Project report: Data valuation using Reinforcement Learning

Chintan Shah, Saurabh Vaidya

December 11, 2020

1 Introduction

Data valuation has high significance in any machine learning task. Obtaining good quality data is primary objective for any machine learning project being undertaken. However, in reality, large volume of high quality data is not easily available. Often times, data is crowd sourced and hence is susceptible to corruption. Evaluating and quantifying value of data has been a topic of research and various developments have been made in that respect. There have been efforts to evaluate value of entire dataset. However, evaluating single datum give more valuable insights. Our project draws work from one such research done by Yoon et al [11] which uses reinforcement learning.

Data valuation can be useful in intrinsically identifying and differentiating between bad and good quality data. Such robust learning can allow us to achieve higher performance compared to a model trained on entire data. This also allows us to get weighted contribution of each datum to the target performance. Data valuation has huge scope in improving machine learning performance as data is at the core of all learning techniques.

Our choice of this project was also motivated by our preference to look out for some unconventional application of reinforcement learning. This project also blends aspects of meta learning which we were also interested in.

2 Objectives and use-cases

There are several objectives and use-cases that estimating the data-value enables. Instead of treating each data point equally, we can lower the priority for samples that do not help us in our downstream task. This can assist with several use cases

Data Valuation Knowing the value of each data point can help being robust to issues in data such as incorrect labels and data points coming from different distributions. It can also learn better features by quantifying data points with usefulness to the target task in mind.

Corrupted Sample Discovery Seldom is a dataset perfectly clean and usable in practical machine learning projects. If the corrupted samples can be discovered, they can be down-weighted to improve the robustness of the model learned.

Augmenting Datasets with Cheap Sources There are several tasks for which we do not have a large dataset and the usual method of building models by transfer learning [7] may not be available. In this case, it may be useful to augment our small datasets with samples from cheap sources such as Google Image search and not worry about the data quality affecting our model.

Domain Adaptation Domain adaptation is a very challenging problem and a lot of effort has been spent by the machine learning community on the problem [2][8]. The scenario is such that the datasets share the same label space yet the samples come from very different distributions. Quantifying the quality of data sets can help in the scenario such that we use samples that are most useful for the target distributions.

3 Problem Formulation

We aim to develop a method such that when given input data, can output values for each data point such that a predictor trained on these selected high value data points can lead to improved performance. There are several ways to frame this problem and we explain both of them in the following sections - Sec 3.1 and Sec 3.2.

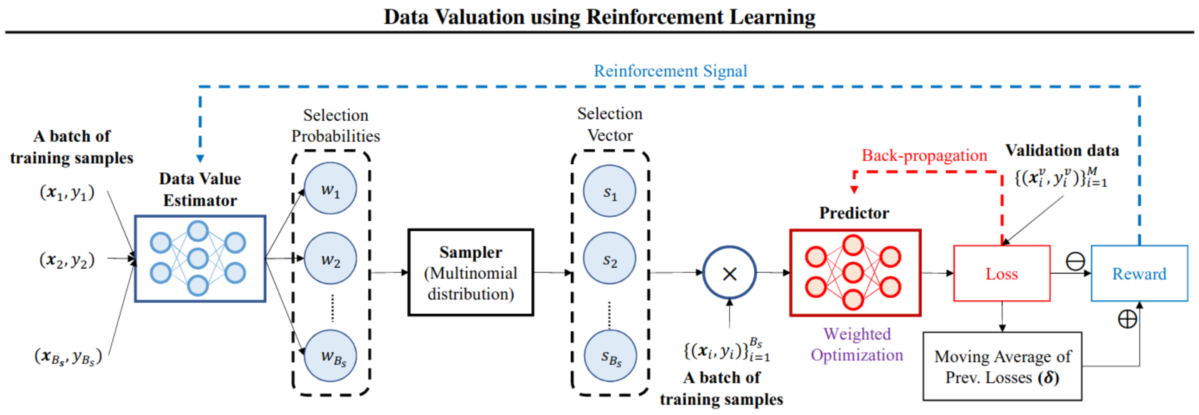


Figure 1: This figure captures the framework for Data Valuation using Reinforcement Learning. The idea is to train a Data Value estimator and a Predictor in an end-to-end fashion using Reinforcement Learning and is further explained in section 3.1

3.1 Data Valuation using Reinforcement Learning

To implement Data valuation using Reinforcement learning (DVRL), we need a set of training and validation data. Training data is denoted as $\mathcal{D} = (x_i, y_i)_{i=1}^{N} \sim \mathcal{P}$. The validation dataset is denoted as $\mathcal{D}^v = (x_i, y_i)_{i=1}^{N_v} \sim \mathcal{P}^t$.

