Spanning Multi-Asset Payoffs With ReLUs: Numerical Implementation Details*

Sébastien Bossu†, Stéphane Crépey‡, Nisrine Madhar§, Hoang-Dung Nguyen¶

February 27, 2024

Abstract

This is complementary material to the paper Spanning Multi-Asset Payoffs With ReLUs, gathering the detail specification of spanning neural networks, data sampling parameters and additional visualizing results in our numerical experiments.

Keywords: multi-asset options, basket options, one-hidden-layer feedforward ReLU neural network.

Mathematics Subject Classification: 91G20, 62G08, 62M45.

1 Experimental Design

We recall the definition of neural network family by formula (5.1) in the main paper. For two positive integers d, l (referring to the numbers of underlying assets and of spanning basket options), let $\mathcal{NN}_{d,l}$ denote the family of functions that take a vector $\mathbf{x} \in \mathbb{R}^d$ as input and return a value in $\mathbb{R}$ through the sequential mapping

$$\mathbb{R}^d \ni \mathbf{x} \xrightarrow{\tilde{F}} \alpha + \boldsymbol{\mu} \cdot \mathbf{x} + \sum_{i=1}^{l} \nu_i \left(\eta_i(\mathbf{w}^{(i)} \cdot \mathbf{x} - k_i)\right)^+ \in \mathbb{R}, \tag{1.1}$$

*python code available on https://github.com/hoangdungnguyen/Spanning_multi_asset_payoffs.

†sbossu@charlotte.edu. UNC Charlotte Department of Mathematics and Statistics.

‡stephane.crepey@lpsm.paris. Laboratoire de Probabilités, Statistique et Modélisation (LPSM), Université Paris-Cité, France. The research of S. Crépey is supported by the Chairs "Capital Markets Tomorrow: Modeling and Computational Issues", a joint initiative of LPSM/Université Paris-Cité and Crédit Agricole CIB under the aegis of Institut Europlace de Finance, and "Futures of Quantitative Finance", a partnership between LPSM at Université Paris Cité, CERMICS at École des Ponts ParisTech, and BNP Paribas Global Markets.

§madhar@lpsm.paris. Université Paris-Cité, France. The research of N. Madhar is funded by a CIFRE grant from Natixis.

¶hdnguyen@lpsm.paris. Université Paris-Cité, France. The research of H.D. Nguyen is funded by a CIFRE grant from Natixis.

Corresponding author: stephane.crepey@lpsm.paris

where α and $\boldsymbol{\mu}$ denote the quantity of cash and the position vector of underlying assets, $\mathbf{w}^{(i)} = [w_1^{(i)}, \cdots w_d^{(i)}]^T$ are vectors of basket weights , while the sign of each parameter η_i determines whether a basket call or put is used. The sign of each parameter ν_i determines whether a long or short position is taken in basket option i with strike $k_i > 0$, in quantity $|\nu_i \eta_i|$. Let denote the set of trainable parameters by

$$\theta = (\mathbf{W}, \boldsymbol{\mu}, \mathbf{k}, \alpha, \boldsymbol{\nu}, \boldsymbol{\eta}),$$

where $\mathbf{W} = [\mathbf{w}^{(1)}, \cdots, \mathbf{w}^{(l)}]^T \in \mathbb{R}^{l \times d}$ stores the $w_j^{(i)}$ in matrix form, while $\boldsymbol{\mu} \in \mathbb{R}^d$, $\mathbf{k} \in (\mathbb{R}_+^*)^l, \boldsymbol{\nu} \in \mathbb{R}^l$, and $\boldsymbol{\eta} \in \mathbb{R}^l$ store the k_i, ν_i and η_i in vector form, and $\alpha \in \mathbb{R}$. We seek an approximation

$$\hat{F} \in \underset{\tilde{F} \in \mathcal{NN}_{d,l}}{\arg\min}\, \widehat{\mathbb{E}} \left[(F(\mathbf{x}) - \tilde{F}(\mathbf{x}))^2 \right], \tag{1.2}$$

where $\widehat{\mathrm{E}}[.] = \frac{[.]_1 + \cdots + [.]_n}{n}$ denotes the sample mean over n observations drawn from values of $\mathbf{x}$ that may be deterministically or randomly sampled and F is the target payoff function. Since the network is architected with a large number of parameters, the above optimization problem is usually solved using gradient-based approaches. We adopted the Adam method (Kingma & Ba 2015) which is a well-known variant of the stochastic gradient descent method.

