MixR: Data Mixing Augmentation for Regression

Seong-Hyeon Hwang
KAIST
Daejeon, Republic of Korea
sh.hwang@kaist.ac.kr

Steven Euijong Whang
KAIST
Daejeon, Republic of Korea
swhang@kaist.ac.kr

Abstract

Data augmentation is becoming essential for improving regression accuracy in critical applications including manufacturing, climate prediction, and finance. Existing techniques for data augmentation largely focus on classification tasks and do not readily apply to regression tasks. In particular, the recent Mixup techniques for classification have succeeded in improving the model performance, which is reasonable due to the characteristics of the classification task, but has limitations in regression. We show that mixing examples that have large data distances using linear interpolations may have increasingly-negative effects on model performance. Hence, we use the stricter assumption that linearity only holds within certain data distances for regression where the degree may vary by each example. We then propose MixR, a data augmentation framework for regression that learns for each example how many nearest neighbors it should be mixed with for the best model performance using a validation set. Our experiments conducted both on synthetic and real datasets show that MixR significantly outperforms state-of-the-art data augmentation baselines applicable to regression. MixR can also be integrated with existing Mixup techniques to significantly improve their performances.

1 Introduction

As machine learning (ML) becomes widely used in critical applications including manufacturing, climate prediction, and finance, data augmentation for regression becomes essential as it provides an opportunity to improve model performance without additional data collection. In comparison to classification tasks such as object detection in images, regression tasks predict real numbers.

To emphasize the importance of data augmentation in regression, we provide a case study of semiconductor manufacturing. Here a common quality check is to measure the layer thicknesses of a 3-dimensional semiconductor and see if they are even. However, directly measuring each thickness results in destroying the semiconductor itself, so a recently-common approach is to take an indirect measurement by applying light waves on the semiconductor, measuring the spectrum of wavelengths that bounce back from all the layers, and use ML to predict the layer thicknesses from the spectrum data (see Fig. 4 in Sec. 4 for an illustration). With enough spectrum data and thickness information, ML models can be trained to accurately predict thicknesses from a spectrum. The main challenge is that there is not enough training data, and the only cost-effective solution is to augment small amounts of labeled data that can be produced. According to collaborators in this industry, modeling the physical system using simulators is not practical due to the the high dimensional and complex situation. Even a small improvement in model performance from the data augmentation has significant impact. In general, any regression task that predicts real values such as emissions, stock prices, or various measurements can also benefit from data augmentation.

Most data augmentation techniques are designed for image classification. In particular, Mixup [Zhang et al., 2018, Berthelot et al., 2019, Yun et al., 2019] is a popular data augmentation technique that is widely used for classification tasks, but is seldom used for regression because it relies on a one-hot
encoded label space. The idea of Mixup is to mix two examples by linear interpolation and use it to estimate the label of any examples in between. Mixup is known to effectively regularize the model being trained. Recently, Manifold Mixup [Verma et al., 2019] has been proposed to improve the hidden representation and decision boundaries where two examples are mixed in multiple hidden layers of a neural network.

However, the Mixup techniques are not readily applicable to a regression setting because taking linear interpolations may result in mixed examples that are unsuitable for training. Since the label space is continuous, interpolating examples that are very different may result in arbitrarily-incorrect labels as shown in Fig. 1a. Hence, the interpolations must be performed in a more limited fashion. Moreover, other data augmentation techniques for classification including image processing (e.g., flipping or rotating) and generative models (e.g., GAN [Goodfellow et al., 2014] and VAE [Kingma and Welling, 2014]) are even less applicable to a regression setting (see Sec. 5).

We propose MixR, a data mixing augmentation framework that is the first to tailor Mixup for regression tasks. MixR assumes that linear interpolations when mixing examples are only effective within certain data distances (i.e., the linearity of a regression model is limited). The distance limit may vary by example, and we formulate the problem of learning for each example how many nearest neighbors it should be mixed with, which we hereby refer to as a kNN mixing policy. MixR employs a framework that searches for these policies that lead to the lowest model losses on a validation set using reinforcement learning (RL) [Zoph and Le, 2017], although other search algorithms like evolution [Ho et al., 2019] can be used as well. Our approach is inspired by the AutoAugment framework [Cubuk et al., 2019, Lim et al., 2019, Zhang et al., 2020], which searches image augmentation policies for classification. A key difference is that AutoAugment searches for the best augmentation operators that are applied to the entire data, while MixR finds the number of neighbors to mix for each example. Fig. 1b shows how limiting the nearest neighbors to mix is better than mixing with all neighbors as in classification. To see if the augmentation is useful, we train simple models on the original four examples (i.e., no augmentation), the augmented data in Fig. 1a, and the augmented data in Fig. 1b. Evaluating the models on 20 random test examples results in Root Mean Square Error (RMSE; see Sec. 4) values of 0.6673, 1.0473, and 0.4257, respectively, where a lower RMSE is better. We thus conclude that carefully mixing examples is important for improving regression performance.

