Shuffled Regression and Classification

Arpit Gupta 160149
Hritvik Taneja 160300
Shashank Gupta 160643

1 Problem Statement

We started this project by looking at the problem of shuffled linear regression. In traditional inference settings, we assume that the data we have is fully and properly labelled with target labels i.e. we work in a fully supervised settings. Many modern datasets do not meet this criteria. On the contrary, it is often easier to obtain some form of partial labelling e.g. labels for some other task, high-level information, etc. Such situations are those of weak supervision, where the labels cannot be directly used, and inference must leverage them in some creative way. Shuffled linear regression is an instance of weak supervision, in which all of the labels are available, but are present in some arbitrary order which is unknown to us. Mathematically, this may be represented as multiplying a vector of labels with a permutation matrix. More formally, our model is

$$\boldsymbol{y} = \boldsymbol{\Pi} \boldsymbol{X} \boldsymbol{w} + \boldsymbol{\epsilon}$$

where $\boldsymbol{y}$ is the label of vectors, $\boldsymbol{X}$ is the matrix of training points, $\boldsymbol{\Pi}$ is an unknown permutation matrix, $\boldsymbol{w}$ is the vector of weights and $\boldsymbol{\epsilon}$ is Gaussian noise. Our objective is to learn the optimal $\boldsymbol{w}$ and $\boldsymbol{\Pi}$. This model is useful for a variety of scientific and real-world settings, such as flow cyclometry experiments and reversing data anonymization.

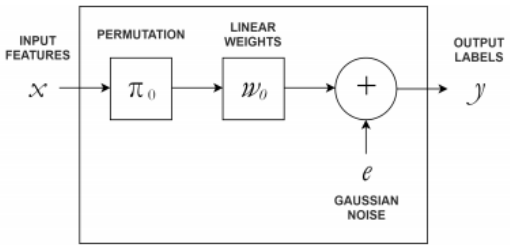


Figure 1: Shuffled Linear Regression

Our reading naturally led us to explore the general problem of matching different sets with each other, and the learning of arbitrary permutation matrices. Matching problems typically assume two sets containing certain objects (e.g. a set of videos and a set of their captions, or words in two different languages), where a one-to-one correspondence must be between items in the sets. As for the latter, permutation matrices arise in a variety of different settings, especially graph theoretic problems. As these matrices are discrete and constrained (rows and columns must sum up to 1), optimizing them is usually NP-hard, and relaxations into the real domain are applied to get an approximate solution. Recently, there has been work on reinforcement learning approaches to tackle these problems, which we take a look at as well. Our original problem may be thought of in this context, where an agent is learning the best policy to reverse the permutation, which jointly learning an optimal weight vector. We then present our experiments, which are mainly with Deep Matching Autoencoders, an architecture developed by Mukherjee et al[8] for the matching problem.

2 Prior Work

There hasn't been much work specifically in the area of shuffled linear regression. But there are some techniques for the basic matching problem. The techniques used in matching problems can be used to to learn the permutation matrix in shuffled regression to learning the mapping, thereafter using general linear regression techniques. We describe the prominent work in the following sections. Some of the less relevant work to our problem can be found in [3], [4], [5], [6], [7], [9], [10], [11].

3 Least Square and Self Moment Matching Estimator

[1] explores the problem of shuffled regression by proposing several estimators that recover the weights of a noisy linear model from labels that are shuffled by an unknown permutation. The methods proposed circumvent the need to estimate the permutation matrix by applying transformations and leveraging the structure of the model. The authors explore two major estimators for this task - a Least Square (LS) Estimator which is analogous to the classical least-squares estimator (termed as the ordinary least squares (OLS) estimator) and an estimator based on the self-moments (SM) of input features and labels.