For faster optimization, we introduce the BatchNorm operator (Ioffe & Szegedy 2015) at certain points in (1.1) so that the output of the neural network family changes to

$$\mathrm{bn}(\alpha + \boldsymbol{\mu} \cdot \mathbf{x}) + \sum_{i=1}^{l} \nu_i \,\mathrm{bn}\left(\left(\eta_i (\mathbf{w}^{(i)} \cdot \mathrm{bn}(\mathbf{x}) - k_i) \right)^+ \right) +, \tag{1.3}$$

where bn is the scalar or element-wise operator normalizing the parameter by its own mean and standard error computed over training data. BatchNorm takes care of the internal covariate shift of neural network parameters while performing stochastic gradient descent (Ioffe & Szegedy 2015), making the optimization procedure less sensitive to network initialization and the learning rate of the Adam optimizer.

To perform restricted spanning strategies mentioned in Section 5 of the main paper, we reflect the agency of financial markets constraints were hard-coded inside the network architecture instead of relying on regularization techniques that often tolerate small violations. Consequently, (1.1) was modified as follows

$$\mathbb{R}^d \ni \mathbf{x} \xmapsto{\tilde{F}} \alpha + \boldsymbol{\mu} \cdot \mathbf{x} + \sum_{i=1}^{l} \nu_i \left(\eta_i (\psi(\mathbf{w}^{(i)}) \cdot \mathbf{x} - k_i) \right)^+ \in \mathbb{R}, \tag{1.4}$$

where $\psi : \mathbb{R}^d \to \mathbb{R}^d$ is an ad-hoc constraint enforcement function, as described in the following paragraphs.

Single-asset payoff constraint: Here, a d-asset payoff is spanned with l single-asset payoffs, where l may be much larger than d. As a consequence, a particular asset may be over-represented in the spanning portfolio. For greater selection flexibility, we let the optimization procedure decide on the number of payoffs written on each asset and their corresponding strikes by means of the constraint function

$$\psi(\boldsymbol{\zeta}) = \left(\omega(\boldsymbol{\zeta})_1 \mathbf{1}_{\omega(\boldsymbol{\zeta})_1 = \|\omega(\boldsymbol{\zeta})\|_\infty}, \dots, \omega(\boldsymbol{\zeta})_d \mathbf{1}_{\omega(\boldsymbol{\zeta})_d = \|\omega(\boldsymbol{\zeta})\|_\infty}\right),$$

where $\|\cdot\|_\infty$ is the maximum norm of a vector, and $\omega(\boldsymbol{\zeta}) = \left(\frac{\exp(\zeta_1)}{\sum_{j=1}^d \exp(\zeta_j)}, \cdots, \frac{\exp(\zeta_d)}{\sum_{i=j}^d \exp(\zeta_j)}\right)$ denotes the standard softmax of a vector $\boldsymbol{\zeta} = (\zeta_1, \dots, \zeta_d) \in \mathbb{R}^d$. The output of ψ is a vector of all-but-one zero elements, so that the structure of the single-asset payoff is guaranteed: each $\psi(\mathbf{w}^{(i)})$ has a unique nonzero component in (1.4). The normalization ω ensures that each coordinate of $\mathbf{w}^{(i)}$ nevertheless contributes to the predictor and ultimately to the loss function. This procedure allows the underlying asset of a payoff to be flexibly changed throughout the minimization of the loss function, that is, a payoff underwritten on the j-th asset may be changed to be underwritten on an j'-th asset during the next gradient step.