Experiments conducted on real and synthetic datasets show that MixR outperforms baselines, especially when the linearity is limited, and the mixing must be done selectively.

## 2 Limitations of Mixup in Regression

We explain why Mixup in classification has limitations in a regression setting. In most cases, the labels in classification are one-hot encoded where many examples may have the same label. The original version of Mixup [Zhang et al., 2018] is to take a linear interpolation between any pair of examples \( x_i \) and \( x_j \) with the labels \( y_i \) and \( y_j \) to produce the new example \( \lambda x_i + (1 - \lambda) x_j \) with the label \( \lambda y_i + (1 - \lambda) y_j \) where \( \lambda \sim \text{Beta}(\alpha, \alpha) \). According to Zhang et al., 2018, mixing all examples outperforms Empirical Risk Minimization on many classification datasets. This strategy works well
for classification where the goal is to figure out whether an example belongs to a certain class more than others. There is no notion of distance among the classes, so mixing examples incorrectly cannot lead to “larger” misclassifications than others either. Hence, there is more benefit in mixing examples regardless of their labels than trying to be more selective in which examples to mix.

In regression, however, such linear interpolations are not suitable as labels are in a continuous space instead of a discrete space where one-hot encodings cannot be used. Here two distant examples may have labels that are arbitrarily different (e.g., 0.1, 1, 10, or any larger value), and simply mixing them could result in intermediate labels that are very different than the actual label. For example, mixing the points $a$ and $d$ in Fig. 1a results in a label nowhere near the actual label. In Sec. 4.1 we also show empirical results where the label error increases for larger data distances. Furthermore, mixing examples with larger data distances tend to have increasingly-negative effects on the model trained on the augmented training set. Fig. 2 shows the model accuracies using RMSE for three real datasets NO2, Product, and Bike (described in Sec. 4) when mixing examples with different ranges of distances. As the data distance increases, the RMSE initially decreases, but then increases to the extent that mixing is not beneficial on average for all datasets.

Our key approach is thus to limit the data distance to improve Mixup for regression. To give a simple analysis on why this approach works, suppose that the regression model $f$ is continuous where $\lim_{x \to c} f(x) = f(c)$ for any $x$ and $c$ within the domain of $f$. We show that a short-enough data distance sufficiently reduces the label distance as well. Given $f$’s domain $D$ and $\lim_{x \to c} f(x) = L$, the following is known to hold: $\forall \epsilon, \exists \delta$ s.t. $\forall x \in D$, if $|x - c| < \delta$, then $|f(x) - L| < \epsilon$. We can use this result to prove that $\forall \epsilon, \exists \delta$ s.t. $\forall x_i, x_j \in D$, if $\alpha x_i + (1 - \alpha) x_j = c$, $0 \leq \alpha \leq 1$, and $|x_i - x_j| < \delta$ then the absolute difference between the mixed example’s $x_j$ value and $L$ is small where $|\alpha f(x_i) + (1 - \alpha) f(x_j) - L| = |\alpha (f(x_i) - L) + (1 - \alpha) (f(x_j) - L)| \leq \alpha |f(x_i) - L| + (1 - \alpha) |f(x_j) - L| < \alpha \epsilon + (1 - \alpha) \epsilon = \epsilon$. While this analysis does not cover all edge cases (e.g., identical examples having different labels), it does give an intuition why limiting the data distances for mixing is a reasonable solution.

3 MixR

The goal of MixR is to identify which examples to mix with which nearest neighbors. Instead of finding the actual distance limits themselves, we solve the identical problem of finding the number of nearest neighbors to mix per example for convenience. We use RL to implement MixR. Our framework is inspired by AutoAugment [Cubuk et al., 2019], but we solve the new problem of mixing examples for regression.