DVRL consists of two function approximators: the predictor and data value estimator(DVE). We denote the prediction as f_θ and value estimator as h_ϕ where θ and ϕ are the parameters. The predictor $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ is trained to minimize a loss $\mathcal{L}_f$ on training data $\mathcal{D}$ (e.g MSE loss for regression or Cross Entropy loss for classification respectively). The DVE model $h_\phi : \mathcal{X} \rightarrow [0, 1]$ is optimized to output weights which become selection probabilities for data i.e denoting quality of data. We denote output of DVE as selection probability of data point, w_i and define a selection vector $\mathbf{s} : s_i = 1/0$ denoting whether the data point is selected/not selected for training the predictor f_θ.

Since we have two different models trying to achieve different objective, we define two different loss objectives that both models would optimize. The predictor model f_θ optimizes a weighted loss $\mathcal{L}_f$ which can be anything like cross-entropy or mean square error. The weights are selection probabilities output by DVE model. Thus the predictor is trained such that it minimizes this objective.

$$f_\theta = \underset{\hat{f} \in F}{\arg\min}\, \mathbb{E}_{(x,y)\sim\mathcal{P}}\left[h_\phi(x, y)\mathcal{L}_f(\hat{f}_\theta(x), y)\right] \tag{1}$$

The DVE model tries to minimize a estimator loss $\mathcal{L}_h$ between the predictor's performance on a held out validation set.

$$h_\phi = \min \mathbb{E}_{(x^v, y^v)\sim\mathcal{P}^t}\left[\mathcal{L}_h(f_\theta(x^v), y^v)\right] \tag{2}$$

Both $\mathcal{L}_h$ and $\mathcal{L}_f$ can be the same or different.

3.2 Data Valuation using Gumbel-Softmax

In the previous section, we developed a framework that developed a way to estimate data values given a training data set $\mathcal{P}$ and a validation set $\mathcal{P}^t$ and train the entire framework end-to-end using reinforcement learning based methods. We needed to use the policy gradient algorithm REINFORCE [9] since the sampling operator is non-differentiable and as such it is not possible to backpropagate through the stochastic sampling node. We can instead use a reparameterization trick in order to make the sampling operation through a discrete distribution differentiable and to enable the process of back-propagation through the sampling operator.

Gumbel-softmax The Gumbel-softmax[1] trick [1] [6] is a way to make the reparameterization trick work with discrete data. In this trick we have a random variable G where $G = -\log(-\log(U))$ and $U \sim \text{Unif}(0, 1)$. We observe that we can reparameterize any discrete variable in terms of G by observing that we can represent any discrete variable X with $P(X = k)$ given by α_k and let $\{G_k\}_{k \leq K}$ be an i.i.d sequence of standard Gumbel distributions, then

$$X = \arg\max_k(\log \alpha_k + G_k) \tag{3}$$

However, this does not yet solve the issue, the $\arg\max$ operator used here is not differentiable due to being discontinuous. To start solving the problem, we realize that essentially, the idea is that we can encode any discrete random variable X as a one-hot encoded vector $\in [0, 1]^K$ with exactly one component being 1 and others 0. Now that we understand that, we can relax the discontinuity by realizing that the one-hot vector lies in the $K - 1$ dimensional probability simplex and in particular - on the corners of the simplex! We can relax this discontinuity by allowing X to take values in the probability simplex. [1] and [6] propose using a mapping function $f_\tau(x)_k$

$$f_\tau(x)_k = \frac{\exp(x_k/\tau)}{\sum_k \exp(x_k/\tau)} \tag{4}$$

Armed with this trick, we can redefine our random variable X as X^τ

$$X^\tau = (X_k^\tau)_k = f_\tau(\log \alpha + G) = \left(\frac{\exp\left((\log \alpha_k + G_k)/\tau\right)}{\sum_{i=1}^K \exp\left((\log \alpha_i + G_i)/\tau\right)} \right)_k \tag{5}$$

