Shuffled Deep Regression

Masahiro Kohjima

NTT Human Informatics Laboratories, NTT Corporation
1-1 Hikarinooka, Yokosuka, Kanagawa 239-0847, Japan
masahiro.kohjima.ev@hco.ntt.co.jp

Abstract

Shuffled regression is the problem of learning regression models from shuffled data that consists of a set of input features and a set of target outputs where the correspondence between the input and output is unknown. This study proposes a new deep learning method for shuffled regression called Shuffled Deep Regression (SDR). We derive the sparse and stochastic variant of the Expectation-Maximization algorithm for SDR that iteratively updates discrete latent variables and the parameters of neural networks. The effectiveness of the proposal is confirmed by benchmark data experiments.

Introduction

Due to the difficulties of comprehensive data collection such as privacy-aware or non-centralized data collection, the data to be analyzed are often given by the form of *shuffled data* where the correspondence between input (feature vector) and output (target value) is unknown (Carpentier and Schlüter 2016; Hsu, Shi, and Sun 2017; Pananjady, Wainwright, and Courtade 2017; Abid and Zou 2018). Since the true target value for each input is unknown, shuffled data can be collected even if the outputs represent sensitive information such as annual income while protecting privacy. To learn regression models from shuffled data, it is impossible to apply standard regression methods using *paired data* whose correspondence between input and output is known.

Shuffled regression (SR) is the problem of estimating a model from shuffled data and arises in domains such as data integration, computer vision, and sensor networks, see e.g., (Pananjady, Wainwright, and Courtade 2017). In the literature of SR, existing studies focus on the learning of linear models (Carpentier and Schlüter 2016; Hsu, Shi, and Sun 2017; Abid and Zou 2018; Fang and Li 2023), and the learning of deep models has not been well investigated. Considering the known benefits of deep learning (Goodfellow, Bengio, and Courville 2016), the use of deep learning must contribute to improving the performance of solving SR.

This study proposes a new deep learning-based method for shuffled regression called Shuffled Deep Regression (SDR). We derive a sparse stochastic expectation-maximization (SSEM) algorithm for parameter estimation that iteratively updates latent variables and the parameters of neural networks, based on the EM algorithm (Dempster, Laird, and Rubin 1977). SSEM adopts sparse EM (Neal and Hinton 1998) and online EM (Cappé and Moulines 2009)-like approaches to naturally handle the huge spaces of discrete latent variables and supports combination with stochastic gradient algorithms (such as Adam) to update neural network parameters. We also show that SSEM is related to the iterative reweighted least squares (IRLS) (Rubin 2004; Bishop 2006). Experiments conducted on benchmark datasets confirm the effectiveness of the proposals.

The contributions of this paper are summarized below:

- We propose a new deep learning-based method for shuffled regression called shuffled deep regression (SDR).
- We develop the sparse and stochastic variant of EM algorithm (SSEM) that can naturally handle the huge space of discrete latent variables and can support the use of stochastic gradient-based algorithms.
- We confirm the effectiveness of the proposed method by numerical experiments on benchmark data.

Related Work

Shuffled regression (SR) (Pananjady, Wainwright, and Courtade 2017; Abid and Zou 2018) which is also called "regression without correspondence" (Hsu, Shi, and Sun 2017), "uncoupled regression" (Xu et al. 2019), and "unlabeled sensing" (Unnikrishnan, Haghighatshoar, and Vetterli 2018), arises in e.g., the pose and correspondence estimation (David et al. 2004), flow cytometry for measuring chemical characteristics of cells (Abid and Zou 2018), and linkage of health records (Shi, Li, and Cai 2021). So SR has been actively studied both theoretically and for application.

Theoretical studies have clarified various difficulties of SR. For instance, recovering the correspondence between input and output is NP-hard in general (Pananjady, Wainwright, and Courtade 2017). (Unnikrishnan, Haghighatshoar, and Vetterli 2018) also explored the conditions for obtaining unique solutions. Developing parameter estimation algorithms is a notable topic as well; (Abid and Zou 2018) developed an expectation-maximization (EM) based algorithm similar to our study, while (Slawski, Rahmani, and Li 2020) elucidated the connection to robust subspace recovery problem. However, these existing studies mainly

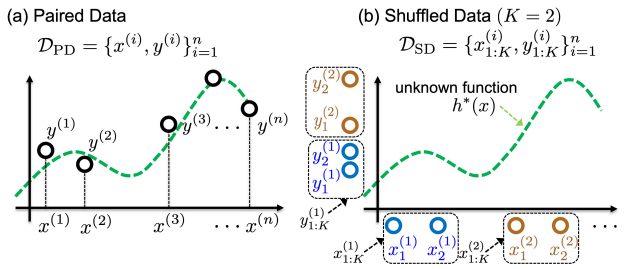


Figure 1: (a) Paired Data and (b) Shuffled Data. Shuffled regression (SR) is the problem of estimating a (true) unknown function from shuffled data.

	parameter	model form
Linear	W, b	$h_\theta(x) = Wx + b$
MLP	$\{W_\ell, b_\ell\}_{\ell=1}^2$	$h_\theta(x)$=$W_2\mathrm{act}(W_1x$+$b_1)$+b_2

Table 1: Examples of regression model h_θ.

	η	$p_0(y)$	$T(y)$	$A(\eta)$
Normal	c/σ	$\frac{\exp(-\frac{y^2}{2\sigma^2})}{\sqrt{2\pi\sigma}}$	y/σ	$\eta^2/2$
Bernoulli	$\log(\frac{r}{1-r})$	1	y	$\log(1+e^\eta)$
Poisson	$\log(\lambda)$	$1/y!$	y	$\exp(\eta)$

Table 2: Examples of exponential family distribution f.

focus on simple models with some specific structure such as the monotone model (Carpentier and Schlüter 2016) or the linear model (Pananjady, Wainwright, and Courtade 2017; Abid and Zou 2018), and their validity is limited to cases that suit the simple models. Our motivation was to apply deep models and develop an applicable estimation algorithm.

