot|a_i)}[f_o] - c(a_i)) && (10)\\
&= \mathbb{E}_{o\sim p(\cdot|a_i)}[v_o] - c(a_i) - \mu_i^a(\boldsymbol{f}) && (11)\\
&= \mathbb{E}_{o\sim p(\cdot|a_i)}[v_o] - c(a_i) - \mu_j^a(\boldsymbol{f}) && (12)\\
&= \mu_j^p(\boldsymbol{f}) + \mathbb{E}_{o\sim p(\cdot|a_i)}[v_o] - c(a_i) - (\mathbb{E}_{o\sim p(\cdot|a_j)}[v_o] - c(a_j)). && (13)
\end{aligned}$$

It is possible that

$$\mathbb{E}_{o\sim p(\cdot|a_i)}[v_o] - c(a_i) \neq \mathbb{E}_{o\sim p(\cdot|a_j)}[v_o] - c(a_j). \quad (14)$$

Function u^p is discontinuous in this case. □

Another property of the principal's utility function u^p that motivates our discontinuous neural networks is Lemma 3.

Lemma 3. *The global optimal contract is on the boundary of a linear piece.*

Proof. Suppose that the global optimal $\boldsymbol{f}^*$ is not on a boundary but is an interior point of a linear piece $\mathcal{Q}_i$ (contracts in $\mathcal{Q}_i$ encourage the agent to take action a_i.):

$$\boldsymbol{f}^* \in \mathcal{Q}_i - \partial\mathcal{Q}_i, \quad (15)$$

where

$$\begin{aligned}
\mathcal{Q}_i &= \cap_{j\neq i}\Gamma_{i,j}; && (16)\\
\Gamma_{i,j} &= \left\{\boldsymbol{f}\in\mathcal{F} \mid \mathbb{E}_{o\sim p(\cdot|a_i)}\left[f_o\right] - c(a_i) \geq \mathbb{E}_{o\sim p(\cdot|a_j)}\left[f_o\right] - c(a_j)\right\}, \forall j\neq i. && (17)
\end{aligned}$$

It follows that

$$\boldsymbol{f}^* \in \left\{\boldsymbol{f}\in\mathcal{F} \mid \mathbb{E}_{o\sim p(\cdot|a_i)}\left[f_o\right] - c(a_i) > \mathbb{E}_{o\sim p(\cdot|a_j)}\left[f_o\right] - c(a_j), \forall j\neq i\right\}, \quad (18)$$

where the inequality is strict. Let

$$\epsilon_{i,j} = \mathbb{E}_{o\sim p(\cdot|a_i)}\left[f_o^*\right] - c(a_i) - \mathbb{E}_{o\sim p(\cdot|a_j)}\left[f_o^*\right] + c(a_j). \quad (19)$$

It holds that $\epsilon_{i,j} > 0, \forall j\neq i$.

Now we consider the contract $\boldsymbol{f}' = \boldsymbol{f}^* - \delta p(\cdot|a_i)$ for some small $\delta > 0$. For any $a_j \neq a_i$, we have

$$\begin{aligned}
&u^a(\boldsymbol{f}';a_i) - u^a(\boldsymbol{f}';a_j)\\
=&\mathbb{E}_{o\sim p(\cdot|a_i)}\left[f_o^* - \delta p(o|a_i)\right] - c(a_i) - \mathbb{E}_{o\sim p(\cdot|a_j)}\left[f_o^* - \delta p(o|a_i)\right] + c(a_j) && (20)\\
=&\epsilon_{i,j} - \delta\mathbb{E}_{o\sim p(\cdot|a_i)}\left[p(o|a_i)\right] + \delta\mathbb{E}_{o\sim p(\cdot|a_j)}\left[p(o|a_i)\right].
\end{aligned}$$

When

$$0 < \delta < \min_{j\neq i}\frac{\epsilon_{i,j}}{\mathbb{E}_{o\sim p(\cdot|a_i)}\left[p(o|a_i)\right] - \mathbb{E}_{o\sim p(\cdot|a_j)}\left[p(o|a_i)\right]}, \quad (21)$$