3.1 LS Estimator

The LS estimator for shuffled regression is the natural extension of the ordinary least square estimator. It searches for the weight that minimizes the least square distance across all the different $N \times N$ permutation matrices Π.

$$\hat{\mathbf{w}}_{LS} = \arg\min_{\mathbf{w}} ||\Pi \mathbf{X}\mathbf{w} - \mathbf{y}||$$

Albeit the OLS estimator is prominent to be consistent for standard linear regression, the LS estimator is not consistent for shuffled linear regression

It may appear at first that utilizing the LS estimator to estimate the weights is computationally intractable, because it requires probing over the space of $n!$ permutation matrices. However, the LS estimator turns out to be computationally feasible. This is because of the fact that we can eliminate the optimization over permutations by re-inditing the above equation as

$$\begin{aligned} L(\boldsymbol{x}, y, w) &= |(\boldsymbol{x}w)^{\uparrow} - y^{\uparrow}|^2 \\ \hat{w}_{\text{LS}} &= \arg\min_{w} L(\boldsymbol{x}, y, w) \end{aligned} \tag{1}$$

But it can be seen that this approach can't be elongated to multidimensional input features. Hence we require some other estimators.

3.2 SM Estimator

Method of moments generally incorporates both the *self-moments*, such as the mean of y or the variance of the second column of $\boldsymbol{x}$, as well as the *cross-moments*, such as the co-variance of the first column of $\boldsymbol{x}$ and y. In our problem setting, without knowing the mutual authoritatively mandating between $\boldsymbol{x}$ and y, we can only calculate the self-moments of $\boldsymbol{x}$ and y. Thus, the self-moments (SM) estimator, $\hat{w}_{\text{SM}}$, estimates w_0 by constraining moments of $\boldsymbol{x} \cdot w$ to equal the respective moments of y. The moments involved in these constraints depend on the value of d, and we discuss concrete cases below.

3.2.1 $d = 1$

For $d = 1$, $\boldsymbol{x}$ is unidimensional (there is no separate intercept term). So we require only one constraint to estimate w_0, which is a scalar. We can thus inscribe down the following constraint for

$\hat{w}_{\text{SM}}$, predicated on the sample betokens:

$$\frac{1}{n}\sum_{i}^{n} x_i \hat{w}_{\text{SM}} = \frac{1}{n}\sum_{i}^{n} y_i \implies \hat{w}_{\text{SM}} = \frac{\sum_i^n y_i}{\sum_i^n x_i}, \tag{2}$$

where x_i is used to refer to the i^{th} entry of the vector $\boldsymbol{x}$, and the same for y_i and y. It can be proved that with a mild condition on the mean of $\boldsymbol{x}$, the SM estimator is unbiased and consistent.

3.2.2 $d > 2$

For $d > 2$, there are d separate unknown linear weights, so we must indite $K \geq d$ equations, each one incorporating a higher moment. The K equations are, for $1 \leq k \leq K$, constraints of the following form:

$$\underbrace{\frac{1}{n}\sum_{i}^{n}(x_i w)^k}_{M_k} = \underbrace{\frac{1}{n}\sum_{i}^{n} y_i^k}_{N_k}. \tag{3}$$

For ease, we denote the pertinent expression for the k^{th} moment of y in (3) as N_k and the expression for the k^{th} moment of $\boldsymbol{x}\cdot\hat{w}$ as M_k. It is generally not possible to solve these D equations analytically; in fact, there may not be a solution to the system. As a result, we instead inscribe a single cost function that minimizes the extent that each of the D constraints are infringed:

$$\begin{aligned} L(\boldsymbol{x}, y, w) &= \sum_{k=1}^{K} f(k)(M_k - N_k)^2 \\ \hat{w}_{\text{SM}} &= \arg\min_{w} L(\boldsymbol{x}, y, w). \end{aligned} \tag{4}$$

Here, the choice of the function $f(k)$ affects the relative contribution from each moment condition. One possible function is $f(k) = k!^{-1}$, which weighs each moment inversely proportional to the expected variance of the k^{th} sample moment of samples taken from a Gaussian distribution. Once we have designated, we minimize it across $\hat{w}$, utilizing standard numerical optimization techniques. As is the case with $d = 2$, this estimator is not consistent. To make it consistent, we require to incorporate moments of the noise source. Furthermore, we note that higher sample moments are liable to exhibit paramount variance, reducing the efficiency of the SM estimator for incrementing values of d. As a result, we find that the SM estimator can have the same or worse efficiency than the LS estimator when d is even just 3.