3.1 Framework

Let $D = \{(x_i, y_i)\}_{i=1}^{n} \sim P$ be the training set where $x_i \in X$ is a $d$-dimensional input example, and $y_i \in Y$ is an $\epsilon$-dimensional label. Let $D^v = \{(x_i^v, y_i^v)\}_{i=1}^{n} \sim P^v$ be the validation set, where $P^v$ is the distribution of the test set, which is not necessarily the same as $P$. Let $f_\phi$ be a regression model, and $L$ the loss function that returns a performance score comparing $f_\phi(x_i)$ with the true label $y_i$ using Mean Square Error (MSE) loss. We assume a list $N$ of possible nearest neighbors (NNs) based on data distance that can be mixed with an example. For instance, $N$ could contain the options {0 NN, 4 NNs, 8 NNs}. Notice that the “0 NN” option means that no mixing occurs because it is not beneficial to model performance. The more fine-grained the kNN options are, the more precisely
MixR can determine the optimal number of NNs to mix per example. On the other hand, there are diminishing returns because the optimization itself may become more difficult (see Sec. 4.3).

MixR’s framework in Fig. 3 is similar to AutoAugment and neural architecture search (NAS) [Zoph and Le, 2017] where a controller is trained to generate kNN mixing policies for all the examples in D. The policies are then sampled probabilistically following the distributions of the softmax layers and used to augment D and train fθ. The models are then evaluated on D’, and the losses are used to update the controller. Alg. 1 shows the pseudocode of MixR. Unlike typical RL settings where an episode consists of a sequence of dependent steps, in our setting an episode consists of independent steps because the controller can reach a goal after taking only one action (i.e., sampling kNN mixing policies). As a result, we can sample policies and compute model losses in parallel (lines 6–10). The collected losses are then used to compute the rewards and update the controller (lines 11–16). More details on the RL and controller are covered in the following sections.

3.2 Policy Optimization RL

Finding the optimal policy involves taking the gradient of the objective function J(θ) = Eπθ[R] where π is the policy, θ is the parameters of the controller, and R is the reward function. We would like to minimize the regression model loss on a validation set when mixing a batch of examples. However, the validation loss is computed using the regression model, which does not involve θ. Hence we cannot analytically compute the differential of the reward function with respect to θ. In addition, there is no training data on how mixing each example affects the model performance. We thus use Proximal Policy Optimization (PPO) [Schulman et al., 2017], which is one of the state-of-the-art on-policy algorithms and is also used in NAS. We use PPO for convenience, and any actor-critic method like A3C [Mnih et al., 2016] can be used as well. In order to minimize the loss, we use the inverse of loss (i.e., 1/loss) as the reward function. For stable learning, we employ an exponential moving average of previous rewards as the baseline function [Sutton and Barto, 2018], which is known to reduce variance.

3.3 Controller Design

The controllers in the previous works AutoAugment [Cubuk et al., 2019] and NAS [Zoph and Le, 2017] use recursive neural networks (RNNs) because of the sequential nature of the predictions to make. In AutoAugment, a policy consists of a sequence of operations on images that should be
We provide experimental results for MixR. We evaluate the regression models trained on the augmented training sets on separate test sets and repeat all model trainings five times. We use PyTorch [Paszke et al., 2017], and all experiments are performed using Intel Xeon Silver CPUs and NVidia RTX GPUs. More experimental setting details are in Sec. A.1.

Measures We use two accuracy metrics. RMSE = \(\sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}\) is the absolute difference between predicted labels and true labels where a lower value is better. \(R^2\) (Coefficient of determination) = \(1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}\) is the relative improvement compared to when returning the label average as a prediction. The value is in the range \([0, 1]\) where a higher value is better. The two measures complement each other. For the data distance measure, we use Euclidean distance.

Datasets We use three real and one synthetic datasets. The NO2 emissions dataset [Aldrin, 2004] contains traffic and meteorological information around roads. The features include cars per hour, wind speed, temperature, and others. The label is the NO2 concentration. The Product dataset obtained through a collaboration with a company contains spectrum data that is generated using the procedure described in Sec. 1 and illustrated in Fig. 4 on 20-layer 3-d semiconductors. The Product dataset is proprietary, so for clarity we also experiment on a public synthetic dataset (Synthetic) provided by the DACON challenge [DACON Co., Ltd. 2020] where a simulator generates spectrum data using the same procedure as Product assuming 4-layer 3-d semiconductors. Finally, the Bike UCI dataset [Dua and Graff, 2017] contains climate and temporal information to predict the number of bikes in demand where the features include temperature, humidity, rainfall, and season. Table I compares the four datasets in more detail. NO2 and Product have lowest and highest dimensions, respectively, and Synthetic has the largest size. We note that the data sizes are intentionally small as it is critical to show that MixR performs well for small datasets. There are no personal identifiers.