This allows us to sample from a discrete distribution, reparameterize in terms of the standard gumbel distribution G. Eq 5 has an interesting property due to the presence of the hyperparameter tau - the idea is that as $\tau \to 0$, X^τ approaches a one-hot encoded vector and as $\tau \to \infty$, X^τ approaches a uniform distribution. Finally, we use the gumbel-softmax trick in learning data values in an end-to-end fashion by mininmizing the following loss function where we define the advantage function $A = v_\theta(x^v, y^v) - \delta$ and $v_\theta(x^v, y^v)$ is the validation accuracy.

$$\mathcal{L}_g = \mathbb{E}_{(\mathbf{x}^v, y^v) \sim P^t}\left[\mathbb{E}_{\mathbf{s} \sim \pi_\phi(\mathcal{D}, \cdot)}\left[\mathbf{s}A\right]\right] \tag{6}$$

4 Proposed methods

4.1 DVRL

The problem of Reinforcement learning is formulated with state, action and reward. The input data to the DVE becomes the state, the DVE model is the policy approximation, and action of the agent is selecting the data points based on selection probabilities output by DVE. And reward is the performance of the predictor on the validation data.

The RL problem is solved by using policy gradients with REINFORCE [9] algorithm. The objective is for us to learn an optimal policy which chooses high valued samples such that training the predictor on those chosen samples can improve it's performance. In policy gradients we usually approximate the policy as a function with some parameters ϕ. The update to our policy corresponds to updating these parameters using some performance measure $J(\phi)$. We use gradient of this performance measure and update the parameters which would maximize the performance.i.e we do gradient ascent

$$\phi = \phi + \alpha \nabla J(\phi) \tag{7}$$

[1]A useful resource to understand the technique is https://casmls.github.io/general/2017/02/01/GumbelSoftmax.html

The performance measure in our case is the DVE loss from equation 2. We now derive the update rule for parameters of our policy i.e DVE

$$\begin{aligned}
\hat{l}(\phi) &= \mathbb{E}_{(\mathbf{x}^v, y^v)\sim P^t}\left[\mathbb{E}_{\mathbf{s}\sim\pi_\phi(\mathcal{D},\cdot)}\left[\mathcal{L}_h\left(f_\theta\left(\mathbf{x}^v\right), y^v\right)\right]\right] \\
&= \int P^t\left(\mathbf{x}^v\right)\left[\sum_{\mathbf{s}\in[0,1]^N}\pi_\phi(\mathcal{D},\mathbf{s})\cdot\left[\mathcal{L}_h\left(f_\theta\left(\mathbf{x}^v\right), y^v\right)\right]\right]d\mathbf{x}^v
\end{aligned}$$

we directly compute the gradient $\nabla_\phi\hat{l}(\phi)$ as:

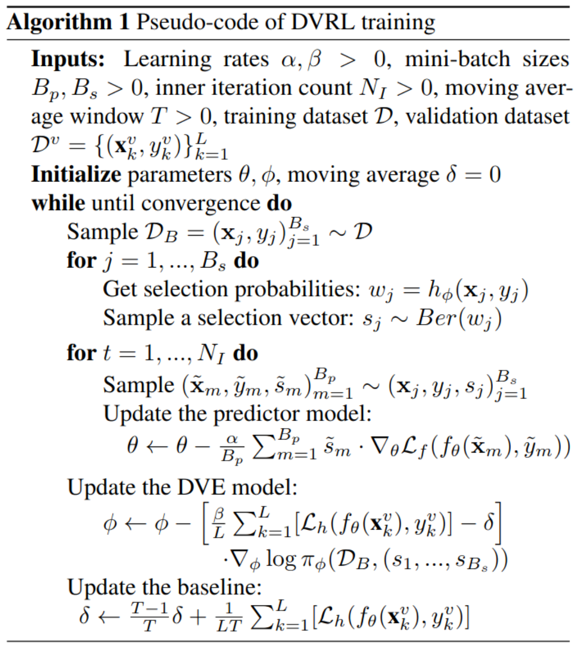
$$\begin{aligned}
\nabla_\phi\hat{l}(\phi) &= \int P^t\left(\mathbf{x}^v\right)\left[\sum_{\mathbf{s}\in[0,1]^N}\nabla_\phi\pi_\phi(\mathcal{D},\mathbf{s})\cdot\left[\mathcal{L}_h\left(f_\theta\left(\mathbf{x}^v\right), y^v\right)\right]\right]d\mathbf{x}^v \\
&= \int P^t\left(\mathbf{x}^v\right)\left[\sum_{\mathbf{s}\in[0,1]^N}\nabla_\phi\log\left(\pi_\phi(\mathcal{D},\mathbf{s})\right)\cdot\pi_\phi(\mathcal{D},\mathbf{s})\cdot\left[\mathcal{L}_h\left(f_\theta\left(\mathbf{x}^v\right), y^v\right)\right]\right]d\mathbf{x}^v \\
&= \mathbb{E}_{(\mathbf{x}^v, y^v)\sim P^t}\left[\mathbb{E}_{\mathbf{s}\sim\pi_\phi(\mathcal{D},\cdot)}\left[\mathcal{L}_h\left(f_\theta\left(\mathbf{x}^v\right), y^v\right)\right]\nabla_\phi\log\left(\pi_\phi(\mathcal{D},\mathbf{s})\right)\right]
\end{aligned}$$