Research directed towards practical settings have considered problems where richer data (to the extent practical) are available (Xu et al. 2019; Kohjima et al. 2022; Yamane et al. 2023). For example, (Xu et al. 2019; Kohjima et al. 2022) use pairwise ranking data, while (Yamane et al. 2023) assume the availability of some intermediate variables that weakly connect inputs and outputs. In this context, the use of more complex models such as Gaussian process model (Kohjima et al. 2022) and deep learning-based method (Yamane et al. 2023) has been investigated. However, these methods cannot be applied if ranking data or intermediate variables are not available. Our study considers the learning of deep models without using such additional data and variables.

Preliminary: Regression with Paired Data

Let $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}$ be an input feature space and an output space, respectively. We call the set of n pairs of input and output, $\mathcal{D}_{\mathrm{PD}} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, paired data, where $x^{(i)} \in \mathcal{X}$ is an input feature and $y^{(i)} \in \mathcal{Y}$ is an target output. See Fig. 1a.

In the (standard) regression problem, given (i) regression model $h_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by θ, (ii) output probability distribution f, and (iii) paired data $\mathcal{D}_{\mathrm{PD}}$, the model parameter θ is estimated by maximizing the log-likelihood.

Possible choices for the regression model h_θ include the linear models and the deep neural network models such as the (one-hidden layer) multi-layer perceptron (MLP) shown in Table 1. Although the main scope of this study lies in the variety of deep models including MLP where the parameter is updated by stochastic algorithms such as SGD or Adam, our algorithm can handle any type of regression model.

For the output probability distribution f, the most standard choice for f is the normal distribution defined as

$$\mathcal{N}(y|c) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{(y-c)^2}{2\sigma^2}\right\}.$$

It is well-known that the use of the normal distribution makes the log-likelihood function correspond to the squared-error loss. Since Gaussian may not be appropriate to handle binary/non-negative/integer target outputs, the following exponential family distribution is the more general choice since it can express various types of distributions:

$$f(y|\eta) = p_0(y) \exp\{\eta \cdot T(y) - A(\eta)\}, \qquad (1)$$

where η is the *natural parameter*, $T(y)$ is *sufficient statistics* and $A(\eta)$ is the log-normalizer. The exponential family includes Normal, Bernoulli (Ber), and Poisson (Poi):

$$\mathrm{Ber}(y|r) = r^y(1-r)^{1-y}, \ \mathrm{Poi}(y|\lambda) = \lambda^y \exp(-\lambda)/y!.$$

By assigning specific values to $T(y)$ and $A(\eta)$, the above distributions are represented by Eq. (1) (See Table 2). Our notation of f makes the output distribution transparent to which distribution is adopted.

Proposed Method: Shuffled Deep Regression

This section describes the method we propose for estimating deep regression models from shuffled data.

Shuffled Data

Shuffled data are data where the correspondence between input feature and target output is unknown. Let $x_{1:K}$ and $y_{1:K}$ be a set of K input features $x_{1:K} = \{x_1, x_2, \cdots, x_K\}$, and a set of K target outputs $y_{1:K} = \{y_1, y_2, \cdots, y_K\}$, respectively ($x_k \in \mathcal{X}$, $y_k \in \mathcal{Y}$). Shuffled data $\mathcal{D}_{\mathrm{SD}}$ are defined as a pair of input-set and output-set, $\mathcal{D}_{\mathrm{SD}} = \{(x_{1:K}^{(i)}, y_{1:K}^{(i)})\}_{i=1}^n$ [1]. See Fig. 1b. Unlike paired data, one-to-one correspondence between input and output is lost in shuffled data, so the output corresponding to feature $x_k^{(i)}$ is one of $\{y_1^{(i)}, ..., y_K^{(i)}\}$, but it is not known which one. Since shuffled data are reduced to paired data when K=1, it can be regarded as a more general representation of paired data. We denote the matrix representation of $x_{1:K}$ by the bold capital letters, $\mathbf{X}_{1:K} \in \mathbb{R}^{K \times d}$. Table 3 shows our notation.

The shuffled regression (SR) problem tackled by this study is to estimate the parameters of the regression model given (i) regression model $h_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by θ, (ii) output probability distribution f, and (iii) shuffled data $\mathcal{D}_{\mathrm{SD}}$. We emphasize that paired data $\mathcal{D}_{\mathrm{PD}}$ are not given.

[1] This setup could easily be extended to an adaptive K setup (K is different for each sample i), but that is is omitted for simplicity.

Symbol	Description
n	size of shuffled data
d	size of dimensions of input features
K	size of a set of input features/target outputs
$\mathcal{D}_{\mathrm{SD}}$	shuffled data $\mathcal{D}_{\mathrm{SD}} = \{(x_{1:K}^{(i)}, y_{1:K}^{(i)})\}_{i=1}^{n}$
$x_{1:K}^{(i)}$	i-th set of input features $\{x_1^{(i)}, \cdots, x_K^{(i)}\}$
$y_{1:K}^{(i)}$	i-th set of target outputs $\{y_1^{(i)}, \cdots, y_K^{(i)}\}$
$\mathbf{X}_{1:K}^{(i)}$	$(K \times d)$ matrix representation of $x_{1:K}^{(i)}$
θ	parameter of regression model (See Table 1)
h_θ	regression model (See Table 1)
f	output probability distribution (See Table 2)
g	link function (See Table 4)
Π	set of all $K \times K$ permutation matrices
$\mathbf{P}^{(i)}$	(latent) permutation matrix for i-th data
$\eta_{1:K}^{(i)}$	set of parameter of f, $\eta_{1:K}^{(i)} = (\eta_1^{(i)}, \cdots, \eta_K^{(i)})$
$\eta_k^{(i)}$	output of link func. $\eta_k^{(i)} = g(h_\theta([\mathbf{P}^{(i)}\mathbf{X}_{1:K}^{(i)}]_k))$
$\tilde{\Pi}^{(i)}$	set of M permutation matrices $\{\tilde{\mathbf{P}}^{(i,m)}\}_{m=1}^{M}$
$\gamma^{(i,m)}$	responsibility of m-th perm. matrix (Eq. (11))
$\tilde{x}_k^{(i,m)}$	shuffled input $\tilde{x}_k^{(i,m)} = [\tilde{\mathbf{P}}^{(i,m)}\mathbf{X}_{1:K}^{(i)}]_k$
$\tilde{\eta}_k^{(i,m)}$	output of link func. $\tilde{\eta}_k^{(i,m)} = g(h_\theta(\tilde{x}_k^{(i,m)}))$
$\check{z}^{(t)}$	latent variable sampled from categorical dist.
$\check{x}_k^{(t)}$	shuffled input $\check{x}_k^{(t)} = [\tilde{\mathbf{P}}^{(t,\check{z}^{(t)})}\mathbf{X}_{1:K}^{(t)}]_k$
$\check{\eta}_k^{(t)}$	output of link func. $\check{\eta}_k^{(t)} = g(h_\theta(\check{x}_k^{(t)}))$