MixR Settings As a default, we evaluate MixR using the MLP controller. In Sec. 4.4, we make a comparison with MixR using the RNN controller. In Alg. 1 we set the time horizon \(T\) to 20 and run the controller training time on various datasets.

We thus propose a more light-weight controller based on a multi-layer perceptron (MLP) instead of an RNN (Fig. 3, controller on the right) where the goal is to predict the best kNN mixing policies of all examples at once. Here the input does not play a role, and we fix it to a vector of 1’s. Also, there is one output node with a softmax layer per example. In Sec. 4.4 we show that using an MLP controller has similar model performances as when using an RNN controller, but is faster in terms of controller training time on various datasets.

4 Experiments

We provide experimental results for MixR. We evaluate the regression models trained on the augmented training sets on separate test sets and repeat all model trainings five times. We use PyTorch [Paszke et al., 2017], and all experiments are performed using Intel Xeon Silver CPUs and NVidia RTX GPUs. More experimental setting details are in Sec. A.1.

MixR Settings As a default, we evaluate MixR using the MLP controller. In Sec. 4.4, we make a comparison with MixR using the RNN controller. In Alg. 1 we set the time horizon \(T\) to 20 and run
Predict layer thicknesses

Figure 4: Spectrum generation on a 3-d semiconductor for the Product dataset.

Table 1: Settings for the four datasets.

| Dataset  | Data dimension | Label dimension | Training set size | Validation set size |
|----------|----------------|-----------------|-------------------|---------------------|
| NO2      | 7              | 1               | 200               | 100                 |
| Product  | 580            | 20              | 300               | 200                 |
| Synthetic| 226            | 4               | 600               | 300                 |
| Bike     | 12             | 1               | 300               | 200                 |

We empirically show that the linearity of a regression model only holds within certain data distances using the following simple experiment. Given a dataset, we first train a regression model. We then generate mixed examples and compute the RMSE between their interpolated labels and the labels predicted by the regression model. While the regression model is not perfect, the relative trend of the error gives sufficient insight. Fig. 5a indeed shows increasing label errors for larger data distances for the three real datasets. We also train regression models on top of augmented training sets where we only mix examples within certain distance ranges. As a result, mixing examples with larger distances has increasingly-negative effects on the model performances as we show in Fig. 2.

4.1 Limited Linearity in Regression

We compare MixR with the four baselines using the settings in Table 1. Table 2 shows the results for the three real datasets (the Synthetic dataset results are similar and shown in Sec. A.2). MixR consistently and significantly outperforms all the baselines in terms of regression model performance, which demonstrates that it is able to pinpoint which examples should be mixed with how many kNNs. Figs. 5b-5d show which kNN options are frequently used by MixR on the three real datasets (the kNN options for Synthetic are in Sec. A.2). Although MixR is slower than the baselines due to the RL, the gained performance can be worth the effort in real applications because data augmentation is a one-time cost that can run in batch mode. Another observation is that Original Mixup performs relatively better for Product than other datasets because linear interpolations work for longer data distances as evidenced by the large k = 223 value of Global kNN. In general, MixR outperforms Original Mixup by a larger margin the less effective the linear interpolations.

Baselines

We employ four baselines. First, we train a regression model on the labeled data without any data augmentation (“No Augmentation”). Next, we faithfully implement the original Mixup algorithm [Zhang et al., 2018] where all example pairs can be mixed (“Original Mixup”). We also implement Manifold Mixup [Verma et al., 2019], which is a state-of-the-art Mixup technique for classification that is also applicable to regression. Finally, we compare with a simplified version of MixR where we use a single k NN value to mix all examples with their nearest neighbors (“Global kNN”). This k parameter is tuned using Bayesian Optimization [Mockus, 1974].
We compare MixR using the default MLP controller with MixR that uses the RNN controller in Table 4 using the NO2, Product, and Bike datasets. We investigate how changing the kNN options affects MixR’s performance in Table 3 using the NO2 dataset results are similar and shown in Sec. A.3. For each dataset, we compare its default series with other possible series. We also fix the entire runtime by using the same number of epochs, so we only need to compare the model performances. We observe that providing more kNN options improves MixR’s performance, but only to a certain extent. Adding options unnecessarily may actually result in lower performances as the best options are more difficult to find within the same time. Hence, MixR performs best with a modest number of options.