where $\nabla_\phi\log\left(\pi_\phi(\mathcal{D},\mathbf{s})\right)$ is

$$\nabla_\phi\log\left(\pi_\phi(\mathcal{D},\mathbf{s})\right) = \nabla_\phi\sum_{i=1}^{N}\log\left[h_\phi\left(\mathbf{x}_i, y_i\right)^{s_i}\cdot\left(1-h_\phi\left(\mathbf{x}_i, y_i\right)\right)^{1-s_i}\right]$$

$$= \sum_{i=1}^{N}s_i\nabla_\phi\log\left[h_\phi\left(\mathbf{x}_i, y_i\right)\right] + \left(1-s_i\right)\nabla_\phi\log\left[\left(1-h_\phi\left(\mathbf{x}_i, y_i\right)\right)\right]$$

Now that we have our policy gradient update rule, we can combine all the components of training a DVRL into a pseudo code.



```
Algorithm 1 Pseudo-code of DVRL training
  Inputs: Learning rates $\alpha, \beta > 0$, mini-batch sizes $B_p, B_s > 0$, inner iteration count $N_I > 0$, moving average window $T > 0$, training dataset $\mathcal{D}$, validation dataset $\mathcal{D}^v = \{(\mathbf{x}_k^v, y_k^v)\}_{k=1}^L$
  Initialize parameters $\theta, \phi$, moving average $\delta = 0$
  while until convergence do
      Sample $\mathcal{D}_B = (\mathbf{x}_j, y_j)_{j=1}^{B_s} \sim \mathcal{D}$
      for $j = 1, ..., B_s$ do
          Get selection probabilities: $w_j = h_\phi(\mathbf{x}_j, y_j)$
          Sample a selection vector: $s_j \sim Ber(w_j)$
      for $t = 1, ..., N_I$ do
          Sample $(\tilde{\mathbf{x}}_m, \tilde{y}_m, \tilde{s}_m)_{m=1}^{B_p} \sim (\mathbf{x}_j, y_j, s_j)_{j=1}^{B_s}$
          Update the predictor model:
              $\theta \leftarrow \theta - \frac{\alpha}{B_p}\sum_{m=1}^{B_p}\tilde{s}_m \cdot \nabla_\theta\mathcal{L}_f(f_\theta(\tilde{\mathbf{x}}_m), \tilde{y}_m))$
      Update the DVE model:
          $\phi \leftarrow \phi - \left[\frac{\beta}{L}\sum_{k=1}^{L}[\mathcal{L}_h(f_\theta(\mathbf{x}_k^v), y_k^v)] - \delta\right]$
                      $\cdot\nabla_\phi \log \pi_\phi(\mathcal{D}_B, (s_1, ..., s_{B_s}))$
      Update the baseline:
       $\delta \leftarrow \frac{T-1}{T}\delta + \frac{1}{LT}\sum_{k=1}^{L}[\mathcal{L}_h(f_\theta(\mathbf{x}_k^v), y_k^v)]$
```

Figure 2: Pseudo code as presented in the paper

As seen in pseudo code, policy gradient with a baseline is used. The baseline δ can be anything that helps to stabilize learning.

The end product of the training is prediction model and a data value estimator model. For inference we pass our data through the DVE model and get data values as output. We use these output values to conduct experiments to verify the quality of these data values.

5 Subtle Implementation Details and Observations

We implemented the DVRL framework as specified in [11] and had to work through numerous roadblocks that we encountered in the way. The paper describes a general framework describing how to train a data value estimator in an end-to-end fashion without significant increase in computational complexity. While the framework is very promising, there are several subtleties and tricks used in the official implementation that they do not talk about in their paper.