3.3 Hybrid Estimators

The SM estimator has the advantage of being consistent (for $d = 1$ and, with erudition of the noise characteristics, for higher dimensions as well). However, in the higher-dimensional setting, it exhibits consequential error because it requires computing higher sample moments. In this setting, the LS estimator is more efficient. We can construct a hybrid estimator that has advantages of both.

We do this by projecting $\boldsymbol{x}$ to a lower dimension, and then utilizing the SM estimator. For example, let us consider the categorical case of projecting $\boldsymbol{x}$ to 1-dimensional space, by utilizing a $d \times 1$ projection matrix, p, afore utilizing the SM estimator. The result is an estimate $\tilde{w}$ defined by

$$\frac{1}{n}\sum_{i}^{n} x_i p \tilde{w} = \frac{1}{n}\sum_{i}^{n} y_i \implies \tilde{w} = \frac{\sum_i^n y_i}{\sum_i^n x_i p}, \tag{5}$$

which can be embedded in the pristine d-dimensional space utilizing the same matrix p to engender $\hat{w}_{\text{P1}} \equiv p\tilde{w}$. Note that $\hat{w}_{\text{P1}}$ satiates the first-moment condition, namely that the mean of $\boldsymbol{x} \cdot \hat{w}_{\text{P1}}$ is equipollent to the mean of y, since:

$$\frac{1}{n}\sum_{i}^{n} x_i \hat{w}_{\text{P1}} = \frac{1}{n}\sum_{i}^{n} x_i p\tilde{w} = \frac{1}{n}\sum_{i}^{n} y_i \tag{6}$$

However, $\hat{w}_{\text{P1h}}$ may still be a poor estimate of w_0 – it depends entirely on the choice of the projection matrix p. This suggests probing over the *projection matrix*, in lieu of directly over the weights, and then utilizing the criterion of minimizing the least-squares distinction between $(\boldsymbol{x} \cdot \hat{w}_{\text{P1}})^{\uparrow}$ and $y^{\uparrow}$ to determine the best p.
This approach, which we refer to as the P1 estimator, efficaciously applies the LS estimator *only* to those weights which slake at least the first-moment condition. We can similarly elongate this approach to the second moment by projecting to 2 dimensions (denote this as the P2 estimator), and so on.

4 Stochastic EM

In [2], the authors propose a latent variable model for the problem, and use EM to estimate the weights. For this, we require the posterior over the permutation matrices, say given by $q(\mathbf{\Pi})$. While this may be computed, there are $n!$ terms in the distribution, which makes computing the expectation for this prohibitive expensive to compute. However, the M step cannot be computed without this expectation. The authors propose two variants to the standard EM approach, given as follows.

4.1 Soft EM

The expectation is thus replaced by a Monte Carlo average, where samples are drawn from the posterior $p(\Pi|X, y, w)$. Note that this Monte Carlo average is no longer a permutation matrix, and constitutes a relaxation of the binary doubly stochastic constraint. The samples are drawn using the Metropolis-Hastings algorithm over a Markov chain defined on the set of all permutations. The different permutations form states, and transitions exist between matrices that differ in exactly two positions. and the transition probability is given by max $(1, \alpha(a, b))$, where $\alpha(a, b)$ represents the ratio of the posterior probability of Π_a and Π_b.

The authors use a uniform prior over the permutations (which feels reasonable), and hence the posterior is equivalent to the joint likelihood. A non-uniform prior would be useful in the case of partial shuffling. $\alpha(a, b)$ is given by

$$\frac{p(\Pi_a, X, y, \hat{w}_m)}{p(\Pi_b, X, y, \hat{w}_m)} = \frac{\exp(-||\Pi_a X\hat{w}_m - y||^2/2\hat{\sigma}_m)}{\exp(-||\Pi_b X\hat{w}_m - y||^2/2\hat{\sigma}_m)}$$

The E and M steps may be written as

E step: Approximate $\hat{\Pi}_m$ (the expectation of the posterior) with MH sampling using $\hat{w}_m, \hat{\sigma}_m^2$

M step: $\hat{w}_m = (X^T X)^{-1} X^T \hat{\Pi}_m y, \hat{\sigma}_m = \dfrac{1}{N - D}||X\hat{w}_m - y||^2$

An empirical estimate is used for σ^2, which could probably be improved by using a suitable prior.

