One Shot Robust Parameter Generation

Aashi Manglik amanglik@andrew.cmu.edu
Nilesh Kulkarni nileshk@andrew.cmu.edu
Rawal Khirodkar rkhirodk@andrew.cmu.edu

Abstract

We propose to learn a generator conditioned on the data to regress directly over parameters of the classifier. Our problem set-up involves having two kinds of data-sets a) Inhand-dataset b) Oracle-dataset. In the inhand dataset, the model has access to the training points and corresponding labels, while for the oracle dataset the model only has access to how well it is doing on the oracle dataset. The objective is to output a better classifier by repeatedly querying the oracle-dataset and understanding the points of vulnerability of the predicted classifier. For now, we test our model on synthetic data and show that it is possible to learn a representative model which is able to generate classifier weights by just looking once at the training data.

1 Introduction

Current classification algorithms rely heavily on train and test set distributions and are very data-driven. An important problem is to move away from data-based models which only work on classifying a given train set. In this proposal we propose a meta-learning algorithm that can learn to find a set of parameters for the underlying classifier without performing back-propagation over the generated parameters. For instance, consider all 2-cluster data-sets drawn from independent gaussians in 2-dimensional space. We would like to learn a neural network that would take input the data and output the linear separator for the two gaussians. Though this problem is very simple conceptually and SVMs [10] are known to to find the max-margin classifier, it requires to solve a Quadratic Program every time we receive a new train set data. On the contrary our algorithm takes input the data and spits out a classifier that would be max-margin on the real underlying distribution of the data.

We know the classifiers robust to noise are max-margin classifiers. Especially in the case of SVMs solving for the max-margin classifier comes from the problem formulation which tries to add the max-margin as an objective in the optimization. Here, we propose to learn robust classifiers by using neural networks without enforcing such a constraint in the objective. Learning robust features over a training set involves various tricks like adding noise to the training, data augmentation but such hacks and trick still cannot cover the complete underlying distribution. This is the same reason why validation/test performance of our networks are lower as compared to the train set.

Second important problem is that there exist multiple solutions to a particular problem and there is no unique classifier that is the best classifier. There is only one Bayes optimal solution but it requires having access to the underlying distribution of the data which more often is not available. Thirdly, we would want our algorithms to understand the data points that are most adversarial to their performance. Keeping this in mind we would want our classifier to hallucinate modes of error and improve the generated parameters so that it becomes more robust on the test set.

In this work, we trained a generator network described in 4.2.1 that outputs the parameters of a linear decision boundary on the given dataset. The output classifier is accurate on training data which needs to be improved further to generalize well to test data and be robust to noise. To impart robustness, we optimize over a reward given by hidden dataset as described in 4.3.

10.701 Class Project 2018.

The contributions of this paper are twofold.

- A generator network is introduced that outputs the classifier of any given linearly-separable dataset in one forward pass.
- The solution given by the generator network is robust to noise and generalize well to test data. This is facilitated by using a policy gradients approach where our agent is the generator network and parameters of classifier are analogous to the actions taken by agent. The world then evaluates the classifier on different input points whose true labels are unknown to the agent. This evaluation score becomes the reward given to the generator network and our method tries to maximize this reward.

2 Related Work

Optimizing neural networks has always been a problem, and a lot of approaches have been tried to address the issue of optimization by proposing various optimizers like SGD, Adam [6], that try to optimize on the heavily non-convex manifold of these loss functions. Such a non-convex manifold suggests existence of various possible optimums, but most of our current approaches converge to one such extrema near their initialization. Optimizing for maximum likelihood leads to over fitting in neural networks.

Bayesian neural networks [1, 8] define an alternate model where the prediction of the function is an expectation over all possible values of the parameters where the parameters are drawn from a learned probability distribution. There has been recent work on modelling parameters of a neural network using high capacity neural networks that try and optimize over the parameter space [4]. Various approaches have been tried to explore the dynamic routing [12] where the prediction function is non static in architecture. [2] models a distribution over weights implying that no one particular combination of the weights is best. Such modelling leads to creation of ensemble of networks that can have multiple predictions for the same neural nets.

3 Dataset

3.1 Synthetic Dataset

We used a synthetic dataset $\mathbb{D}$ for the task of binary classification. For every $D \in \mathbb{D}$, we sample two clusters each consisting of 500 points. Each cluster is sampled from a multivariate Gaussian distribution of randomly chosen mean and covariance. Thus, the training dataset can be denoted by $\mathbb{D}_{\mathbb{T}} = \{D_1, D_2, D_3, ..., D_K\}$. Each test example consists of a binary classification task on randomly created 2-clusters. Here, we report the results from bivariate Gaussian distributions.