Convergence and stabilization tricks In the paper and the pseudo-code mentioned above 2, the authors mention using mean $\mathcal{L}_h$ on the validation set as the reward signal. In order to reduce variance of the policy gradient estimator, a known trick is to use the REINFORCE algorithm with a baseline and the authors use δ defining it as a sliding-window moving average of the $\mathcal{L}_h$ values in the previous iterations. When we implement the pseudo-code, we observe that the framework fails to converge. On further inspection of their code, we realized that they have not used a sliding-window based approach instead they set the reward signal to the validation accuracy and used a fixed baseline $\delta = v_{\text{acc}}^{\mathbf{w}}$ where $\mathbf{w}$ is learned from a separate neural network trained on the training set before DVRL training begins. We observe that the training process is highly sensitive to the choice (and the presence) of the specific measure of the reward function.

Marginal Information An interesting subtlety that turned out to be crucial to stabilization is a factor the authors call marginal information. They define marginal information as $I_j = |f_\eta(\mathbf{x}_j) - y_j|$ where f_η is a model trained on the validation dataset before commencing training of DVRL and x_j, y_j are samples from the training dataset. I_j is then the absolute difference between the logits as output of f_η and y_j. The authors suggested using the measure in order to learn a better data value estimator and we observe that the presence of marginal information is crucial in stabilizing the framework and as such is not optional.

Exploration Bonus One of the issues with the DVRL framework is that the model does not necessarily learn good data value estimates and quickly saturates to a value of 0 or 1 for all data points in the training process. We observe this phenomenon in almost all our experiments. The code open-sourced by the authors[2] revealed an interesting solution - the authors use an augmented loss function $\mathbf{L} = \mathcal{L}_h + E$ where $E = \max(M - 0.9, 0) + \max(0.1 - M, 0)$ where $M = \frac{1}{B_s}\sum_{j=1}^{B_s} \mathcal{L}_f(f_\theta(x_j), y_j)$. This has an interesting property in that the loss penalizes $M < 0.1$ or $M > 0.9$ that is very high or low values.

Architectural Details The particular choice of the architecture for the Data Value Estimator turned out to be (unexpectedly) crucial in stabilizing the learning framework. As such, the initial choice of using a pre-trained ResNet-18 [11] in order to reduce computational power - did not work. We had to initialize the architecture with random weights in order to stabilize the learning process. The initial thought process of DVRL being a general, model-agnostic framework did not hold true in our preliminary experiments. The framework required significant tuning of the number of layers and particular choice of encoding the input and labels. As such, we don't have a good understanding why this might be the case.

Dataset size used during experiments We noticed in the source code provided by authors that they don't use the entire dataset of image datasets like CIFAR10 [3] and FashionMNIST [10]. We thus emulated their setup to run our experiments on small batch of 4000 training images for CIFAR10. However, we found such size to be very small to achieve any learning. We were only able to see some improvement in dve loss after using 10000 images of CIFAR10 dataset.

[2] https://github.com/google-research/google-research/blob/master/dvrl/

6 Experiments

The source code for our project can be found here `https://github.com/chnsh/dvrl`

6.1 Experiment Setup

We want to verify whether DVRL leads to qualitative data values. Authors in the paper have designed one experiment that can help prove efficacy of data values obtained by DVRL for training predictor. For all the experiments we conduct, we choose CIFAR10 [3] and FashionMNIST [10] data sets.

We initially ran our experiments with MNIST [4] however we found out that MNIST is too simple of a data set to see any significant effect of data values.

On the given datasets, we first want to verify the statement from the paper that *"Removing low value samples from the training dataset can improve the predictor model performance, especially in the cases where the training dataset contains corrupted samples. On the other hand, removing high value samples, especially in the case of small training datasets, would decrease the performance significantly. Overall, the performance after removing high/low value samples is a strong indicator for the quality of data valuation"*. We thus remove top 10 - 50 % of high and low value samples and at each removal we train a new predictor just on the remaining data. We then plot the accuracy of the predictor trained on this data and plot accuracy versus fraction of samples removed.

In order to incorporate some form of corruption, we flip 20% of the labels for our training data at uniform random probability. Our goal is for the DVRL to be able to learn to differentiate between the clean and the corrupted samples through it's data values. To see how the state of our models is during the training phase, we also plot the mean data value of a training batch for corrupted and clean samples.

Further, the paper also reports overall prediction accuracy of predictor when used along with DVE in a DVRL setting to show robust learning. We recreate this experiment where we report accuracy of predictors trained on entire training data but with a DVE model.