(where note the denominator is > 0), we have

$$u^a(\boldsymbol{f}';a_i) - u^a(\boldsymbol{f}';a_j) > 0, \forall j\neq i, \quad (22)$$

which means $\boldsymbol{f}'$ incentivizes the agent to take action a_i. Therefore, for δ in this range, the principal's utility given $\boldsymbol{f}'$ is:

$$\begin{aligned} u^p(\boldsymbol{f}') &= \mathbb{E}_{o\sim p(\cdot|a_i)}\left[v_o - f_o^* + \delta p(o|a_i)\right] \\ &= \mathbb{E}_{o\sim p(\cdot|a_i)}\left[v_o - f_o^*\right] + \mathbb{E}_{o\sim p(\cdot|a_i)}\left[\delta p(o|a_i)\right] \\ &= u^p(\boldsymbol{f}^*) + \delta \sum_o p^2(o|a_i) \\ &> u^p(\boldsymbol{f}^*). \end{aligned} \tag{23}$$

We thus find a contract $\boldsymbol{f}'$ that induces greater utility for the principal, contradicting with the fact that $\boldsymbol{f}^*$ is the global optimal contract. This finishes the proof.

Note that here we consider the boundary resulting from changes in the agent's best responses. We can extend the proof to cover another type of boundary, which pertains to the requirement that contracts are non-negative, by defining $\mathcal{Q}_i$ to be $\mathcal{Q}_i = \{\boldsymbol{f}|\boldsymbol{f} \geq 0\} \cap \Gamma_{i,1} \cap \cdots \cap \Gamma_{i,i-1} \cap \Gamma_{i,i+1} \cap \cdots$. □

Lemma 4 claims another property that may influence the design of DeLU networks.

Lemma 4. *The principal's utility function u^p can be written as a summation of a concave function and a piecewise constant function.*

Proof. We first prove the agent's utility function $u^a(\boldsymbol{f})$ is a convex function.

We need to prove that, for any two contracts $\boldsymbol{f}^{(1)} \in \mathcal{F}$ and $\boldsymbol{f}^{(2)} \in \mathcal{F}$, it holds that $u^p(\lambda\boldsymbol{f}^{(1)} + (1-\lambda)\boldsymbol{f}^{(2)}) \leq \lambda u^p(\boldsymbol{f}^{(1)}) + (1-\lambda)u^p(\boldsymbol{f}^{(2)}), \forall\lambda \in [0,1]$. Denote $\boldsymbol{d} = \boldsymbol{f}^{(2)} - \boldsymbol{f}^{(1)}$. It suffices to prove that the derivative of $u^p(\boldsymbol{f}^{(1)} + \delta\boldsymbol{d})$ with respect to δ is a non-decreasing function for $\delta \in [0,1]$.

We have

$$\frac{\partial}{\partial\delta}u^a(\boldsymbol{f}^{(1)} + \delta\boldsymbol{d}) = \frac{\partial}{\partial\delta}\left[\mathbb{E}_{o\sim p(\cdot|a_\delta)}[f_o^{(1)} + \delta d_o] - c(a_\delta)\right], \tag{24}$$

where a_δ is the agent's action given the contract $\boldsymbol{f}^{(1)} + \delta\boldsymbol{d}$.

Case 1: When a_δ does not change,

$$\frac{\partial}{\partial\delta}u^a(\boldsymbol{f}^{(1)} + \delta\boldsymbol{d}) = \mathbb{E}_{o\sim p(\cdot|a_\delta)}[d_o] \tag{25}$$

is a constant, which is a non-decreasing function.

Case 2: When a_δ changes. Suppose that there exists $\delta_1 \in [0,1]$ such that $\boldsymbol{f}^{(1)} + \delta_1\boldsymbol{d}$ is on the boundary of linear piece $\mathcal{Q}_i$ and $\mathcal{Q}_j$. Let

$$\begin{aligned} \delta_1^+ &= \delta_1 + \epsilon, \\ \delta_1^- &= \delta_1 - \epsilon, \end{aligned} \tag{26}$$

where ϵ is a small number and

$$\begin{aligned} \boldsymbol{f}^{(1)} + \delta_1^-\boldsymbol{d} &\in \mathcal{Q}_i, \\ \boldsymbol{f}^{(1)} + \delta_1^+\boldsymbol{d} &\in \mathcal{Q}_j. \end{aligned} \tag{27}$$

Because the agent is self-interested, it follows that a_j is the best response when the contract is $\boldsymbol{f}^{(1)} + \delta_1^+\boldsymbol{d}$:

$$u^a(\boldsymbol{f}^{(1)} + \delta_1^+\boldsymbol{d}) = u^a(\boldsymbol{f}^{(1)} + \delta_1^+\boldsymbol{d}; a_j) > u^a(\boldsymbol{f}^{(1)} + \delta_1^+\boldsymbol{d}; a_i). \tag{28}$$