4.3 K Nearest Neighbor Options

We investigate how changing the kNN options \( N \) affects MixR’s performance in Table 3 using the NO2 and Product datasets. The other dataset results are similar and shown in Sec. A.3. For each dataset, we compare its default series with other possible series. We also fix the entire runtime by using the same number of epochs, so we only need to compare the model performances. We observe that providing more kNN options improves MixR’s performance, but only to a certain extent. Adding options unnecessarily may actually result in lower performances as the best options are more difficult to find within the same time. Hence, MixR performs best with a modest number of options.

4.4 Comparison with RNN Controller

We compare MixR using the default MLP controller with MixR that uses the RNN controller in Table 4 using the NO2 and Product datasets (the Bike dataset results are similar and shown in Sec. A.4). To clearly observe the RNN controller’s performance depending on the example ordering in the RNN, we consider two scenarios where (1) the examples are ordered after dimensionality reduction to one dimension using PCA [F.R.S., 1901] and (2) the examples are ordered randomly. As
Figure 6: (a) Model training convergence comparison on the Product dataset. (b) MixR runtime against the training set size on the Synthetic dataset. (c) Rewards when using the MLP and RNN controllers. (d) MixR performances for different $\lambda$ values on the NO2 dataset.

Table 3: MixR performances when using different kNN options on the NO2 and Product datasets.

| Dataset | kNN Options                      | RMSE     | $R^2$    |
|---------|----------------------------------|----------|----------|
| NO2     | $\{0\} \cup \{4^i|i \in [0, 3]\}$ | 0.5357 $\pm 0.0020$ | 0.5431 $\pm 0.0034$ |
|         | $\{0\} \cup \{2^i|i \in [0, 7]\}$ (Default) | 0.5248 $\pm 0.0015$ | 0.5615 $\pm 0.0025$ |
|         | $\{0\} \cup \{10^i|i \in [1, 19]\}$ | 0.5317 $\pm 0.0011$ | 0.5498 $\pm 0.0019$ |
| Product | $\{0\} \cup \{8^i|i \in [0, 2]\}$ | 1.2751 $\pm 0.0041$ | 0.7456 $\pm 0.0018$ |
|         | $\{0\} \cup \{4^i|i \in [0, 4]\}$ (Default) | 1.2345 $\pm 0.0015$ | 0.7691 $\pm 0.0022$ |
|         | $\{0\} \cup \{2^i|i \in [0, 8]\}$ | 1.2366 $\pm 0.0006$ | 0.7685 $\pm 0.0016$ |

As a result, there is no significant difference in model performances among the three approaches, which suggests that any influence among examples when mixing with neighbors is either minor or difficult to capture. As discussed in Sec. 3.3, we suspect that there is no easy way to model the influence by ordering the examples at least for our datasets. Fig. 6c shows that the reward trends for the two controllers are similar as well. However, using the MLP controller instead of the RNN controller improves the entire runtime by 7–33%.

4.5 Justification for Setting $\lambda = 0.5$

We show how setting $\lambda = 0.5$ empirically results in the best MixR performance. Fig. 6d shows MixR’s performance when we vary $\lambda$ using the same setting in Table 1 for the NO2 dataset. As a result, the performance peaks when $\lambda = 0.5$. The results for the other real datasets are similar and shown in Sec. A.5. We suspect that generating examples in the sparsest areas helps the regularization the most. In a classification setting, however, setting $\lambda = 0.5$ is not ideal [Zhang et al., 2018] because the generated examples may interfere with decision boundaries between mixing examples.

4.6 Improving Other Mixup Techniques through Integration

MixR can also be integrated with other Mixup techniques to improve their performances, and we demonstrate by combining it with Manifold Mixup [Verma et al., 2019]. Here examples are mixed in multiple layers of the MLP with their NNs (see Sec. A.6 for more details). Table 5 shows that the integrated method significantly improves the Manifold Mixup performance on all the real datasets.

5 Related Work

There are largely two branches of work for data augmentation in regression. One is semi-supervised regression [Kostopoulos et al., 2018; Zhou and Li, 2005; Kang et al., 2016; Kim et al., 2020] where the goal is to utilize unlabeled data for training. In comparison, MixR does not assume the availability of unlabeled data. Another branch is data augmentation when there is no unlabeled data, which is our research focus. To the best of our knowledge, there is no technique tailored to regression, so we instead cover data augmentation techniques for classification that can be extended to regression.