Lastly, for image datasets, the authors have used Inception-v3 pretrained model on ImageNet with classifier layer removed, as encoding function. Thus all images are encoded using the output of the pretrained model. We have chosen our encoder as Resnet18 architecture model without any pretrained weights as we saw performance dips when using pretrained models.

6.2 Model Details

Baseline Models We experiment with a ResNet-18 encoder for CIFAR-10 and a simple convolutional network inspired by LeNet [5] for FashionMNIST to evaluate its performance in presence of label noise.

DVRL In this setup, we introduce a Data Value Estimator that shares the the same architecture as the baseline model with the difference that it takes a concatenated $\mathcal{X}||\mathcal{Y}$ as input and outputs a single value w_j as output where w_j is the value of the data point between 0 and 1. This framework is then trained in an end-to-end fashion by wrapping the Baseline Models.

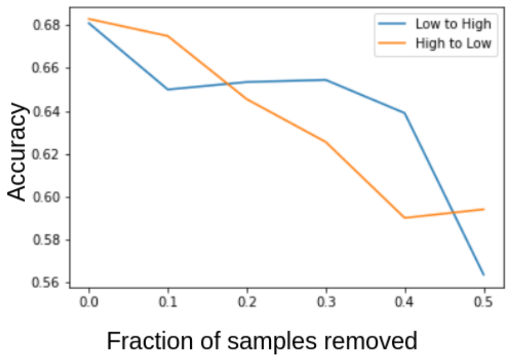
Gumbel-Softmax In this setup, we repeat the same experiment as with DVRL except that we replace the reinforcement learning based-loss to a gumbel-softmax trick and now learn a one-hot vector $\in [0,1]^2$ where the second column determines the selection vector instead of the Bernoulli sampling process.

6.3 Results

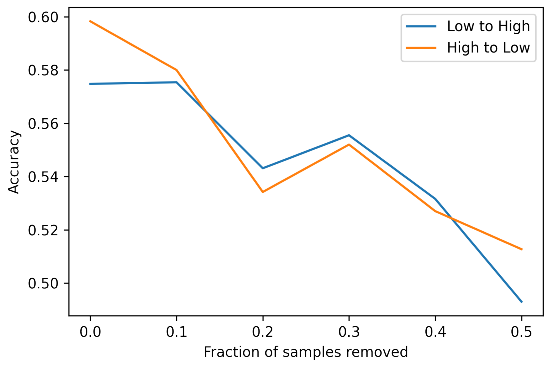
We report the accuracy of the predictor trained on entire training data with and without DVRL and with Gumbel and compare their performances in table 1

Data	Baseline	DVRL	Gumbel-Softmax
Noisy CIFAR	0.5463	0.6526	0.6356
Noisy Fashion MNIST	0.8229	0.8641	0.8645

Table 1: Accuracy on validation set of a predictor - compared with the predictor trained in isolation, within the DVRL framework and within the Gumbel-Softmax framework



(a) Data values by DVRL



(b) Data values by Gumbel softmax

Figure 3: CIFAR10 : Accuracy vs Fraction removed plot shows that decrease in accuracy has similar slope for both high and low value samples. This suggests that high value samples aren't looking any different than low value samples thereby bringing into question the quality of values estimated by DVE. A similar behaviour is seen when the sampling is done by gumbel-softmax and remaining reward signal remains the same

Figure 3a shows performance of predictor after removing fractions of training data from CIFAR10 based on the data values as given by DVRL. As we see in the figure, removing of high values and low values have almost a similar slope. This shows that DVRL didn't not have a clear distinction in what it thought of as high value versus a low value data point. Here are the results in figure 4 that authors got on running similar experiment on different datasets. They clearly show the trend of removing low samples causing slightly better or similar performance and removing of high value samples causing significant decrease in performance.