Table 3: Notation summary for the paper

Note that in early works on SR such as (Hsu, Shi, and Sun 2017; Pananjady, Wainwright, and Courtade 2017; Abid and Zou 2018) consider the setting where data size n=1. In practical data collection scenarios such as survey data collection, data are often collected multiple times from different questionnaire respondents; even if the correspondence between the input and output is obscured to protect privacy, we can know which data collection event the data was gathered in and the data are represented by shuffled data with n>1.

Generative Process

The generative process of shuffled data $\mathcal{D}_{\mathrm{SD}}$ is described by the following three steps. For each $i = 1, \cdots, n$, (Step-1) a set of input features $x_{1:K}^{(i)}$ is determined subject to some probability distribution $p(x_{1:K})$. (Step-2) a latent (unknown) $K \times K$ permutation matrix $\mathbf{P}^{(i)}$ used for shuffling the order of inputs (row of $\mathbf{X}_{1:K}$) is generated by uniform distribution $u(\mathbf{P})$ [2]. Then, in (Step-3) , a set of outputs $y_{1:K}^{(i)}$ is determined subject to

$$p(y_{1:K}|\mathbf{P}, x_{1:K}, \theta) = \prod_{k=1}^{K} f\big(y_k|\eta_k\big), \qquad \eta_k = g(h_\theta([\mathbf{P}\mathbf{X}_{1:K}]_k)), \tag{2}$$

[2]Permutation matrix $\mathbf{P}$ is the doubly stochastic matrix whose elements are 0 or 1, i.e., $\sum_{k=1}^{K}[\mathbf{P}]_{k\ell} = 1\ (\forall \ell)$, $\sum_{\ell=1}^{K}[\mathbf{P}]_{k\ell} = 1\ (\forall k)$, $[\mathbf{P}]_{k\ell} \in \{0, 1\}\ (\forall k, \ell)$. Since the size of the set of all $K \times K$ permutation matrices is $K!$, $u(\mathbf{P}) = 1/K!$ for all $\mathbf{P} \in \Pi$.

typical use	dist. f	link func. $g(\hat{y})$
Linear Regression	Normal	$\hat{y}$
Logistic Regression	Bernoulli	$1/\{1+\exp(-\hat{y})\}$
Poisson Regression	Poisson	$\exp(\hat{y})$

Table 4: Typical use of distributions and (inverse of) link functions based on the generalized linear model.

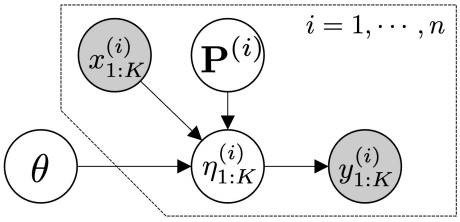


Figure 2: Graphical model representation of shuffled data. Shaded nodes represent observed variables.

where $[\mathbf{P}\mathbf{X}_{1:K}]_k$ denotes the k-th row vector of $\mathbf{P}\mathbf{X}_{1:K}$, and g is the link function that connects the output of the regression model and the (natural) parameter of the output distribution f, similar to the generalized linear model (Nelder and Wedderburn 1972; Fang and Li 2023) (See Table 4 for typical choise). The use of link function also can be found in deep models (Ranganath et al. 2015).

Summarizing the above, the joint probability of the shuffled data $\mathcal{D}_{\mathrm{SD}}$ and a set of permutation matrix $\mathcal{P}$=$\{\mathbf{P}^{(i)}\}_{i=1}^{n}$ given parameters θ is given by

$$p(\mathcal{D}_{\mathrm{SD}}, \mathcal{P}|\theta) = \prod_{i=1}^{n} p(y_{1:K}^{(i)}|\mathbf{P}^{(i)}, x_{1:K}^{(i)}, \theta) p(x_{1:K}^{(i)}) u(\mathbf{P}^{(i)}).$$

Figure 2 shows the graphical model representation. Note that we denote the set of natural parameters as $\eta_{1:K}^{(i)} = (\eta_1^{(i)}, \cdots, \eta_K^{(i)})$ and $\eta_k^{(i)} = g(h_\theta([\mathbf{P}^{(i)}\mathbf{X}_{1:K}^{(i)}]_k))$. Marginalizing out $\mathcal{P}$, we get the following likelihood of the shuffled data $\mathcal{D}_{\mathrm{SD}}$:

$$p(\mathcal{D}_{\mathrm{SD}}|\theta) = \sum\nolimits_{\mathbf{P}^{(1)} \in \Pi} \cdots \sum\nolimits_{\mathbf{P}^{(n)} \in \Pi} p(\mathcal{D}_{\mathrm{SD}}, \mathcal{P}|\theta), \tag{3}$$

where Π is the set of all $K \times K$ permutation matrices. The size of set $|\Pi|$ is $K!$, and later we show how to deal with the huge size of set Π. Parameter θ is estimated by maximizing the following log-likelihood function.

$$\hat{\theta}_{\mathrm{MLE}} = \arg\max_{\theta} \log p(\mathcal{D}_{\mathrm{SD}}|\theta). \tag{4}$$

Note that the order of applying regression model h_θ (with link function g) and permutation by $\mathbf{P}$ can be exchangeable; the identical likelihood function is derived regardless of the order since $g(h_\theta([\mathbf{P}\mathbf{X}_{1:K}]_k)) = [\mathbf{P}\big(g(h_\theta(x_1)), \cdots, g(h_\theta(x_K))\big)^\top]_k$ holds.