Long-only basket payoff constraint: This constraint is easily implemented with any element-wise positive function ψ in (1.4), for example:

$$\psi(\boldsymbol{\zeta}) = |\boldsymbol{\zeta}| = (|\zeta_1|, \dots, |\zeta_d|)\,.$$

1.1 Data Sampling

We use synthetic data to generate our training and test sets, respectively denoted as $\mathcal{D}^{train}$ and $\mathcal{D}^{test}$ with sizes n^{train}, n^{test}. Each data point is a bundle of asset performances and corresponding payoff value $(\mathbf{x}, F(\mathbf{x})) \in \mathbb{R}^{d+1}$. Table 1 provides the regular grid sampling specifications used to generate the training and test sets in low dimension $d = 2$ to 5. This approach ensures there are no gaps in sampling and facilitates comparisons between option payoffs. Note that some payoffs require a different $\mathbf{x}$-sampling range to curb the number of zero payoff values. Estimates are evaluated on a test set of 50,000 to 200,000 points drawn uniformly in the same range as the training grid.

	BOC/ BOBC	BOP/ BOBP	WOP/ WOBP	WOC/ WOBC	Mex	DC	DP
$d = 5$	$[-2, 2]$		$[0, 2]$		$[-\sqrt{3}, \sqrt{3}]$	$[-2, 2]$	
$d = 2, 3, 4$	$[-2, 2]$						

Table 1: $\mathbf{x}$-sampling range for $d \leqslant 5$.

In higher dimension $d > 5$, grid sampling is no longer feasible due to the curse of dimensionality. Instead, we use random sampling based on a conditional multi-lognormal

distribution

$$x_i = x_i^0 \exp\left(rT - \frac{\sigma_i^2}{2}T + \sigma_i\sqrt{T}\xi_i\right),$$

where $r = 2\%, T = 1$, σ's are uniformly drawn in $[0.1, 0.7]$, and ξ's are correlated standard normals with a random correlation matrix generated by a numerically stable algorithm (Davies & Higham 2000) which can be found in the Python Scipy package. In addition, for each payoff, the initial value x_i^0 is uniformly drawn within a range given in Table 2, and the training and test sample sizes are set to $n^{train} = 100,000, n^{test} = 200,000$. The sampling method is summarized in Algorithm 1.

	BOC, BOBC	BOP, BOBP	WOP, WOBP	WOC, WOBC	Mex	DC	DP
$d = 20$	$[0.5, 0.7]$	$[0.2, 0.5]$	$[2.2, 2.5]$	$[1.4, 1.8]$	$[0, 0.2]$	$[0.8, 1.2]$	$[0.6, 1]$
$d = 50$	$[0.3, 0.6]$	$[0, 0.5]$	$[2, 2.5]$	$[3, 3.5]$	$[0, 0.2]$	$[0.7, 1.2]$	$[0.7, 1]$

Table 2: Uniform distribution ranges for sampling initial asset prices.

name : DataSampling
input : Payoff function F, training and test set sizes n^{train}, n^{test}, a number of underlying assets d
output : Training and test data sets $\mathcal{D}^{train}, \mathcal{D}^{test}$

if $d \leqslant 5$ **then**
  Determine the lower and upper bounds a, b for the assets
  Generate a training data set of n^{train} regular grid points over $[a, b]^d$
  Generate a test data set of n^{test} points i.i.d. uniformly distributed over $[a, b]^d$
else
  `// Generate (independent) training and test data sets of lognormally distributed points`
  Generate asset i initial value x_i^0 and asset volatility σ_i, $i = 1 \dots d$, and a correlation matrix Γ
  for $n \in \{n^{train}, n^{test}\}$ **do**
    Generate n asset prices at the maturity $T = 1$,

$$x_i = x_i^0 \exp\left(rT - \frac{\sigma_i^2}{2}T + \sigma_i\sqrt{T}\xi_i\right), \quad \text{for } i = 1, \dots, d$$

    for $r = 2\%$ and $\mathcal{N}(0, 1)$ random variables ξ_i with correlation matrix Γ
  end
end
For each data point $\mathbf{x}$, compute $F(\mathbf{x})$

Algorithm 1: Sampling method for spanning payoffs.