We explore data augmentation techniques for classification and investigate if they can be used in a regression setting. There are largely three approaches: generative models, policies, and Mixup techniques. Generative models including GANs [Goodfellow et al., 2014] and VAEs [Kingma and Welling, 2014] are popular in classification where the idea is to generate realistic data that cannot be
Table 4: MixR performances when using an MLP or RNN controller. When using an RNN controller, we either (1) sort the examples using dimensionality reduction or (2) order them randomly.

| Dataset | Method                   | RMSE    | $R^2$     | Runtime (mins) |
|---------|--------------------------|---------|----------|----------------|
| NO2     | MixR (MLP)               | 0.5248±0.0015 | 0.5615±0.0025 | 50             |
|         | MixR (RNN, dim. reduction) | 0.5277±0.0031 | 0.5638±0.0024 | 75             |
|         | MixR (RNN, rand. selection) | 0.5279±0.0004 | 0.5593±0.0016 | 75             |
| Product | MixR (MLP)               | 1.2345±0.0015 | 0.7691±0.0022 | 195            |
|         | MixR (RNN, dim. reduction) | 1.2239±0.0027 | 0.7662±0.0011 | 211            |
|         | MixR (RNN, rand. selection) | 1.2382±0.00178 | 0.7697±0.0031 | 210            |

Table 5: Manifold mixup [Verma et al., 2019] performance changes when integrated with MixR.

| Dataset | RMSE Change (Improvement %) | $R^2$ Change (Improvement %) |
|---------|-----------------------------|------------------------------|
| NO2     | −0.0130 (2.4%)               | +0.0216 (4.0%)               |
| Product | −0.0251 (1.9%)               | +0.0074 (1.0%)               |
| Bike    | −23.34 (5.9%)                | +0.0432 (7.0%)               |

distinguished from the real data by a discriminator. However, a major assumption is that the labels of the generated examples are the same, which does not necessarily hold in a regression setting where most examples may have different labels. Another approach is to use policies [Cubuk et al., 2019], which specify fixing rules for transforming the data while maintaining the label value. For example, image processing policies may flip, rotate, or adjust the brightness of images to generate new valid images. However, in a regression setting, the same transformed examples may now have different unknown labels, making them unsuitable for training. The recently-proposed Mixup [Zhang et al., 2018, Berthelot et al., 2019, Yun et al., 2019, Berthelot et al., 2020, Hendrycks et al., 2020, Verma et al., 2019] takes the alternative approach of generating both data and labels together by mixing existing examples with different labels assuming linearity between training examples [Chapelle et al., 2000, Wu et al., 2020]. Although the Mixup paper mentions that its techniques can easily be extended to regression, we make the non-trivial observation that the linearity of a regression model is limited where mixing examples with far-away examples may not be beneficial and even detrimental to model performance. Instead, MixR shows good performance by carefully choosing which nearest neighbors to mix with, a technique that only works in a regression setting.

Within reinforcement learning [Sutton and Barto, 2018, Kaelbling et al., 1996], we are interested in policy searching methods [Neumann and Peters, 2015] where our goal is to find the best policy for mixing examples. In particular, MixR’s framework is inspired by the recent AutoAugment framework [Cubuk et al., 2019, Lim et al., 2019, Zhang et al., 2020], which uses PPO [Schulman et al., 2017] to search for data augmentation policies that dictate how to modify existing examples to generate additional training data for image classification. In comparison, we solve the completely different problem of data augmentation for regression where we extend Mixup and find the best kNN neighbors to mix for each example.

6 Conclusion

We proposed MixR, which to our knowledge is the first Mixup-based data augmentation framework tailored to regression tasks. We observe that linearity of a regression model only holds up to a certain data distance per example. Hence, MixR extends Mixup by determining the k nearest neighbors to mix per example using reinforcement learning where the objective is to minimize the regression model’s loss on a validation set. Since MixR only limits the examples to mix, it can be integrated with any existing Mixup technique to improve its performance. In our experiments, we showed that MixR outperforms various data augmentation baselines for regression by effectively selecting among a modest number of k nearest neighbor options. MixR demonstrates that data augmentation can significantly improve the model performance for regression tasks. However, if not controlled, data augmentation may also introduce new bias in the data that makes the model more discriminative. Thus an interesting future work is to augment data while ensuring model fairness. MixR also has limitations where it can only mix existing examples instead of creating new ones and has computational overhead.
Acknowledgement

Sung Yoon Ryu, Soo Seok Lee, and Gwang Nae Gil of Samsung Electronics provided the Product dataset and key insights on the 3-d semiconductor application. This work is supported by the Samsung Electronics’ University R&D program [Algorithm advancement and reliability-enhancing technique development for spectrum data-based deep learning] and a Kwon Oh-Hyun Assistant Professorship.