Lower Bound of Objective Function

For maximizing the objective function of Eq. (4), we use the expectation maximization (EM) algorithm (Dempster,

Laird, and Rubin 1977). EM indirectly maximizes the objective by using the following lower-bound function, $\mathcal{L}$:

$$\mathcal{L}(q,\theta) = \mathbb{E}_{q(\mathcal{P})}\big[\log p(\mathcal{D}_{\mathrm{SD}},\mathcal{P}|\theta) - \log q(\mathcal{P})\big], \quad (5)$$

where $\mathbb{E}_{q(\mathcal{P})}$ is the expectation w.r.t. $q(\mathcal{P})$. By Jensen's inequality, $\mathcal{L}$ is shown to be a lower bound of the log-likelihood:

$$\log p(\mathcal{D}_{\mathrm{SD}}|\theta) = \log \mathbb{E}_{q(\mathcal{P})}\Big[\frac{p(\mathcal{D}_{\mathrm{SD}},\mathcal{P}|\theta)}{q(\mathcal{P})}\Big] \geq \mathbb{E}_{q(\mathcal{P})}\Big[\log \frac{p(\mathcal{D}_{\mathrm{SD}},\mathcal{P}|\theta)}{q(\mathcal{P})}\Big] = \mathcal{L}.$$

Lower bound $\mathcal{L}$ can be expanded to yield the following form:

$$\mathcal{L}(q,\theta) = \log p(\mathcal{D}_{\mathrm{SD}}|\theta) - \mathrm{KL}\big(q(\mathcal{P})||p(\mathcal{P}|\mathcal{D}_{\mathrm{SD}},\theta)\big), \quad (6)$$

where $\mathrm{KL}(Q||P)$ is the KL-divergence between Q and P.

Difficulties Posed by EM Algorithm in Handling Shuffled Data

EM algorithm is the following two-step procedure that maximizes the log-likelihood $\log p(\mathcal{D}_{\mathrm{SD}}|\theta)$: **(E-step)** Maximize $\mathcal{L}$ w.r.t. $q(\mathcal{P})$ and **(M-step)** Maximize $\mathcal{L}$ w.r.t. θ.

For E-step, from Eq. (6), $\mathcal{L}$ is maximized when q is equivalent to the posterior probability of $\mathcal{P}$:

$$(\text{Exact E-Step}) : \hat{q}_{\mathrm{Ex}}(\mathcal{P}) = p(\mathcal{P}|\mathcal{D}_{\mathrm{SD}},\theta), \quad (7)$$

where the above posterior is decomposed as $p(\mathcal{P}|\mathcal{D}_{\mathrm{SD}},\theta) = \prod_{i=1}^{n} p(\mathbf{P}^{(i)}|\mathcal{D}_{\mathrm{SD}},\theta)$ and

$$p(\mathbf{P}^{(i)}|\mathcal{D}_{\mathrm{SD}},\theta) = \frac{\prod_{k=1}^{K} f\big(y_k^{(i)}|\eta_k^{(i)}\big)}{\sum_{\mathbf{P}\in\Pi}\prod_{k=1}^{K} f\big(y_k^{(i)}|g(h_\theta([\mathbf{P}\mathbf{X}_{1:K}^{(i)}]_k))\big)}. \quad (8)$$

However, the above exact E-Step is infeasible for general K since $\hat{q}_{\mathrm{Ex}}$ is the probability distribution on a $|\Pi|(=K!)$-dimensional probabilistic vector. To avoid this difficulty, (Abid and Zou 2018) adopts the following Winner-Takes-All (WTA) variant of E-step (for linear model and normal distribution):

$$(\text{WTA E-Step}) : \hat{q}_{\mathrm{WTA}}(\mathcal{P}) = \mathbb{I}(\mathcal{P} = \hat{\mathcal{P}}), \quad \hat{\mathcal{P}} = \arg\max_{\mathcal{P}} p(\mathcal{P}|\mathcal{D}_{\mathrm{SD}},\theta). \quad (9)$$

This WTA variant has some computational advantage in general but local convergence (monotonic decrease in objective function) is not assured (Neal and Hinton 1998). Accordingly, we adopt the approach of the sparse EM algorithm (Neal and Hinton 1998). Sparse-EM falls between the exact-EM and WTA-EM, and well balances computation cost against convergence performance.

Sparse EM Algorithm for Shuffled Regression

Let $\tilde{\Pi}^{(i)} = \{\tilde{\mathbf{P}}^{(i,1)}\cdots,\tilde{\mathbf{P}}^{(i,m)},\cdots,\tilde{\mathbf{P}}^{(i,M)}\}$ be a set of M ($\ll K!$) candidates of the permutation matrix. Details on how to construct $\tilde{\Pi}^{(i)}$ are given at the end of the section. Sparse-EM (Neal and Hinton 1998) considers the following E-step with the restriction that the probability of a permutation matrix being other than the matrices contained in $\tilde{\Pi}^{(i)}$ is zero:

$$(\text{Sparse E-Step}) : \hat{q}_{\mathrm{Sp}}(\mathbf{P}^{(i)}) = \begin{cases} \gamma^{(i,m)} & (\text{if } \mathbf{P}^{(i)} = \tilde{\mathbf{P}}^{(i,m)}) \\ 0 & (\text{otherwise}) \end{cases} \quad (10)$$

where $\gamma^{(i,m)}$, defined below, can be seen as the *responsibility* of the m-th permutation matrix for i-th data:

$$\gamma^{(i,m)} = \frac{\prod_{k=1}^{K} f\big(y_k^{(i)}|\tilde{\eta}_k^{(i,m)}\big)}{\sum_{m'=1}^{M}\prod_{k=1}^{K} f\big(y_k^{(i)}|\tilde{\eta}_k^{(i,m')}\big)}, \quad \tilde{\eta}_k^{(i,m)} = g(h_\theta(\tilde{x}_k^{(i,m)})), \quad \tilde{x}_k^{(i,m)} = [\tilde{\mathbf{P}}^{(i,m)}\mathbf{X}_{1:K}^{(i)}]_k. \quad (11)$$

Thus this E-step avoids to need to compute the summation over Π.