1.2 Network and Optimizer Configuration

We use the Adam stochastic gradient optimizer of (Kingma & Ba 2015) provided by PyTorch package on Python for our neural networks. We divide the data set into 10

batches and train over 1,000 epochs for a total of 10,000 gradient steps. The learning rate is initially set at 0.01 and decreases by a factor of 0.8 every 300 epochs. A 1% regularization defined as the squared Euclidean norm of all network parameters is implemented through weight decay and contributes to the loss function (see Algorithm 2).

name : NeuralSpanningAlgo
input : $\{(\mathbf{x}_1, F(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, F(\mathbf{x}_n))\}$, a partition B of subsets $batch \subseteq \{1 \ldots n\}$, a number of basket payoffs l, a number of epochs $E \in \mathbb{N}^*$, an initial learning rate $\gamma > 0$, a neural spanning network restriction function ψ. Adam optimizer default setting: $\beta_1 = 0.9, \beta_2 = 0.999$
output : Trained parameters $\hat{\theta}$ of the spanning network

Define the spanning network architecture $F_\theta \in \mathcal{NN}^{\psi}_{d,l}$ depending on the given restrictions ψ
Define the loss function $\text{MSE}(\theta, batch) = \widehat{\text{E}}_{batch}\left[(F(\mathbf{x}) - F_\theta(\mathbf{x}))^2\right]$
Initialize the network parameters $\hat{\theta}$
`// Solve by Adam optimizer (batched version of Algorithm 1 in Kingma & Ba (2015))`
for $epoch = 1, \ldots, E$ **do**
  for $batch \in B$ **do**
    $\hat{\theta} \leftarrow \text{AdamStep}(\gamma, \beta_1, \beta_2, \text{MSE}(\hat{\theta}, batch))$
  end
  Update γ
end

Algorithm 2: Neural spanning network trained by Adam optimizer

2 Visualization of Unconstrained Neural Network Spanning With $d = 2$ Underlying Assets

Figure 1 gives a classic 3D representation from two different camera angles of the best-of call payoff surface $F(x_1, x_2) = (\max(x_1, x_2) - K)^+$ together with values of the trained spanning portfolio payoff $\hat{F}(x_1, x_2)$ shown as red dots. We can see that the training is able to learn the payoff shape, however this type of data visualization is not ideal to get a precise idea of the quality of fit. In Figure 3 we use contour plots to report the target payoff F, trained payoff $\hat{F} \in \mathcal{NN}_{2,l}$ and absolute spanning error $|F(x_1, x_2) - \hat{F}(x_1, x_2)|$ for each target option in our study. We can see that the predicted surface provided by NN fits the surface of the target payoff very well for most options. Remarkably, higher spanning errors are observed around areas where the target payoff is nondifferentiable, with binary options exhibiting the largest errors around discontinuity areas (Contour plots (c), (d), (h) and (i) in Figure 3).