4.2 Hard EM

In this approach, the expectation is replaced by the probability of the permutation which maximises the following condition

$$\hat{\Pi}_m = \arg\max_{\Pi} p(\Pi|X, y, \hat{w}_m)$$

The authors show that this is equivalent to finding the permutation that reorders the elements of y according to the order of $X\hat{w}_m$. This essentially reduces into an ALT-OPT procedure for Π and w.

4.2.1 Results

The Soft EM procedure performs better than the Hard EM procedure in the case of synthetic data and partially ordered real data, with smaller improvements in the case of fully shuffled real data. The reason for this could be that Hard EM is extremely sensitive to the initialisation of w both in its computations and given that the loss function is highly non-convex. Soft EM reduces this dependency with soft updates.

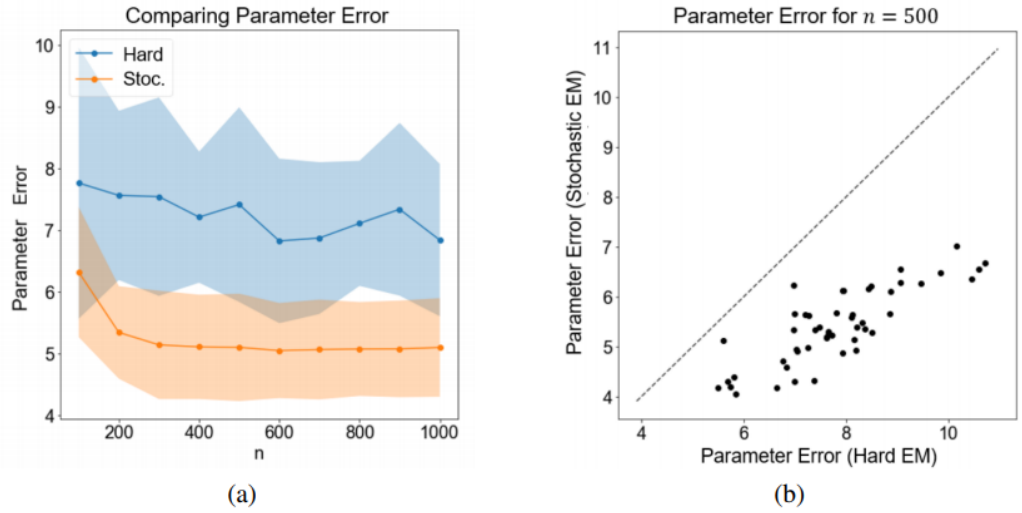


Figure 2: Parameter error for Hard EM and Stochastic EM. (a) The graph shows the standard deviation and average parameter errors. (b) Each dot in this graph shows the parameter error produced by Stochastic EM and Hard EM for one dataset

A major challenge faced was that of overfitting (as termed by the authors), where the noise caused another permutation to be more likely than the true permutation. This was especially pernicious in the case of real data, where the noise is heteroscedastic, non-AWGN and/or large in magnitude, and when the relationship between X and y is not linear. This was why the algorithm did not do so well on real datasets.

5 Deep Matching Autoencoder

5.1 Multi-View Autoencoders

Let us denote the autoencoders of $\boldsymbol{x}$ and $\boldsymbol{y}$ as

$$\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{x};\boldsymbol{\Theta}_{\boldsymbol{x}});\boldsymbol{\Theta}_{\boldsymbol{x}}), \quad \boldsymbol{f}_{\boldsymbol{y}}(\boldsymbol{g}_{\boldsymbol{y}}(\boldsymbol{y};\boldsymbol{\Theta}_{\boldsymbol{y}});\boldsymbol{\Theta}_{\boldsymbol{y}}),$$

where $\boldsymbol{g}(\cdot)$ is an encoder and $\boldsymbol{f}(\cdot)$ is a decoder function, $\boldsymbol{\Theta}_{\boldsymbol{x}}$, and $\boldsymbol{\Theta}_{\boldsymbol{y}}$ are the autoencoder parameters. Our goal is to learn commensurable representation embeddings $\boldsymbol{g}_x(\cdot)$ and $\boldsymbol{g}_y(\cdot)$ given no paired training data. This is a significantly harder quandary than other multi-modal autoencoder approaches that rely on paired data