References

Magne Aldrin. Cmu statlib dataset. http://lib.stat.cmu.edu/datasets/, 2004. Accessed May 12th, 2022.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. CoRR, abs/1607.06450, 2016.

David Berthelot, Nicholas Carlini, Ian J. Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. In NeurIPS, pages 5050–5060, 2019.

David Berthelot, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution matching and augmentation anchoring. In ICLR, 2020.

Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir Vapnik. Vicinal risk minimization. In NIPS, pages 416–422, 2000.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In CVPR, pages 113–123, 2019.

DACON Co., Ltd. Challenge on semiconductor thin film thickness analysis. https://dacon.io/competitions/official/235554/data/, 2020. Accessed May 12th, 2022.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml

Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In NIPS, pages 2672–2680, 2014.

Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In ICLR, 2020.

Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In ICML, pages 2731–2741, 2019.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. J. Artif. Int. Res., 4(1):237–285, May 1996.

Pilsung Kang, Dongil Kim, and Sungzoon Cho. Semi-supervised support vector regression based on self-training with label uncertainty: An application to virtual metrology in semiconductor manufacturing. Expert Syst. Appl., 51:85–106, 2016.

Sung Wook Kim, Young Gon Lee, Bayu Adhi Tama, and Seungchul Lee. Reliability-enhanced camera lens module classification using semi-supervised regression method. Applied Sciences, 10(11), 2020.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In ICLR, 2014.

Georgios Kostopoulos, Stamatis Karlos, Sotiris Kotsiantis, and Omiros Ragos. Semi-supervised regression: A recent review. J. Intell. Fuzzy Syst., 35(2):1483–1500, 2018.

Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In NeurIPS, pages 6662–6672, 2019.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In ICML, pages 1928–1937, 2016.
Jonas Mockus. On bayesian methods for seeking the extremum. In Proceedings of the IFIP Technical Conference, page 400–404, Berlin, Heidelberg, 1974. Springer-Verlag. ISBN 3540071652.

Gerhard Neumann and Jan Peters. Policy search: Methods and applications. In ICML, 2015.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In NIPS Autodiff Workshop, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017.

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. A Bradford Book, 2018. ISBN 0262039249.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In ICML, 2019.

Sen Wu, Hongyang R. Zhang, Gregory Valiant, and Christopher Ré. On the generalization effects of linear transformations in data augmentation. In ICML, pages 10410–10420, 2020.

Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In ICCV, pages 6022–6031, 2019.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In ICLR, 2018.

Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. In ICLR, 2020.

Zhi-Hua Zhou and Ming Li. Semi-supervised regression with co-training. In IJCAI, page 908–913, 2005.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In ICLR, 2017.
A Appendix

A.1 More Experimental Settings

We continue describing our experimental settings from Sec. 4. For the regression model, we use layer normalization [Ba et al., 2016] for the Product and Synthetic datasets. For the controller, we employ an exponential moving average of previous rewards with a weight of 0.95 as the baseline function. For better exploration, we add an entropy term into the loss with a weight of 0.01. When using an MLP controller, we use a multi-layer perceptron with 4 hidden layers and set the number of nodes per hidden layer to be [100, 100, 100, 100] for the four datasets. When using an RNN controller, we use an embedding layer to adjust the dimension of inputs and LSTM cells with a hidden state size of 100. We use a learning rate of $\alpha = 0.0002$ for the MLP controller and $\alpha = 0.0005$ for the RNN controller. For the regression model $f$, we use a learning rate of 0.001 for Synthetic and 0.0001 for the other datasets. When training $f$, we use a batch size of 32 for NO2 and 64 for the other datasets.

When constructing the Synthetic dataset, we take a subset of the entire DACON challenge dataset [DACON Co., Ltd. 2020]. While the the range of thicknesses for the full datasets is [10, 300] for all four layers, we only use examples with thicknesses within the range [10, 100]. The purpose is to reduce the training time while still making a clear comparison between MixR and the other methods. When constructing the Bike dataset, we take a random subset of the original dataset [Dua and Graff, 2017].