It follows that

$$\frac{\partial}{\partial\delta}u^a(\boldsymbol{f}^{(1)} + \delta\boldsymbol{d})\bigg|_{\delta=\delta_1^+} = \lim_{\epsilon\to 0}\frac{u^a(\boldsymbol{f}^{(1)} + \delta_1^+\boldsymbol{d}) - u^a(\boldsymbol{f}^{(1)} + \delta_1\boldsymbol{d})}{\epsilon} \tag{29}$$

$$> \lim_{\epsilon\to 0}\frac{u^a(\boldsymbol{f}^{(1)} + \delta_1^+\boldsymbol{d}; a_i) - u^a(\boldsymbol{f}^{(1)} + \delta_1\boldsymbol{d})}{\epsilon}. \tag{30}$$

We now look at the two terms in the numerator of Eq. 30:

$$u^a(\boldsymbol{f}^{(1)} + \delta_1^+ \boldsymbol{d}; a_i) = \mathbb{E}_{o\sim p(\cdot|a_i)}[f_o^{(1)} + \delta_1^+ d_o] - c(a_i) \tag{31}$$
$$= \mathbb{E}_{o\sim p(\cdot|a_i)}[f_o^{(1)} + (\delta_1 + \epsilon) d_o] - c(a_i) \tag{32}$$
$$= \mathbb{E}_{o\sim p(\cdot|a_i)}[f_o^{(1)} + \delta_1 d_o] - c(a_i) + \epsilon \mathbb{E}_{o\sim p(\cdot|a_i)}[d_o] \tag{33}$$
$$= u^a(\boldsymbol{f}^{(1)} + \delta_1 \boldsymbol{d}) + \epsilon \mathbb{E}_{o\sim p(\cdot|a_i)}[d_o], \tag{34}$$

and

$$u^a(\boldsymbol{f}^{(1)} + \delta_1 \boldsymbol{d}) = \mathbb{E}_{o\sim p(\cdot|a_i)}[f_o^{(1)} + \delta_1 d_o] - c(a_i) \tag{35}$$
$$= \mathbb{E}_{o\sim p(\cdot|a_i)}[f_o^{(1)} + (\delta_1^- + \epsilon) d_o] - c(a_i) \tag{36}$$
$$= \mathbb{E}_{o\sim p(\cdot|a_i)}[f_o^{(1)} + \delta_1^- d_o] - c(a_i) + \epsilon \mathbb{E}_{o\sim p(\cdot|a_i)}[d_o] \tag{37}$$
$$= u^a(\boldsymbol{f}^{(1)} + \delta_1^- \boldsymbol{d}) + \epsilon \mathbb{E}_{o\sim p(\cdot|a_i)}[d_o]. \tag{38}$$

Therefore,

$$\begin{aligned} \frac{\partial}{\partial \delta} u^a(\boldsymbol{f}^{(1)} + \delta \boldsymbol{d})\Big|_{\delta=\delta_1^+} &> \lim_{\epsilon\to 0} \frac{u^a(\boldsymbol{f}^{(1)} + \delta_1^+ \boldsymbol{d}; a_i) - u^a(\boldsymbol{f}^{(1)} + \delta_1 \boldsymbol{d})}{\epsilon} \\ &= \lim_{\epsilon\to 0} \frac{u^a(\boldsymbol{f}^{(1)} + \delta_1 \boldsymbol{d}) - u^a(\boldsymbol{f}^{(1)} + \delta_1^- \boldsymbol{d})}{\epsilon} \\ &= \frac{\partial}{\partial \delta} u^a(\boldsymbol{f}^{(1)} + \delta \boldsymbol{d})\Big|_{\delta=\delta_1^-}. \end{aligned} \tag{39}$$

This finishes the proof that u^a is a convex function. Furthermore, we have that

$$u^p(\boldsymbol{f}) = -u^a(\boldsymbol{f}) - c(a^*(\boldsymbol{f})) + \mathbb{E}_{o\sim p(\cdot|a^*(\boldsymbol{f}))}[v_o], \tag{40}$$

where $-u^a(\boldsymbol{f})$ is a concave function and $-c(a^*(\boldsymbol{f})) + \mathbb{E}_{o\sim p(\cdot|a^*(\boldsymbol{f}))}[v_o]$ is a piecewise constant function with the value $-c(a_i) + \mathbb{E}_{o\sim p(\cdot|a_i)}[v_o]$ when $\boldsymbol{f} \in \mathcal{Q}_i$. □

B Why we use another network to generate the last-layer bias?