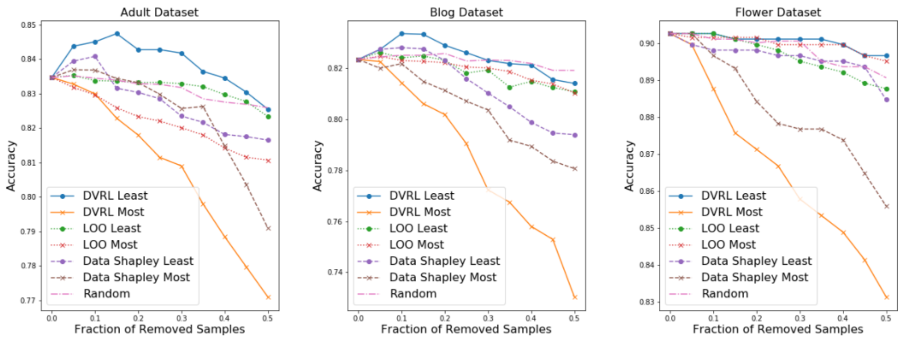
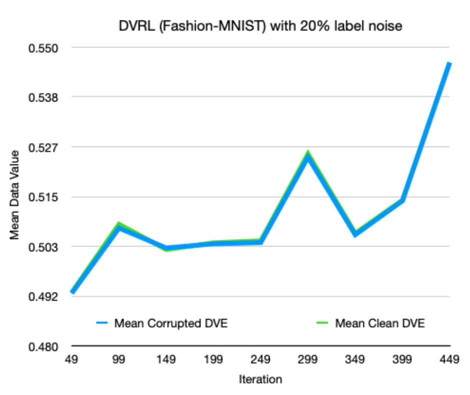
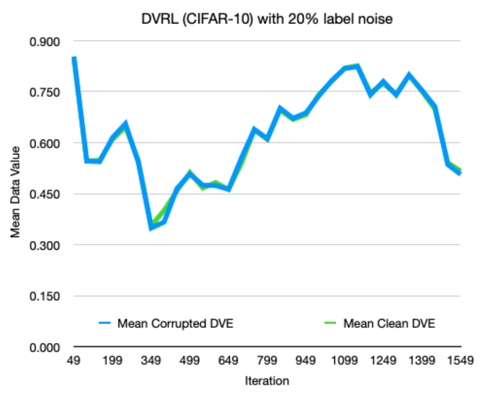


Figure 4: Accuracy vs Fractions of samples removed from original paper shows that removing low value samples in fact improves performance initially. The accuracy starts to drop eventually as we reduce the amount of training data. In case of removing higher samples, the accuracy keeps decreasing successively thereby confirming that DVRL indeed estimated high values for data that were crucial for training.

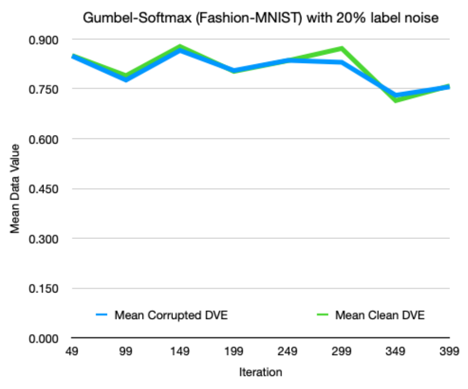


(a) Plot for Fashion MNIST

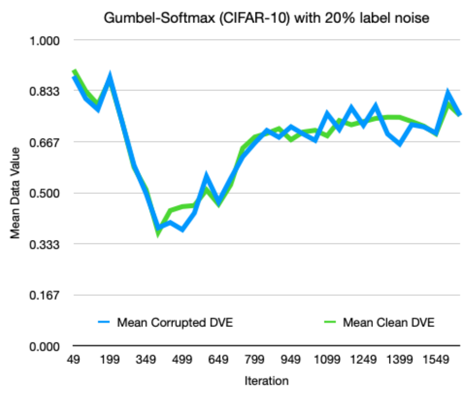


(b) Plot for CIFAR10

Figure 5: Mean DVE values for corrupted and clean samples using DVRL show no distinction. Ideally DVRL should have been able to differentiate between mean values of corrupted and clean samples with corrupted samples having a smaller mean than clean samples



(a) Plot for Fashion MNIST



(b) Plot for CIFAR10

Figure 6: Mean DVE values for corrupted and clean samples using Gumbel-softmax follow similar trends as DVRL showing its ineffectiveness in estimating data values correctly.

7 Our hypothesis - why does this not work?

The above section described the experiments that we implemented in order to further understand the efficacy of the framework as specified in [11] and we see that while accuracy in table 1 is reasonable, it is unclear to us why the accuracy is better - we think it is simply an artifact of training the model for several more iterations under the DVRL or the Gumbel-Softmax framework. We spent a significant effort ensuring that our implementation is bug-free and while there is always a chance of some subtlety creeping in, we are fairly confident in the correctness of our implementation. In this section, we discuss some of the aspects that we think might be causes for why the premise of estimating data values per sample may not work. Of course, this section is not necessarily proven science but possible hypothesis that we think might be causes for failure.