For M-step, from Eq. (5), parameter θ that maximizes $\mathcal{L}$ is given by

$$(\text{Exact M-Step}) : \hat{\theta} = \arg\max_{\theta} \mathbb{E}_{q(\mathcal{P})}\big[\log p(\mathcal{D}_{\mathrm{SD}},\mathcal{P}|\theta)\big].$$

When the sparse E-step is utilized, q takes the form of sparse distribution $\hat{q}_{\mathrm{Sp}}$ with many zeros (Eq. (10)). Thus the sparse variant of M-step given below also can avoid the summation over Π:

$$(\text{Sparse M-Step}) : \hat{\theta} = \arg\max_{\theta} \mathbb{E}_{\hat{q}_{\mathrm{Sp}}(\mathcal{P})}\big[\log p(\mathcal{D}_{\mathrm{SD}},\mathcal{P}|\theta)\big],$$

where the objective of sparse M-step can be expanded as

$$\mathbb{E}_{\hat{q}_{\mathrm{Sp}}}\big[\log p(\mathcal{D}_{\mathrm{SD}},\mathcal{P}|\theta)\big] = \sum_{i=1}^{n}\sum_{k=1}^{K}\sum_{m=1}^{M}\gamma^{(i,m)}\log f\big(y_k^{(i)}|\tilde{\eta}_k^{(i,m)}\big) + \mathrm{Const.} \quad (12)$$

This objective function can be interpreted as the objective of the weighted regression problem using paired data with input $\tilde{x}_k^{(i,m)}$, output $y_k^{(i)}$, and sample weight $\gamma^{(i,m)}$. Therefore, if distribution f is Gaussian and linear models are adopted, the sparse EM algorithm that iteratively apply the sparse E-step and M-step can be seen as the iterative reweighted least squares (IRLS) (Rubin 2004; Bishop 2006).

The use of numerical optimization techniques is allowed for M-step since if the objective value (Eq. (12)) is improved (even if the maximum point is not strictly obtained), monotone increase of the likelihood function can be assured (while $\tilde{\Pi}^{(i)}$ is fixed), similar to the generalized EM algorithm (Dempster, Laird, and Rubin 1977; Neal and Hinton 1998). At infrequent intervals, $\tilde{\Pi}^{(i)}$ should be updated due to the change in the other parameters.

Sparse Stochastic EM for Shuffled Deep Regression

Here we develop the proposed sparse and stochastic variant of EM algorithm (SSEM). Combined with the stochastic gradient algorithm such as SGD or Adam (Kingma and Ba 2014) and the sparse EM algorithm presented in previous subsection, we can build an algorithm suitable for estimating deep learning models [3].

[3] SGD or Adam itself cannot be directly applied due to the existence of the (latent) permutation matrix.

Algorithm 1: SSEM algorithm for Shuffled Deep Regression

Input: model h_θ, output distribution f, shuffled data $\mathcal{D}_{\mathrm{SD}}$, hyperparameters (e.g., # of steps T_{max}, interval C).
Output: model parameter θ

1: Randomly initialize $\{\tilde{\Pi}^{(i)}\}_i$ and θ.
2: (Optional) Initialize $\{\tilde{\Pi}^{(i)}\}_i$ by solving SLR.
3: **for** $t = 1$ to T_{max} **do**
4: Randomly sample $(x^{(t)}_{1:K}, y^{(t)}_{1:K}) \sim \mathcal{D}_{\mathrm{SD}}$.
5: *//E-Step*
6: Compute $\gamma^{(t,m)}$ following Eq. (11).
7: Randomly sample $\check{z}^{(t)} \sim \mathrm{Cat}(z^{(t)}|\{\gamma^{(t,m)}\}_m)$.
8: *//M-Step*
9: Compute $\check{\eta}^{(t)}_k$ following following Eq. (13)
10: Compute loss $\mathcal{F}$ (Eq. (14)) and update θ by Adam.
11: *//Infrequent updates*
12: Every C steps/epochs, update $\{\tilde{\Pi}^{(i)}\}_i$.
13: **end for**

The proposed SSEM updates parameters using a part of stochastically selected (mini-batch) data and permutation matrix. The pseudo code of the SSEM algorithm is written in Alg. 1. At each (algorithmic) time step t, the algorithm randomly samples $(x^{(t)}_{1:K}, y^{(t)}_{1:K})$ from given shuffled data $\mathcal{D}_{\mathrm{SD}}$. As the E-step (line 5-7), the parameters of $\hat{q}_{\mathrm{Sp}}$, $\gamma^{(t,m)}$, are computed and randomly samples $\check{z}^{(t)}$ following categorical distribution (Cat) with parameter $\{\gamma^{(t,m)}\}_m$. Next, as M-step (line 8-10), using the $\check{z}^{(t)}$-th permutation matrix, we compute

$$\check{\eta}^{(t)}_k = g(h_\theta(\check{x}^{(t)}_k)), \quad \check{x}^{(t)}_k = [\tilde{\mathbf{P}}^{(t,\check{z}^{(t)})}\mathbf{X}^{(t)}_{1:K}]_k. \tag{13}$$

Then, compute loss $\mathcal{F}$, see below, which is the stochastic approximation of the M-step objective (Eq. (12)) ignoring constant terms and inverting signs:

$$\mathcal{F}(\theta; \{x^{(t)}_{1:K}, y^{(t)}_{1:K}\}) = \sum\nolimits_{k=1}^{K} -\log f(y^{(t)}_k|\check{\eta}^{(t)}_k) \tag{14}$$

and update parameters by any of several optimizers such as Adam (Kingma and Ba 2014). Note that If (mini-)batch data $\{(x^{(t)}_{1:K}, y^{(t)}_{1:K})\}_{t=1}^{S}$ are available, we can update the parameters by minimizing the objective function with (mini-)batch data $\frac{1}{S}\sum_{t=1}^{S}\mathcal{F}(\theta; \{x^{(t)}_{1:K}, y^{(t)}_{1:K}\})$.