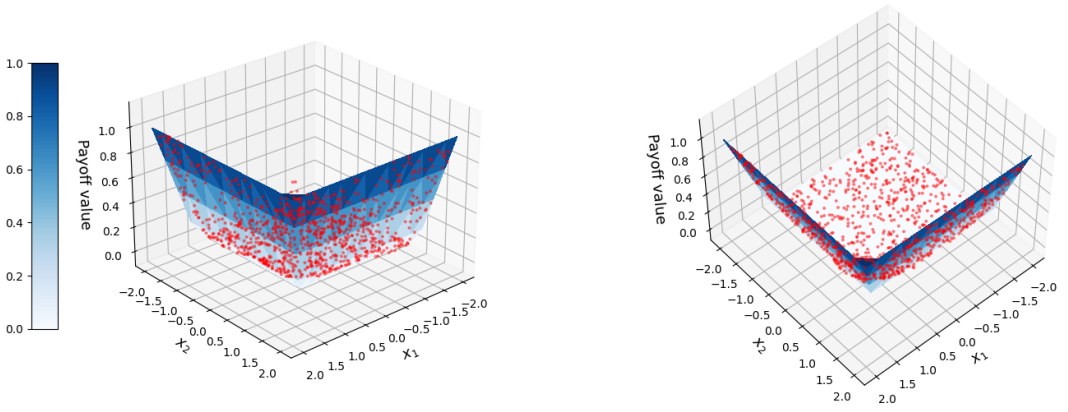
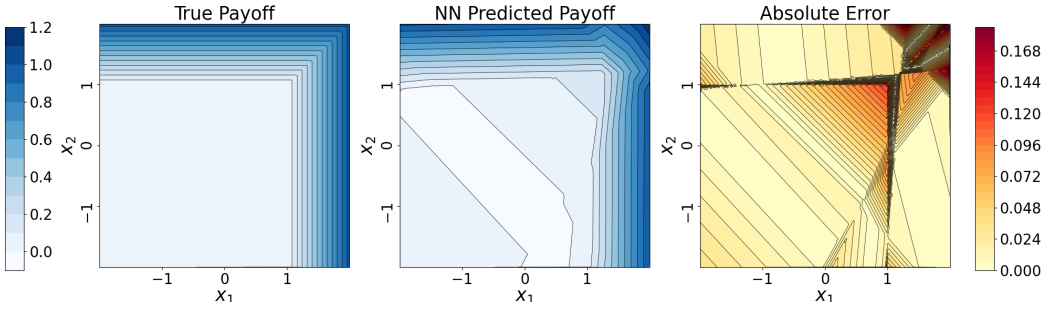
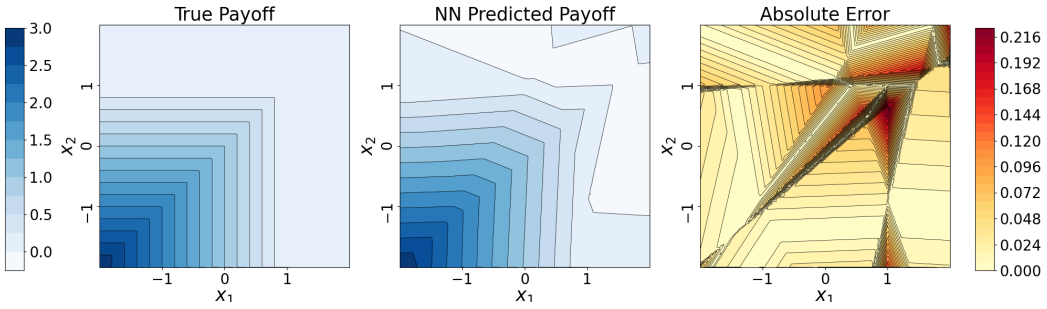


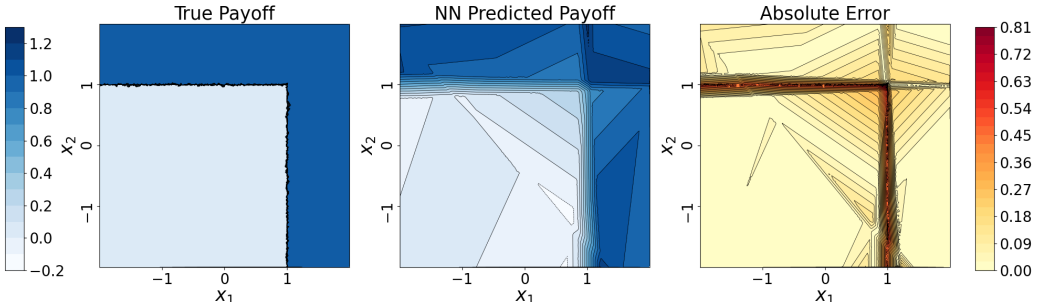
Figure 1: Best of call payoff surface. The red points represent the payoff of the basket spanning portfolio.