4 Methods

In this work we are trying to solve two sub-problems.

- Search for Solution: We try to learn a model which predicts a solution to the problem by just observing the data (this is what we mean by *one shot*)
- Search for Good Solution: We address the fact that not all solutions are desirable, you might be interested in a particular kind of solution. We further tried to model only such desirable solutions. Our metric for defining "goodness" of the solution is simply how it performs on both training and test data.

For ease of understanding we divide the methodology section into three subsections. First subsection introduces notation relevant to our work. Next two subsections are devoted to solving both the sub-problems introduced above.

4.1 Notation

- $\mathbb{T}$ be the task of interest for which we seek to find solutions, our preliminary work focuses on $\mathbb{T}$ = binary classification.

- $\mathbb{D}$ is the distribution of all possible datasets pertaining to the task $\mathbb{T}$.
- Dataset $D \in \mathbb{D}$, where $D = \{(x_1, y_1), (x_2, y_2), ...(x_N, y_N)\}$
- For simplicity, we assume number of data points N and dimensionality p of each data point is same for all $D \in \mathbb{D}$
- $S(D)$ represent the set of all solutions which successfully solves the dataset D in the context of task $\mathbb{T}$
- $\theta \in S(D)$ represent one such solution to the dataset D.
- $\Theta = \bigcup_{D \in \mathbb{D}} S(D)$. Θ represents the solution space of all our datasets $D \in \mathbb{D}$.

4.2 Search for Solution

We wish to learn a function $G : \mathbb{D} \rightarrow \Theta$ such that the function G directly maps the dataset D to one of the solutions in the set $S(D)$ i.e

$$G(D) = \theta, \theta \in S(D)$$

We know proceed with modelling and learning this function G.

4.2.1 Modelling G

We model the function G using a fully connected neural network. G consists of two hidden layers with ReLU activations.

In our experiments we set $N = 1000, p = 2$, i.e each dataset $D \in \mathbb{D}$ contains 1000 data points and $x \in \mathbb{R}^2$. As $\mathbb{T}$ is binary classification, $y \in \{-1, 1\}$ and we ensure equal proportion of both classes in the dataset D. This also implies that our solution space $\Theta \subset \mathbb{R}^2$.

Furthermore, D can be represented as concatenation of $[{x_i}^T, 1, y_i] \forall i \in \{1, 2, 3..N\}$, overall a flat 4000 dimensional vector which acts as our input to G.

To model the uncertainty in our output θ, we use the probabilistic approach and predict the mean θ_μ and standard deviation of the θ_σ representing the normal distribution of θ, i.e

$$G(D) = \theta_\mu, \theta_\sigma$$

$$\theta \sim \mathcal{N}(\theta_\mu, {\theta_\sigma}^2)$$

$$\theta \in S(D)$$

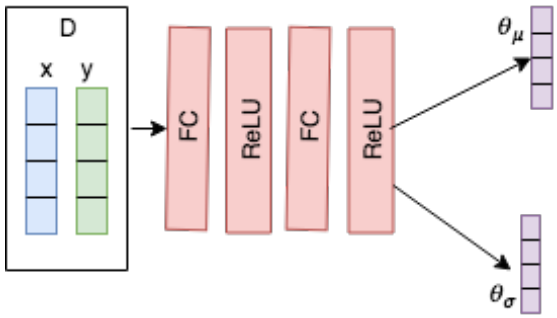

Figure 1: Methodology

4.2.2 Loss Function

We train G by evaluating the output θ on D using the Hinge Loss.

$$G = \text{argmin}_G \, L(\mathbb{D}, G)$$

$$L(\mathbb{D}, G) = \sum_{D \in \mathbb{D}} L_G(D)$$

$$L_G(D) = \sum_{i=1}^{N} l_{G(D)}(x_i, y_i)$$

$$l_\theta(x, y) = \max(0, -y\theta^T x)$$

The overall loss is differentiable with respect of parameters of G and can be trained end to end using backpropagation.

4.3 Search for Good Solution

In this section, we further investigate whether we can push our model G to find desirable solutions rather than just any solutions. For example, in the simple case of linearly separable data, there can exist many decision boundaries correctly separating the data, but the max margin decision boundary is desirable as it is most robust to noise.