Updating Candidates of Permutation Matrix

At the end of this section, we detail the construction of $\tilde{\Pi}^{(i)}$. The key is the following Proposition, which states that *best* permutation matrix can be found by a sort operation.

Proposition 1 *The permutation matrix $\hat{\mathbf{P}}^{(i)}$ that maximizes posterior probability (Eq.* (8)*) is the one that rearranges the elements of $\{x^{(i)}_k\}_{k=1}^K$ such that the order of magnitude of values $\{g(h_\theta(x^{(i)}_k))\}_{k=1}^K$ matches the order of $\{T(y^{(i)}_k)\}_{k=1}^K$.*

Dataset	MPG	Abalone	Boston	Conc.
data size n_B	398	3341	506	1030
dimension d	7	10	13	8

Table 5: Dataset statistics

Proof *Without loss of generality, we assume that $y_{1:K}$ are sorted in ascending order of $\{T(y_k)\}_{k=1}^K$, i.e., $T(y_k) \le T(y_\ell)$ if $k < \ell$. Let us denote $g(h_\theta(x_k))$ as g_k and define $\boldsymbol{g} = (g_1, \cdots, g_K)$ for simplicity. The proof is done by contradiction. Let us assume the posterior probability (or equivalently, logarithm of its numerator $\sum_k \log f(y_k|g'_k)$) is maximized at $\boldsymbol{g}' = (g'_1, \cdots, g'_K)$ where the entries are not in ascending order, i.e., there exists k, ℓ such that $k < \ell$ and $g'_k \ge g'_\ell$. But this explicitly implies $\log f(y_k|g'_k) + \log f(y_\ell|g'_\ell) \le \log f(y_k|g'_\ell) + \log f(y_\ell|g'_k)$ since*

$$\begin{aligned}
&\log f(y_k|g'_k) + \log f(y_\ell|g'_\ell) - \{\log f(y_k|g'_\ell) + \log f(y_\ell|g'_k)\} \\
&= g'_k \cdot T(y_k) + g'_\ell \cdot T(y_\ell) - g'_\ell \cdot T(y_k) - g'_k \cdot T(y_\ell) \\
&= -(g'_k - g'_\ell) \cdot \{T(y_\ell) - T(y_k)\} \le 0.
\end{aligned}$$

Therefore, we get

$$\begin{aligned}
&\sum\nolimits_{k'=1}^{K} \log f(y_{k'}|g'_{k'}) \\
&= \log f(y_k|g'_k) + \log f(y_\ell|g'_\ell) + \sum\nolimits_{k' \ne k,\ell}^{K} \log f(y_{k'}|g'_{k'}) \\
&\le \log f(y_k|g'_\ell) + \log f(y_\ell|g'_k) + \sum\nolimits_{k' \ne k,\ell}^{K} \log f(y_{k'}|g'_{k'}).
\end{aligned}$$

This result contradicts the assumption that $\boldsymbol{g}'$ is maximal. □

Note that this proposition is an extension of the result in (Abid and Zou 2018) where a Gaussian distribution is used as f to (any) exponential family distribution.

In constructing $\tilde{\Pi}^{(i)}$, it is promising to use the neighborhood of $\hat{\mathbf{P}}^{(i)}$ obtained by the sort stating in Prop. 1. After rearranging the order of the K elements by $\hat{\mathbf{P}}^{(i)}$, and by further re-ordering the ℓ-th and $(\ell+1)$-th smallest adjacent elements, we can obtain the neighborhood of the permutation matrix $\hat{\mathbf{P}}^{(i)}$. Repeating this for all $\ell = 1, \cdots, K-1$, we get $K-1$ neighbors; adding $\hat{\mathbf{P}}^{(i)}$ to it, we can construct K candidate permutation matrices that can be used as $\tilde{\Pi}^{(i)}$. In later experiments, we use this way of construction when updating $\tilde{\Pi}^{(i)}$ (line 12 in Alg.1). Since this construction depends on the current parameter θ, this update should be done at some (infrequent) intervals.

The above construction can also be used to get a "good" initial setting to learn deep models (line 2 in Alg.1). For example, after estimating the parameters by solving shuffled regression using an easy-to-learn model such as a linear model, the estimated linear model can be used to get an initial setting of $\tilde{\Pi}^{(i)}$ in the same way as above.

Experiment

This section confirms the effectiveness of the proposed SDR against linear model-based methods for shuffled regression.

Dataset Source	K	(Oracle) Supervised Regression LR	DR	Shuffled Regression SLR	SDR (ours)
MPG-1D	2	23.97 ± 2.51	17.57 ± 2.40	24.79 ± 2.76	**17.34 ± 2.07**
	4	23.98 ± 2.48	17.67 ± 2.44	23.89 ± 2.64	**17.96 ± 1.94**
	8	23.98 ± 2.48	17.92 ± 2.78	24.36 ± 2.37	**17.82 ± 1.94**
	16	23.95 ± 2.47	17.64 ± 2.43	24.81 ± 2.22	**17.99 ± 1.65**
MPG	2	11.92 ± 1.10	7.69 ± 1.00	12.31 ± 1.30	**9.53 ± 1.74**
	4	11.93 ± 1.09	7.74 ± 1.00	13.24 ± 1.59	**10.44 ± 2.09**
	8	11.93 ± 1.09	7.65 ± 1.11	14.78 ± 3.31	**11.51 ± 3.65**
	16	11.87 ± 0.91	8.12 ± 1.23	24.60 ± 8.51	**21.40 ± 7.87**
Abalone	2	4.99 ± 0.47	4.81 ± 0.73	4.98 ± 0.45	**4.92 ± 0.95**
	4	4.99 ± 0.47	4.81 ± 0.73	5.03 ± 0.39	**4.79 ± 0.65**
	8	4.99 ± 0.47	4.81 ± 0.65	5.26 ± 0.43	**4.98 ± 0.51**
	16	4.99 ± 0.47	4.79 ± 0.66	5.84 ± 0.73	**5.35 ± 0.52**
Boston	2	23.86 ± 5.69	11.01 ± 3.38	33.68 ± 6.10	**11.68 ± 4.05**
	4	23.88 ± 5.70	11.17 ± 3.69	24.60 ± 4.95	**12.36 ± 4.88**
	8	23.80 ± 5.79	10.93 ± 3.43	26.65 ± 5.75	**18.71 ± 5.91**
	16	24.14 ± 6.19	10.89 ± 3.90	50.86 ± 10.38	**48.57 ± 10.46**
Concrete	2	119.06 ± 7.00	82.61 ± 37.10	139.79 ± 4.13	**107.20 ± 17.76**
	4	119.06 ± 7.00	82.40 ± 37.08	120.10 ± 5.58	**111.24 ± 16.56**
	8	119.06 ± 7.00	82.24 ± 36.93	129.14 ± 4.69	**126.46 ± 12.06**
	16	119.06 ± 6.96	94.69 ± 28.62	**150.94 ± 20.14**	156.73 ± 21.73