5.2 Learning from Unpaired Data

To learn from unpaired data we introduce a permutation matrix to represent the unknown correspondence between data items in two views . Let π be an permutation function over $\{1, 2, \ldots, n\}$, and let $\boldsymbol{\Pi}$ be the corresponding permutation bespeaker matrix:

$$\boldsymbol{\Pi} \in \{0,1\}^{n\times n}, \boldsymbol{\Pi}\mathbf{1}_n = \mathbf{1}_n, \text{and } \boldsymbol{\Pi}^\top \mathbf{1}_n = \mathbf{1}_n,$$

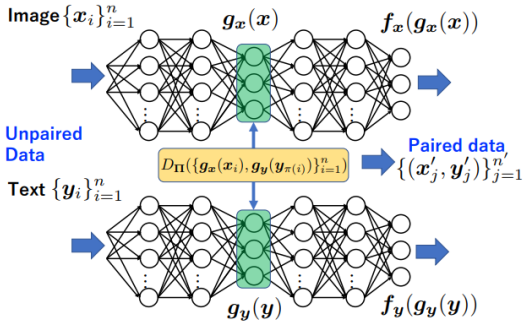


Figure 3: Deep Matching Autoencoder Architecture

where $\mathbf{1}_n$ is the n-dimensional vector with all ones.

Then, we consider the following optimization problem:

$$\min_{\boldsymbol{\Theta}_{\boldsymbol{x}}, \boldsymbol{\Theta}_{\boldsymbol{y}}, \boldsymbol{\Pi}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{x}_i))\|_2^2 + \|\boldsymbol{y}_i - \boldsymbol{f}_{\boldsymbol{y}}(\boldsymbol{g}_{\boldsymbol{y}}(\boldsymbol{y}_i)))\|_2^2 - \lambda D_{\Pi}(\{\boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{x}_i), \boldsymbol{g}_{\boldsymbol{y}}(\boldsymbol{y}_{\pi(i)})\}_{i=1}^n), \tag{7}$$

where we are simultaneously optimising the autoencoders ($\boldsymbol{\Theta}_{\boldsymbol{x}}$ and $\boldsymbol{\Theta}_{\boldsymbol{y}}$) as well as the cross-domain match ($\boldsymbol{\Pi}$) with tradeoff parameter λ. The key component here is the function $D_{\boldsymbol{\Pi}}(\cdot, \cdot)$ which is a non-negative statistical dependence measure between the x and y views. $D_{\boldsymbol{\Pi}}(\cdot, \cdot)$ needs to be a quantification which does not require commensurable representations *a priori* in order to enable learning to get commenced.

5.3 Dependence Measures

The statistical dependence measure is the crucial component in achieving our goal. In this paper, we explore two alternatives: the unnormalized kernel target alignment (KTA) [**?**] and the squared-loss mutual information (SMI) [**?**, **?**, **?**]. Note that SMI is an independence measure. However, since we optate to make $\boldsymbol{\Theta}_{\boldsymbol{x}}$ and $\boldsymbol{\Theta}_{\boldsymbol{y}}$ engender kindred representations, we utilize SMI as a dependence measure.

Unnormalized kernel target alignment (uKTA) In uKTA [**?**], we consider the following similarity function D for paired data:

$$\text{uKTA}(\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n) = \text{tr}\left(\boldsymbol{K}\boldsymbol{L}\right),$$

where $\text{tr}(\cdot)$ is the trace operator, $\boldsymbol{K}$ is the Gram matrix for $\boldsymbol{x}$ and $\boldsymbol{L}$ is the Gram matrix for $\boldsymbol{y}$. This homogeneous attribute function takes sizably voluminous value if the Gram matrices $\boldsymbol{K}$ and $\boldsymbol{L}$ are homogeneous, and a minute value if they are not homogeneous.