To model model the dependency of the last-layer bias on the activation pattern, we use a neural network, rather than a simpler, linear, and learnable function. The reason is that the bias does not always depend linearly on the activation pattern. Here is an example to illustrate this. There are two outcomes with values $\mathbf{v} = [20, 1]$, four actions with costs $\mathbf{c} = [1.0, 2.1, 2.3, 4.7]$, and the action-outcome transition kernel is

$$P = \begin{bmatrix} 0.211 & 0.789 \\ 0.398 & 0.602 \\ 0.430 & 0.570 \\ 0.684 & 0.316 \end{bmatrix}.$$

Suppose we consider linear contracts, where $\mathbf{f} = \alpha \mathbf{v}, \alpha > 0$. Then the principal's utility function for different contracts is:

$$u^p(\alpha) = \begin{cases} -5\alpha + 5 & 0.2 < \alpha < 0.3 \\ -8.57\alpha + 8.57 & 0.3 < \alpha < 0.4 \\ -9.17\alpha + 9.17 & 0.4 < \alpha < 0.5 \\ -14\alpha + 14 & \alpha > 0.5 \end{cases}.$$

Suppose that we have a 2-dimensional activation pattern, and the linear function converting activation patterns to the bias has parameters $[b_1, b_2]$. Then the bias for each of the four pieces would be 0, b_1, b_2, and $b_1 + b_2$, respectively. The difference between each piece's bias needs to model the discontinuity at contract parameter $\alpha = 0.3, 0.4, 0.5$, but this is impossible with this linear model. To see this, we first assume that the piece $0.2 < \alpha < 0.3$ has bias 0. Then the differences of biases of the other 3 pieces would need to be 2.5, 2.86, and 5.28, which cannot be achieved with b_1, b_2, and $b_1 + b_2$. It can be easily verified that the cases where other pieces have a bias of 0 are similar, demonstrating that a linear bias function cannot express the discontinuity. By contrast, appealing to a second network allows for non-linear dependency on activation, and can handle this problem.

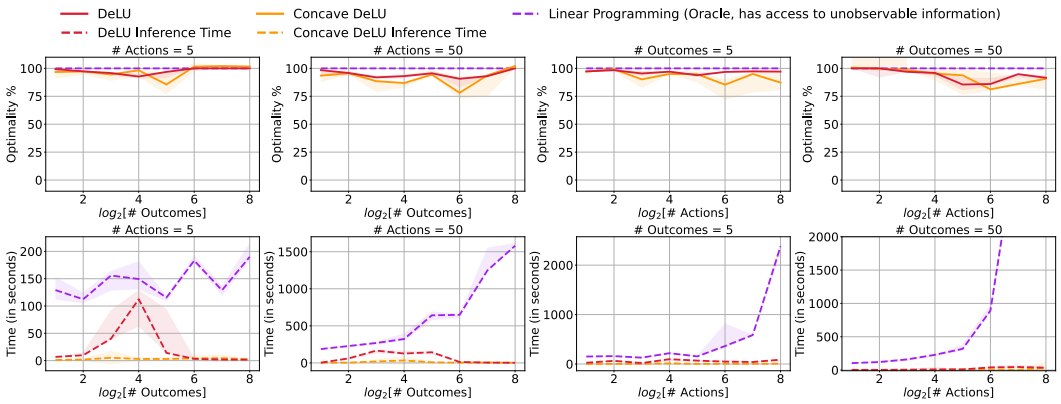

Figure 8: Optimality (normalized principal utility, divided by the result obtained by the direct LP solver `Oracle LP`) and inference time of DeLU, `Concave DeLU`, and `Oracle LP` on increasing problem sizes.