A.2 Accuracy and Runtime Results for the Synthetic Dataset

We continue from Sec. 4.2 and show the accuracy and runtime results for the Synthetic dataset in Table 6. The results are similar to the Product dataset in Table 2.

Table 6: Accuracy and runtime performances for the Synthetic dataset. The other settings are identical to those of Table 2.

| Dataset       | Method           | RMSE         | $R^2$         | Runtime (mins) |
|---------------|------------------|--------------|---------------|----------------|
| Synthetic     | No Augmentation  | 15.6838 ±0.2790 | 0.7158±0.0092 | 0.37           |
|               | Original Mixup   | 13.8618±0.1263 | 0.7747±0.0039 | 18             |
|               | Manifold Mixup   | 14.0457±0.1067 | 0.7628±0.0036 | 26             |
|               | Global kNN (k=61) | 13.7587±0.0493 | 0.7751±0.0016 | 150            |
|               | MixR             | 13.5384±0.0248 | 0.7842±0.0009 | 330            |

kNN Options Frequency Histogram

We also show which kNN options are frequently used in our default setting for the Synthetic dataset in Fig. 7.

Figure 7: kNN option frequency histogram for the Synthetic dataset.

A.3 K Nearest Neighbor Options for Other Datasets

We continue from Sec. 4.3 and provide the kNN options experiments for the Synthetic and Bike datasets in Table 7. The observations are similar to those of Table 3 where a modest number of kNN options is sufficient for good model performance.
Table 7: Comparing kNN options for MixR on the Synthetic and Bike datasets.

| Dataset  | kNN Options                                        | RMSE       | $R^2$       |
|----------|---------------------------------------------------|------------|-------------|
| Synthetic| {$0\} \cup \{8^i | i \in [0,3]\}$ (Default)                      | 13.7498±0.0449 | 0.7761±0.0013 |
|          | {$0\} \cup \{2^i | i \in [0,9]\}$                                  | 13.5384±0.0248 | 0.7842±0.0009 |
|          | {$0\} \cup \{30^i | i \in [1,19]\}$                                | 13.5510±0.0916 | 0.7839±0.0029 |
| Bike     | {$0\} \cup \{8^i | i \in [0,2]\}$                                   | 378.02±2.6582  | 0.6675±0.0028 |
|          | {$0\} \cup \{4^i | i \in [0,4]\}$ (Default)                              | 368.80±3.0689  | 0.6766±0.0057 |
|          | {$0\} \cup \{2^i | i \in [0,8]\}$                                  | 370.68±0.6302  | 0.6742±0.0015 |

A.4 Comparison with RNN Controller for the Bike Dataset

We continue from Sec. 4.4 and compare the performances of MixR using the default MLP controller with MixR using the RNN controller for the Bike dataset in Table 8. The results are similar to Table 4 where there is no significant difference in model performances among the approaches.

Table 8: MixR performances when using an MLP or RNN controller for the Bike dataset. When using an RNN controller, we either (1) sort the examples using dimensionality reduction or (2) order them randomly.

| Dataset | Method                     | RMSE       | $R^2$       | Runtime (mins) |
|---------|----------------------------|------------|-------------|----------------|
| Bike    | MixR (MLP)                 | 368.86±3.0689 | 0.6766±0.0057 | 107            |
|         | MixR (RNN, dim. reduction) | 368.03±3.5014 | 0.6767±0.0061 | 125            |
|         | MixR (RNN, rand. selection)| 370.22±1.3731 | 0.6759±0.0019 | 123            |

A.5 Justification for Setting $\lambda = 0.5$ for Other Datasets

We continue from Sec. 4.5 and show MixR’s performances when varying $\lambda$ using the same setting in Table 1 for the Product and Bike datasets. Similar to Fig. 6d, the performances peak when $\lambda = 0.5$.

Figure 8: MixR’s performances for different $\lambda$ values on the Product and Bike datasets.

A.6 Integrating MixR with Manifold MixUp

We continue from Sec. 4.2 and provide more details on integrating MixR with Manifold Mixup [Verma et al., 2019]. We first run MixR and determine which examples to mix. Next, instead of mixing examples in the training data before model training, we mix examples on multiple layers in the MLP during model training. For the second mixing step, we use Manifold Mixup as is where we start with a set of eligible layers for mixing. We always set the first three layers as eligible. For each training batch, we randomly select a mixing layer and mix the features and labels of the examples that MixR decides to mix.