We use the Gaussian kernel:

$$\boldsymbol{K}_{ij} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{2\sigma_x^2}\right), \ \boldsymbol{L}_{ij} = \exp\left(-\frac{\|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2^2}{2\sigma_y^2}\right),$$

where $\sigma_y > 0$ and $\sigma_x > 0$ are the Gaussian width.

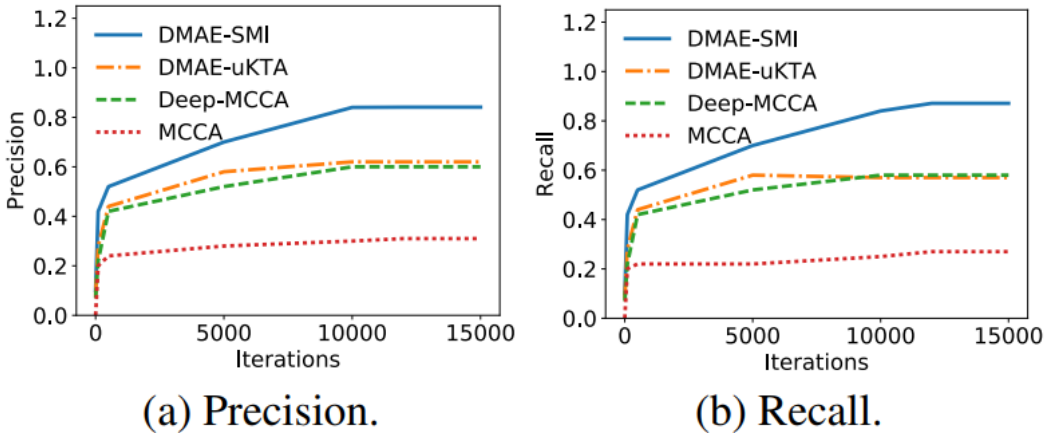
Squared-Loss Mutual Information (SMI) The squared loss mutual information (SMI) between two random variables is defined as [**?**]

$$\text{SMI} = \iint \left(\frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x})p(\boldsymbol{y})} - 1\right)^2 p(\boldsymbol{x})p(\boldsymbol{y})\text{d}\boldsymbol{x}\text{d}\boldsymbol{y},$$

which is the Pearson divergence [**?**] from $p(\boldsymbol{x}, \boldsymbol{y})$ to $p(\boldsymbol{x})p(\boldsymbol{y})$. The SMI is an f-divergence [**?**] that is it is a non-negative measure and is zero only if the random variables are independent.

5.4 Results

Deep Matching Autoencoder outperforms all the existing unsupervised matching algorithms and stands apart by achieving high accuracy. The plots below show the matrix prediction accuracy during unsupervised classifier learning.



(a) Precision. (b) Recall.

Figure 4: Evolution of Π (label) matrix prediction accuracy during unsupervised classifier learning

6 Experiments

The code was not available for the Deep Matching Autoencoders. Hence, we wrote the code and tried to analyze the utilization of the idea in a new setting. We used dictionaries and word2Vec embeddings for the evaluation of our code. We used individual German and Spanish word2Vec embeddings and tried to learn the mapping of german words to spanish words by providing 5000 shuffled words, i.e. our dataset included the word2Vec embeddings of upto 5000 german words along with their spanish meaning words without correspondence. The following table shows average number of correct assignments done by random assignments and Deep Matching Autoencoders:

Number of Words	Random Assignment	Deep Matching Autoencoders
100	1.127	12.25
1000	11.27	132.37
5000	56.35	786.45

We also tried to learn the transformation matrix from one language to another in the embeddings (encoded). This was achieved by modelling spanish embeddings as a linear transformation of the german embeddings. The results that we get are not very ideal closely resembles random assignments. This is primarily because we model the complex relationship betwenn the embeddings of two languages in a linear fashion. Following table shows the average number of correct assignments on testing data (100 words):

Number of Words in Training Data	Random Assignment	Deep Matching Autoencoders
100	1.127	2.25
1000	1.127	3.21
5000	1.127	3.5

7 Problems Faced

There were many problems that we faced during the course of this project. Firstly, the problem of shuffled regression itself is a hard problem since it can be shown that computing a permutation matrix which gives least squared loss is NP-Hard.