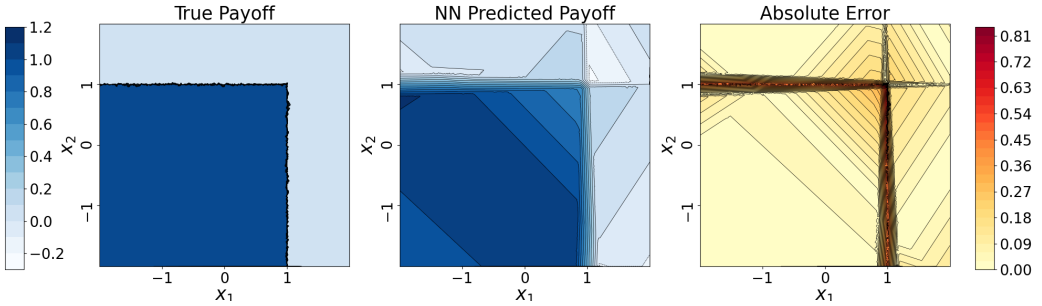
(a) Best of call



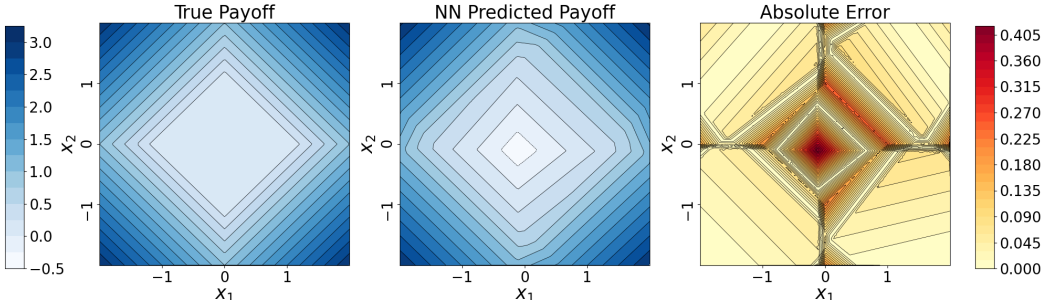
(b) Best of put



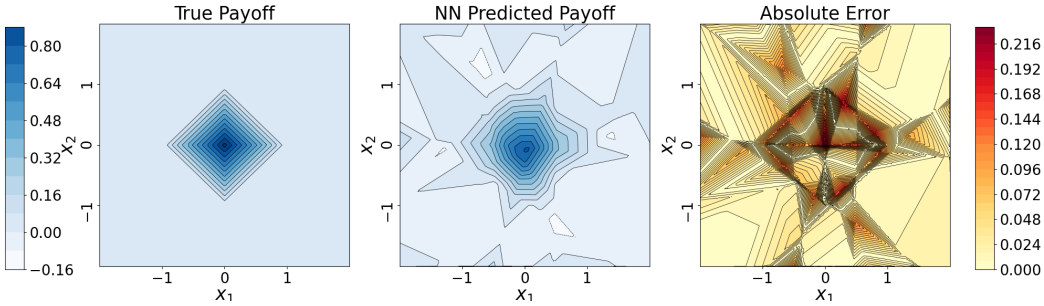
(a) Best of binary call



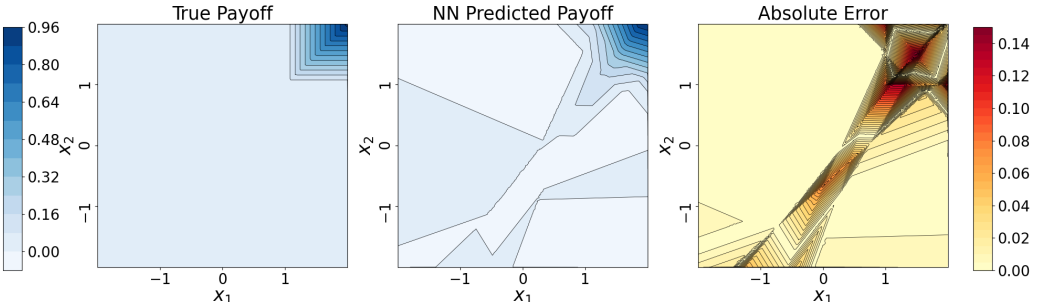
(b) Best of binary put



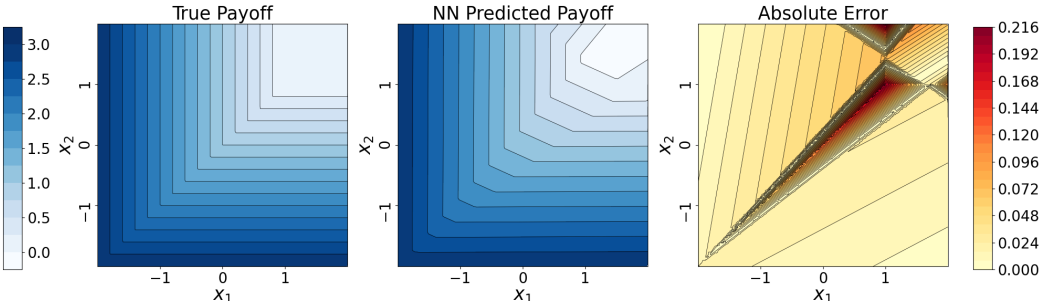
(c) Dispersion call



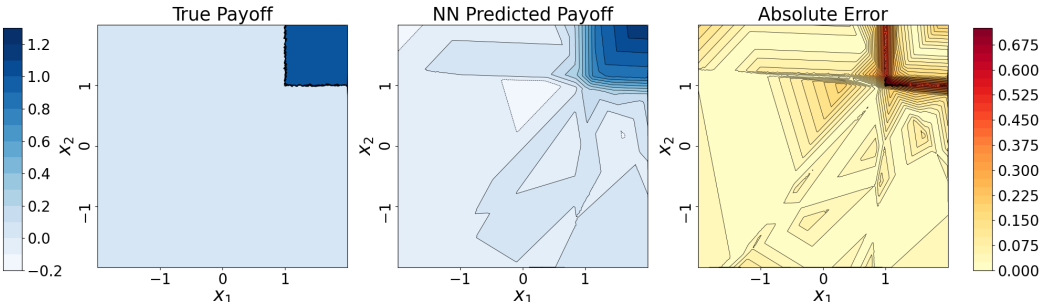
(d) Dispersion put



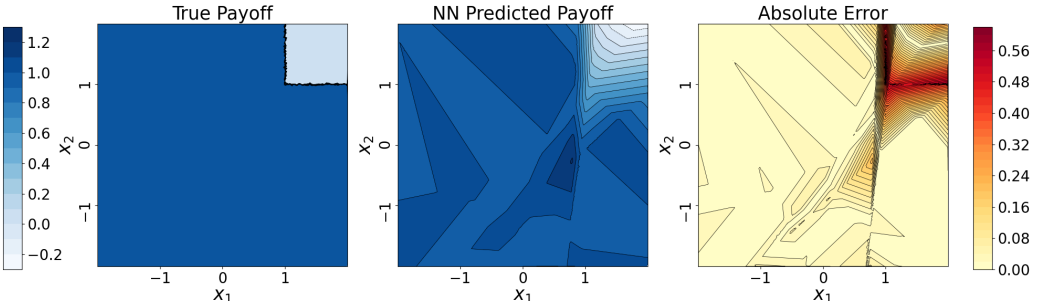
(e) Worst of call



(f) Worst of put



(g) Worst of binary call



(h) Worst of binary put

Figure 3: Contour plots of the target payoff (*left*), of the NN prediction (*center*) and of the spanning error (*right*) for the (a) best-of call, (b) best-of put, (c) best-of binary call, (d) best-of-binary put, (e) dispersion call, (f) dispersion put, (g) the worst-of call, (h) worst-of put payoffs.

References

Davies, P. I. & Higham, N. J. (2000), 'Numerically stable generation of correlation matrices and their factors', *BIT Numerical Mathematics* **40**(4), 640–651.

Ioffe, S. & Szegedy, C. (2015), Batch normalization: Accelerating deep network training by reducing internal covariate shift, *in* 'International conference on machine learning', PMLR, pp. 448–456.

Kingma, D. P. & Ba, J. (2015), 'Adam: A method for stochastic optimization', *the 3rd International Conference on Learning Representations* .