C Introducing Concavity into the DeLU Network

From Lemma 4, the principal's utility function u^p is a summation of a concave function and a piecewise constant function. Further, our DeLU network, which is used to approximate the function u^p, can be written as

$$\xi(\boldsymbol{f}_i;\theta_\eta,\theta_\zeta) = \eta(\boldsymbol{f}_i;\theta_\eta) + \zeta(r(\boldsymbol{f}_i);\theta_\zeta). \tag{41}$$

In particular, $\zeta(r(\boldsymbol{f}_i);\theta_\zeta)$ is a piecewise constant function, because given an activation pattern $r(\boldsymbol{f}_i)$, ζ is a constant. However, the first term in Eq. 41, in a general DeLU network, is an arbitrary function. This raises the question as to whether it is useful to further restrict the network architecture, constraining the sub-network η to be a concave function. In this section, we discuss how to introduce concavity into η, and how this restriction affects the performance of DeLU-based contract design.

C.1 Concave DeLU architecture

We can make η a concave function by (1) enforcing its weights (for all but the first layer) to be non-negative, and (2) taking the negation of its output. The first modification will make η a convex function because η uses ReLU activation, which is a convex and non-decreasing function. When the weights after a ReLU activation are non-negative, η becomes a composition of several convex, non-decreasing functions, which is still a convex function. Since the negation of a convex function is a concave function, the second modification will make η concave.

Formally, in `Concave DeLU`, the sub-network η calculates

$$\begin{aligned}\boldsymbol{h}^{(L)}(\boldsymbol{x}) &= |\boldsymbol{W}^{(L)}|\boldsymbol{R}^{(L-1)}(\boldsymbol{x})\left(\cdots\left(|\boldsymbol{W}^{(2)}|\boldsymbol{R}^{(1)}(\boldsymbol{x})\left(\boldsymbol{W}^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)}\right)+\boldsymbol{b}^{(2)}\right)\cdots\right)+\boldsymbol{b}^{(L)},\\ \eta(\boldsymbol{x}) &= -|\boldsymbol{W}^{(L+1)}|\boldsymbol{R}^{(L)}\boldsymbol{h}^{(L)}(\boldsymbol{x}),\end{aligned} \tag{42}$$

where $\boldsymbol{R}^{(k)}$ is a diagonal matrix with diagonal elements equal to the layer activation pattern $\boldsymbol{r}^{(k)}$. The other components, as well as the training process, of `Concave DeLU` are the same as a DeLU network. In the next sub-section, we evaluate the performance of `Concave DeLU`.

C.2 Experiments on Concave DeLU networks

In Fig. 8, we compare `Concave DeLU` against DeLU and the direct LP solver (`Oracle LP`). For this, we fix DeLU and `Concave DeLU` to have the same size, and both use the LP inference algorithm. We test different problem sizes. In the first and second column, we set the number of actions n to 5 and 50, respectively, and increase the number of outcomes m from 2^1 to 2^8. In the third and fourth column, we set the number of outcomes m to 5 and 50, respectively, and increase the number of actions n from 2^1 to 2^8. For each problem size, the same 12 combinations of α_p and β_p are tested. The median

performance as well as the first and third quartile (shaped area) of these 12 combinations are shown. The first row compares optimality, while the second row compares inference time efficiency. We again report normalized principal utilities when it comes to optimality.

A surprising result is that for most problem sizes, DeLU achieves better optimality than `Concave DeLU`. Although the function class represented by `Concave DeLU` is a better fit with u^p, it seems that the larger function class of DeLU aids optimization and leads to better performance. At the same time, it is somewhat surprising that `Concave DeLU` can reduce inference time substantially. Unlike DeLU, the inference time of `Concave DeLU` remains relatively stable when the problem size increases. We conjecture that this behavior can be attributed to the non-negativity constraint on network weights, which reduces the number of valid activation patterns and speeds up LP inference.

D Experimental Setups

D.1 Network architecture and training

We use a simple architecture for the DeLU network. In all experiments, the sub-network η has one hidden layer with 32 hidden units and ReLU activations. We deliberately restrict the size of this sub-network to limit the number of valid activation patterns and speedup LP-based inference. However, this restriction may reduce the representational capacity of DeLU networks: it is in contrast to the common practice of overparameterization, which has contributed to the success of deep learning. To alleviate this concern, we employ a relatively larger network for the bias network ζ. In our experiments, ζ has a hidden layer with 512 (Tanh-activated) neurons.

The two sub-networks η and ζ are trained in an end-to-end manner by the MSE loss (Eq. 5). The optimization is conducted using RMSprop with a learning rate of 1×10^{-3}, α of 0.99, and with no momentum or weight decay. For the DeLU and the baseline ReLU networks, training samples are randomly shuffled in each of the training epochs.