Table 6: Result on benchmark datasets. Average and standard deviation are shown. Bold face means the best (lowest) MSE performance among methods for shuffled regression. Note that LR and DR use paired data and do not solve shuffled regression.

Setting

Data. As the regression problem instances we used four publicly available data sets provided in the UCI machine learning repository[4]: auto-MPG data (MPG), abalone data (Abalone), Boston housing data (Boston), and concrete compressive strength data (Concrete). Their sample size, n_B, and input dimension, d, are summarized in Table 5. We excluded samples with missing values and converted the categorical feature into a one-hot vector in MPG. Input features are standardized for all data sets. We also made MPG-1D whose input feature dimension is d=1 by extracting the horse power feature. We prepared 5 data sets by randomly dividing the data and using 60% for training, 20% for validation, and 20% for testing. We made the shuffled data used for training/validation by randomly dividing the training/-validation data into n sets whose number of elements is K [5] and shuffling the indices in each set. The test data are paired data for evaluation. We ran 5 trials using 5 sets of training, validation and test data.

Evaluation Metric. We use test mean squared error (test MSE) as the performance metric. Test MSE is defined as $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x^{(m)},y^{(m)})\in\mathcal{D}_{\text{test}}} \{y^{(m)} - h_\theta(x^{(m)})\}^2$, where $\mathcal{D}_{\text{test}}$ is the (paired) test data and $|\cdot|$ indicates the number of elements in the set. A lower test MSE indicates the true regression function is precisely estimated.

[4]https://archive.ics.uci.edu/ml/index.php

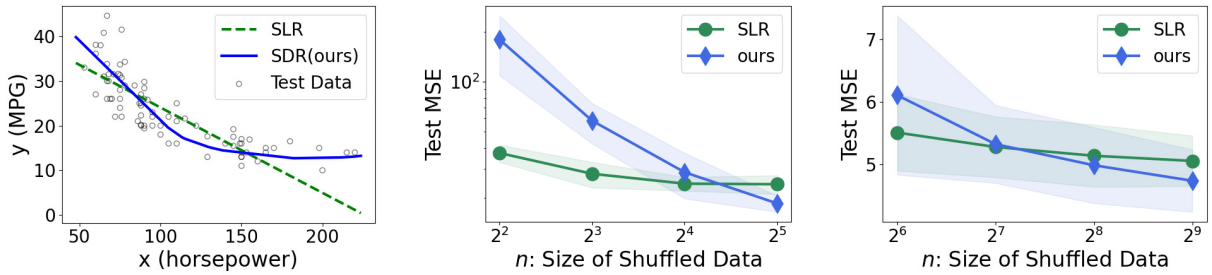
[5]Training data size n is about the 60% of the source data size divided by K. Data remaining after dividing by K were excluded.

Baselines. As one baseline, we used linear models for shuffled regression (SLR), which can be seen as an extension of (Abid and Zou 2018) for $n>1$, trained by IRLS scheme as explained in the proposed method section. As oracle baselines, we also examined (standard) linear-regression (LR) and deep-regression (DR) using paired data with data size of $K \times n$. We used normal distribution as output distribution f and the identity function as link function g in all methods.

Hyperparameters. Both DR and SDR used a one-hidden-layer feedforward neural network with the ReLU activation function. The number of units was set to 20 for all problems. The parameters were optimized using Adam (Kingma and Ba 2014) with learning rate of 0.001. The mini-batch size of SDR and that of DR were $32/K$ and 32, respectively. The maximum number of epochs was 2000, and we used early-stopping with validation data (which were also shuffled data) (Goodfellow, Bengio, and Courville 2016). The above was implemented using PyTorch (Paszke et al. 2019). Experiments were run on a computer with an Intel Xeon CPU, and a GeForce GTX TITAN GPU.

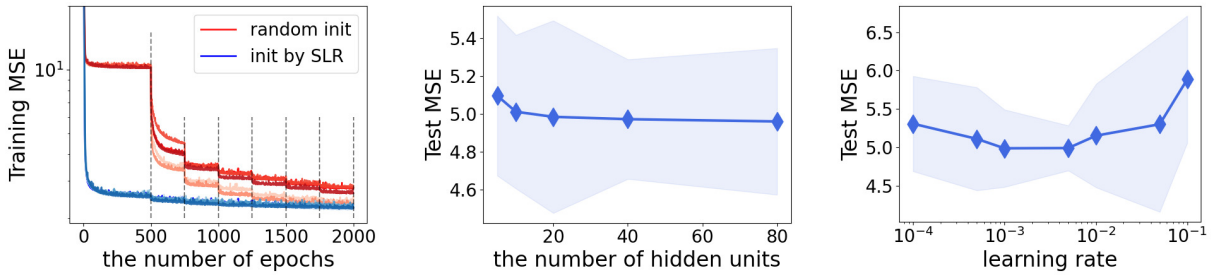
Results

Overall Results. Table 6 shows the results of the experiments. These results indicate that our SDR outperforms SLR in almost all settings. In addition, the performance of SDR exceeds LR and is comparable to DR which use paired data under some setting such as MPG-1D (K=2∼16), Abalone (K=2∼4), and Boston (K=2∼4). From Fig. 3a, we can also see that our method captures the non-linear structure behind the data, and this contributes to its improved



(a) Learned model in MPG-1D (b) MPG-1D (c) Abalone

Figure 3: Detailed comparison of SLR and SDR (ours): (a) Estimated regression lines for MPG-1D when $K = 16$, and (b)(c) MSE performance varying the size of shuffled data (n) in MPG-1D/Abalone when $K = 4$ (fixed). Average and standard deviation are shown. Lower values are better.