Deep Neural Networks are powerful function approximators and can be (to a degree) robust to noisy/corrupt data The DVRL framework makes a strong assumption that the reward signal over a clean validation is reliable enough to learn strong policy parameters. While that sounds like a practical choice, we observe that the validation results are not necessarily a very strong signal. The reason is that deep neural networks are strong function approximators and are known to be robust to noise and as such, we observe that while the training performance saturates - the validation accuracy improves as training proceeds. Since the performance on the validation set is the only source of reinforcement learning signal (or back-propagation through gumbel-softmax), we posit that it may not be a strong enough signal.

The problem as its framed is not a pure MDP and as such creates challenges In learning data values through reinforcement learning, we use policy gradient methods that are designed to work under Markovian dynamics. Under our framework, unfortunately the dynamics aren't deterministic given the current state and selection vector. As such, the selection vector in the first epoch has an effect on the predictor model weights even until the last epoch. Also, the environment is continuing and we don't see the effects of actions immediately - while the predictor model continues to learn while training proceeds, the selection vector at the particular iteration is not fully determining the improvement in performance.

Low-valued data points need to have sufficient commonality for the model to learn discriminative features In the experiments that we implemented, the input to the DVE model is a vector $\mathcal{X}||\mathcal{Y}$ and while we corrupt the labels, it is challenging for the model to learn discriminative features for the input samples possibly due to the lack of commonality between corrupt data points. We also confirm this hypothesis when we explicitly train DVE on the corruption signal as mentioned in section 5. It is also likely that the input to DVE may not be sufficient to learn the value of the data and it seems that the authors noticed that and used marginal information as additional input - although that doesn't seem to be very helpful in learning stable data values.

DVE fails to learn even with explicit corruption signal In order to understand the failure modes better, we tried a very simple ablation study - we tried to learn the Data Value estimates by explicitly giving the Data Value Estimator the corruption labels. In our experiments, we corrupt the labels for a fraction of the data points and obtain a value $c_i \in (1, 0)$ indicating the presence and absence of corruption respectively. With this experiment, we aimed to understand if the failure of the experiments lied with the reinforcement learning signal or the Data Value Estimator and hence trained the Data Value Estimator to reduce the cross entropy loss between the learned data values $h_\phi(x_i)$ and c_i. We observe that even with this explicit corruption signal, the model fails to learn good data value estimates and as such is a random predictor. This makes us question the idea if per-sample data values are learnable?

8 Conclusion

In this work we explore the DVRL framework [11] and work on extending and contrasting it with the Gumbel-softmax trick [6] [1]. The premise of the project is to learn data values per datum in order to improve the performance of the learning process. The value of the data point determines how likely it is to be included in the training process. If effective, the framework holds a lot of promise in a versatile set of applications including but not limited to Data Valuation, Corrupted Sample Discovery, Augmenting High Value Datasets with Cheap Sources, and Domain Adaptation. We implement and experiment with the framework and describe the challenges faced in the implementation process. We learn through the experiments that the framework is hard to stabilize and does not learn what it is supposed to and we reason through the failure modes and hypothesize reasons for the same. We end this project with an open-ended question - It would be interesting to work on this framework and understand the validity of the proposed solution - more importantly asking the if the posed question (learning per sample data-values) is the right one?

References

[1] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax.* 2017. arXiv: `1611.01144 [stat.ML]`.

[2] Wouter M. Kouw and Marco Loog. *An introduction to domain adaptation and transfer learning.* 2019. arXiv: `1812.11806 [cs.LG]`.

[3] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: `http://www.cs.toronto.edu/~kriz/cifar.html`.

[4] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: `http://yann.lecun.com/exdb/mnist/`.

[5] Yann Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[6] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. *The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables.* 2017. arXiv: `1611.00712 [cs.LG]`.

[7] Chuanqi Tan et al. *A Survey on Deep Transfer Learning.* 2018. arXiv: `1808.01974 [cs.LG]`.

[8] Mei Wang and Weihong Deng. *Deep Visual Domain Adaptation: A Survey.* 2018. arXiv: `1802.03601 [cs.CV]`.

[9] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3–4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: `10.1007/BF00992696`. URL: `https://doi.org/10.1007/BF00992696`.

[10] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* 2017. arXiv: `1708.07747 [cs.LG]`.

[11] Jinsung Yoon, Sercan Arik, and Tomas Pfister. "Data Valuation using Reinforcement Learning". In: 2020.