D.2 Infrastructure

Across all experiments, DeLU, and the baseline ReLU networks are trained on a NVIDIA A100 GPU. Gradient-based inference is also parallelized on the A100 GPU. The direct LP solver (`Oracle LP`) and the LP-based inference algorithm are based on the linear programming toolkit PuLP [47], and we parallelize five LP solvers on CPUs.

E More Experiments

In this section, we carry out experiments to study (1) using random samples to initialize gradient-based inference (Appx. E.1); and (2) the influence of cost correlation on the optimality of DeLU learners (Appx. E.2).

E.1 Gradient-based inference initialized with random samples

In Sec. 4.2.2, we introduce a gradient-based inference algorithm, and Alg. 1 gives the matrix-form expression of its parallel implementation. There is a choice in using Alg. 1, as to whether the input ("probe points") $\boldsymbol{X}^{(0)} \in \mathbb{R}^{K \times m}$ is taken from the training set or a different, random sample set. In this section, we report the results of experiments to empirically compare these two setups.

We start from the same trained DeLU networks on small ($n = 5, m = 16$), middle ($n = 32, m = 50$), and large ($n = 50, m = 128$) problem sizes, where n is the number of actions and m is the number of outcomes. For each problem size, we consider 12 different combinations of (α_p, β_p) as in other experiments and report the mean and variance of the performance. We run Alg. 1 with two different inputs $\boldsymbol{X}^{(0)}$: `Training Set` uses $50K$ training samples while `Random Set` uses $50K$ randomly generated contracts. In Table. 1, we present the normalized principal utility (divided by the result obtained by `Oracle LP`) of these two setups.

We can observe that the optimality of these two setups are very close, especially when the problem size is large. We thus recommend running Alg. 1 initialized with the training set to reduce the possible time and memory overhead of generating a new random sample set.

Table 1: Optimality (normalized principal utility %) of two setups of gradient-based inference: using the training set (`Training Set`) or a random sample set (`Random Set`) as input $\boldsymbol{X}^{(0)}$ of Alg. 1. Mean and variance over 12 different combinations of (α_p, β_p) are shown. In this table, the problem size is defined by (n, m), where n is the number of actions and m is the number of outcomes.

Problem size (n, m)	`Training Set`	`Random Set`
(5, 16)	95.29 ± 0.20	95.33 ± 0.19
(32, 50)	88.43 ± 0.97	88.46 ± 0.98
(50, 128)	94.28 ± 0.02	94.28 ± 0.02

Table 2: Optimality (normalized principal utilities %) of DeLU networks under different combinations of (α_p, β_p). Mean and variance over 32 problem sizes are shown.

	$\alpha_p = 0.5$	$\alpha_p = 0.7$	$\alpha_p = 0.9$
$\beta_p = 0.0$	88.79±12.77	89.67±12.57	87.20±11.97
$\beta_p = 0.3$	93.94±11.16	93.28±11.48	87.94±13.45
$\beta_p = 0.6$	95.22±7.93	95.08±8.78	91.97±10.32
$\beta_p = 0.9$	97.23±6.62	90.69±18.95	96.43±7.45

E.2 Influence of correlated costs

Across our experiments, we test 12 different combinations of α_p and β_p, where $(\alpha_p, \beta_p) \in \{0.5, 0.7, 0.9\} \times \{0, 0.3, 0.6, 0.9\}$. Recall that a greater β_p value means we put more weights on the independent cost, and a greater α_p value indicates that the correlated cost is more close to the expected value of the action. It is interesting to investigate the influence of these two parameters on the performance of DeLU contract designers.

In Table 2, we show the optimality of DeLU learners (using the LP inference algorithm) under different α_p and β_p values. For each (α_p, β_p) combination, we test 24 different problem sizes: $(m, n) \in \{25, 50, 100\} \times \{2, 4, 8, 16, 32, 64, 128, 256\}$. We give the median and standard deviation of these 24 instances. We observe that β_p exerts influence on the performance of DeLU networks: optimality of the learned contracts generally increases for greater values of β_p, whatever the value of α_p, and we see that DeLU seems better suited to handle problems in which the costs of actions are relatively independent of the expected values of actions. This claim is also supported by the results regarding α_p, where the optimality under $\alpha_p = 0.5$ is typically better than those under other α_p values for most β_p values. It will be interesting in future work to further study this phenomenon, and to see whether suitable modifications can be made to the DeLU architecture to further improve optimality in regimes with higher cost correlation.