In the same spirit, for the specific task $\mathbb{T}$ of binary classification in case of linearly separable data, we train our model G using reinforcement learning, hoping the solutions generated by it would be robust to noise.

Section 4.2 successfully established our model G's capability in regressing over one of the possible decision boundary $\theta \in S(D)$ by just looking at the data D once.

4.3.1 Data Split

To achieve robustness and model the real life noise, we try to hide some of our data from the network and still force it to do well on the hidden data. We believe this captures the essence of the robust solutions (for example, max margin decision boundary).

Therefore, we divide our input dataset D into two kinds

- Inhand dataset D_i: Contains 0.1 randomly sampled data points from D
- Oracle dataset D_o: Contains 0.9 randomly sampled data points from D

$$D = [D_i, D_o]$$

4.3.2 Inhand Dataset D_i

D_i acts as the visible data to our model and is the input to our network G. We compute hinge loss from section 4.2 only on the D_i to train G in a supervised fashion.

4.3.3 Oracle Dataset D_0

D_o acts as hidden real life data to our model. We only evaluate the accuracy of the output $\theta = G(D_i)$ on D_o for the task $\mathbb{T}$ of binary classification. The accuracy acts as a reward signal r to G while training. Note, the target labels y in D_o are completely hidden from our model G unlike D_i.

$$G(D_i) = \theta_\mu, \theta_\sigma$$

$$\theta \sim \mathcal{N}(\theta_\mu, {\theta_\sigma}^2)$$

$$r = accuracy(D_o, \theta)$$

$$\theta \in S([D_i, D_o]$$

4.3.4 Loss Function

The loss function has two components, we minimize the hinge loss on D_i and maximise our reward on D_o.

$$G = \text{argmin}_G \, L(\mathbb{D}, G)$$

$$L(\mathbb{D}, G) = \sum_{D \in \mathbb{D}} L_G(D) = \sum_{D \in \mathbb{D}} L_G([D_i, D_o])$$

$$L_G([D_i, D_o]) = \sum_{i=1}^{0.1N} l_{G(D_i)}(x_i, y_i) - R(G(D_i), D_o)$$

$$l_\theta(x, y) = \max(0, -y\theta^T x)$$

$$R(G(D_i), D_o) = R(\theta_\mu, \theta_\sigma, D_o) = \sum_{s=1}^{1000} p(\theta_s) \text{accuracy}(\theta_s, D_o)$$

$$\theta_s \sim \mathcal{N}(\theta_\mu, {\theta_\sigma}^2)$$

$$p(\theta_s) = \text{Probability}[\theta_s | \mathcal{N}(\theta_\mu, {\theta_\sigma}^2)]$$

We use REINFORCE [11] to fine tune the model along with the hinge loss signal computed on the inhand dataset alternatively. We sample 1000 times from the output of our network and compute reward weighted by the probability of the sampled decision boundary

The goal of this methodology is to show that our model will directly regress to the maximum margin decision boundary by just looking at the training data once without directly optimizing for a maximum margin objective function.

5 Experiments and Results

5.1 Datasets

We conduct our experiments on synthetic datasets. Our datasets have the following characteristics a) Linearly Separable b) Two-dimensional in nature. Our task is that of binary classification. Later we discuss how to extend our methods to datasets with larger dimensions and multi-class. A dataset is collection of (X, Y) tuples where X represents a set of points and Y represents corresponding labels (-1, +1) for those points. We have two kinds of datasets a) Sampled from Linear distribution b)Sampled from Gaussian Distribution.

5.1.1 Gaussian Dataset

In this dataset we create 500 (X, Y) tuples. To create one (X, Y) tuple we sample μ_1, $\mu_2 \in \mathbb{R}^2$. We then randomly sample T_N (e.g 500) points each from $\mathcal{N}(\mu_1, 1)$ and $\mathcal{N}(\mu_2, 1)$. The points from μ_1 will get a label as +1 and μ_2 get a label -1. We call this in-hand data $(X_{in-hand}, Y_{in-hand})$. Now we create one more sampling called (X_{oracle}, Y_{oracle}) where we sample V_N (e.g. 9500) points each from $\mathcal{N}(\mu_1, 3)$ and $\mathcal{N}(\mu_2, 3)$. We called this the oracle dataset.