While doing experiments, we found that the determinant of the permutation matrix (relaxed) shoots up after a certain number of epochs (around 30-35). This caused the determinant to diverge and hence, the learning had to be stopped thereon. We tried to solve this by making a naive assumption that the normalization of the permutation matrix doesn't affect the core algorithm. This turns out to be a wrong assumption and the training gets worse with this.

8 Description of tools used

All the model were programmed and tested either in Pytorch. Pytorch is an automatic differentiation libraries, which has become really popular for Deep Learning models. Using these kind of libraries, one can define the forward propagation in arbitrary networks, and these libraries will compute the backpropagation itself.

9 Things we Learnt

We learnt a lot from this project:

- We surveyed a lot of papers understanding the existing literature thoroughly. This gave us a wholesome view of the field of matching problems
- We came up with a lot of problems and some possible approaches during the course of this project. These approaches were tried out in different settings. Moreover, these approaches were well-thought out and experimented with than we began with, thus giving us valuable lessons in research.
- We became more familiar with implementing deep learning models, going from an idea to implementation quickly. We also gained more proficiency in PyTorch.

10 Future Work

We think that, after surveying a lot of literature in shuffled regression and in general matching problem, the problem is quite interesting and has scope for finding an optimal way of learning the permutation matrix efficiently. We also think that following points can be taken into into consideration while exploring more on this problem:

- The encoded embeddings of different languages can be used to learn a transformation function from one language to another. We did this using a simple matrix. But in general this can be extended to different Deep Learning models.
- The problem of shuffled linear regression can be solved by adding the a lost function in the main objective function only. This woould mean learning Π, w simultaneously. Bayesian linear regression could also be used after learning the permutation matrix, by taking the MAP of Π as the true matching in raining data.
- Also there are some techniques used in Reinforcement Learning to learn stochastic matrices. Those methods could be looked upon to learn permutation matrix in a similar way.

11 Acknowledgement

We would like to thank the authors of the research papers mentioned in Reference for their contribution to the community. Also, we would like to acknowledge them for the various figures and plots present in our report

References

[1] Abubakar Abid, Ada Poon, and James Zou. Linear regression with shuffled labels. *arXiv preprint arXiv:1705.01342*, 2017.

[2] Abubakar Abid and James Zou. Stochastic EM for Shuffled Linear Regression. *arXiv e-prints*, page arXiv:1804.00681, Apr 2018.

[3] Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. *arXiv preprint arXiv:1805.07010*, 2018.

[4] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[5] Daniel J Hsu, Kevin Shi, and Xiaorui Sun. Linear regression without correspondence. In *Advances in Neural Information Processing Systems*, pages 1531–1540, 2017.

[6] Jiancheng Lyu, Shuai Zhang, Yingyong Qi, and Jack Xin. Autoshufflenet: Learning permutation matrices via an exact lipschitz continuous penalty in deep convolutional neural networks. *arXiv preprint arXiv:1901.08624*, 2019.

[7] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

[8] Tanmoy Mukherjee, Makoto Yamada, and Timothy M Hospedales. Deep matching autoencoders. *arXiv preprint arXiv:1711.06047*, 2017.

[9] Ashwin Pananjady, Martin J Wainwright, and Thomas A Courtade. Linear regression with shuffled data: Statistical and computational limits of permutation recovery. *IEEE Transactions on Information Theory*, 64(5):3286–3300, 2018.

[10] Martin Slawski, Emanuel Ben-David, et al. Linear regression with sparsely permuted data. *Electronic Journal of Statistics*, 13(1):1–36, 2019.

[11] Manolis C Tsakiris, Liangzu Peng, Aldo Conca, Laurent Kneip, Yuanming Shi, and Hayoung Choi. An algebraic-geometric approach to shuffled linear regression. *arXiv preprint arXiv:1810.05440*, 2018.