(a) convergence speed (b) hidden units (c) learning rate

Figure 4: Effect of hyperparameters on SDR for Abalone (K=8): (a) convergence behavior of random initialization (red) and initialization using SLR (blue) for $\tilde{\Pi}^{(i)}$ with 5 different runs. Dashed vertical lines indicate the epochs when the candidate of permutation matrices $\tilde{\Pi}^{(i)}$ is updated. (b)(c) MSE performance while varying the number of hidden units/learning rate.

performance. These results confirm the effectiveness of our method using deep models.

Effect of Data Size n. SDR and SLR have comparable performance for same dataset with small data size n, see e.g., Boston ($K = 16$) and Concrete ($K = 16$) in Table 6 [6]. So we investigated their performance while varying n (and fixing K) as shown in Fig. 3bc. These results confirm that, although SLR works well when data size n is small, the performance of SDR improves as n increases, and eventually outperforms SLR. This result is convincing since deep learning-based methods require a large amount of data to perform well in general. Therefore, it is confirmed that the proposed deep learning-based method (SDR) outperforms the linear model-based method (SLR) in solving the shuffled regression problem when sufficient number of data is available, similar to the standard regression problem.

Effect of Hyperparameters. We also investigated the impact of hyperparameters on SDR as shown in Fig. 4. Figure 4a shows the convergence property of SDR with random initialization and with initialization using SLR for the candidate of permutation matrices $\tilde{\Pi}^{(i)}$. We can see that SDR with the random initialization shows slow convergence, but that with SLR initialization shows faster convergence for all five runs. That implies that the initial values should be set carefully, and appropriate choice can significantly accelerate convergence speed. Figure 4b shows that, although the MSE performance is degraded when the number of units is smaller than 10, the performance is stable when the number of units is larger than 10. Similarly, the stable performance is confirmed when the learning rate is set to values between 10^{-3} and 10^{-2} from Fig. 4c. So we can say that SDR works well without the need for sensitive tuning of these two hyperparameters. This supports the usefulness of our method.

Conclusion

In this study, we proposed SDR for learning deep regression models from shuffled data. We derived the SSEM algorithm for parameter estimation and confirmed the effectiveness of the proposals by experiments on benchmark datasets. The remaining work of this study is to theoretically analyze how data size n, set size K, and input dimension d affect the performance.

[6] n=18 for Boston (K=16) and n=38 for Concrete (K=16)

References

Abid, A.; and Zou, J. 2018. A stochastic expectation-maximization approach to shuffled linear regression. In *Annual Allerton Conference on Communication, Control, and Computing*, 470–477.

Bishop, C. M. 2006. *Pattern recognition and machine learning*. Springer.

Cappé, O.; and Moulines, E. 2009. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 71(3): 593–613.

Carpentier, A.; and Schlüter, T. 2016. Learning relationships between data obtained independently. In *Artificial Intelligence and Statistics*, 658–666.

David, P.; Dementhon, D.; Duraiswami, R.; and Samet, H. 2004. SoftPOSIT: Simultaneous pose and correspondence determination. *International Journal of Computer Vision*, 59: 259–284.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1): 1–22.

Fang, G.; and Li, P. 2023. Regression with label permutation in generalized linear model. In *International Conference on Machine Learning*, 9716–9760.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press Cambridge.

Hsu, D. J.; Shi, K.; and Sun, X. 2017. Linear regression without correspondence. In *Advances in Neural Information Processing Systems*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. arXiv:1412.6980.

Kohjima, M.; Nambu, Y.; Kurauchi, Y.; and Yamamoto, R. 2022. General Algorithm for Learning from Grouped Uncoupled Data and Pairwise Comparison Data. In *International Conference on Neural Information Processing*, 153–164.

Neal, R. M.; and Hinton, G. E. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in graphical models*, 355–368.

Nelder, J. A.; and Wedderburn, R. W. 1972. Generalized linear models. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 135(3): 370–384.

Pananjady, A.; Wainwright, M. J.; and Courtade, T. A. 2017. Linear regression with shuffled data: Statistical and computational limits of permutation recovery. *IEEE Transactions on Information Theory*, 64(5): 3286–3300.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, 8024–8035.

Ranganath, R.; Tang, L.; Charlin, L.; and Blei, D. 2015. Deep exponential families. In *Artificial Intelligence and Statistics*, 762–771.

Rubin, D. B. 2004. Iteratively reweighted least squares. *Encyclopedia of statistical sciences*, 6.

Shi, X.; Li, X.; and Cai, T. 2021. Spherical regression under mismatch corruption with application to automated knowledge translation. *Journal of the American Statistical Association*, 116(536): 1953–1964.

Slawski, M.; Rahmani, M.; and Li, P. 2020. A Sparse Representation-Based Approach to Linear Regression with Partially Shuffled Labels. In *Uncertainty in Artificial Intelligence*, 38–48.

Unnikrishnan, J.; Haghighatshoar, S.; and Vetterli, M. 2018. Unlabeled sensing with random linear measurements. *IEEE Transactions on Information Theory*, 64(5): 3237–3253.

Xu, L.; Niu, G.; Honda, J.; and Sugiyama, M. 2019. Uncoupled regression from pairwise comparison data. In *Advances in Neural Information Processing Systems*, 3992–4002.

Yamane, I.; Chevaleyre, Y.; Ishida, T.; and Yger, F. 2023. Mediated Uncoupled Learning and Validation with Bregman Divergences: Loss Family with Maximal Generality. In *Artificial Intelligence and Statistics*, 4768–4801.