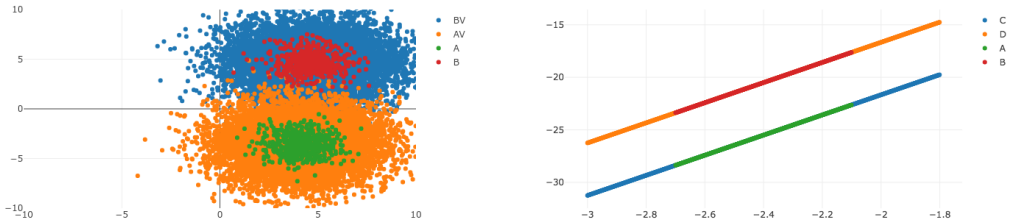

Figure 2: *Left* : Gaussian Data sample. Orange : Oracle data from μ_1, Blue : Oracle data from μ_2, Green : In-hand data from μ_1, Red : In-hand data from μ_2. *Right* : Line Data sample. Similar convention as earlier

5.1.2 Line Dataset

In this dataset we create 500 (X, Y). To create one (X, Y) tuple we sample M points on a line (basically we sample a slope, and intercept). Now given a slope m, and intercept c we sample M (e.g 10000) points in $\mathbb{R}^2$ such that they lie between (-5,5) in their first dimension. In a similar fashion another set of M points are sampled on the line at distance of 5 units from the current line with slope (m) and intercept(c) which has slope m and intercept c$'$. Now of the M sampled points we choose T_N (e.g. 500) points as the part of the inhand data $(X_{in-hand}, Y_{in-hand})$ and the rest $V_N = M - T_N$ (e.g. 9500) points as part of the oracle data (X_{oracle}, Y_{oracle}). We do the splitting process for both the datasets and assign the line with intercept c as +1 labels and the one with intercept c$'$ as -1 labels

5.2 Evaluation : Quantitative

We evaluate the network G on the above two mentioned datasets as toy task. We used 500 data tuples from the gaussian data process to train. Below in Table 5.2 we report our performance on oracle and in-hand datas using both the methods described above as Supervised and Supervised + PG

Method	In-Hand	Oracle
Supv	95.2%	89.3%
Supv+RL	97.3%	92.6%

5.3 Evaluation : Qualitative

In the following examples we see the difference between the classifiers generated by the above two methods As we see int the Figure 3 the observation we have is that once the Supv method achieves

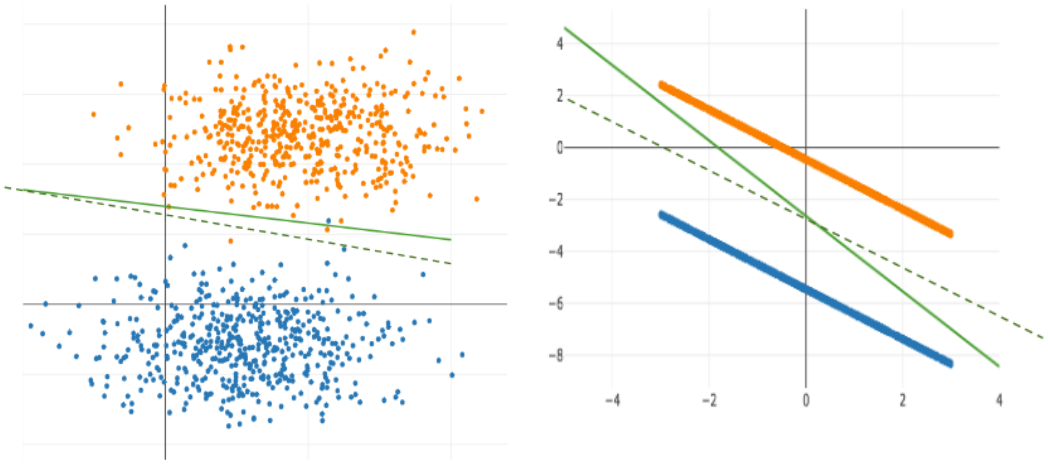

Figure 3: The thick line represents the classifier learned via using the supervised (Supv) method, and the dotted line represents the classifier when using the Supv + PG. *Left* Gaussian data, *Right* Line data

100 % accuracy on the In-Hand data, because of our loss formulation (hinge loss) the loss values drops to zero for any classifier lying in the margin between the two data clusters. This is traditionally handled by adding the max-margin constraint to the objective and in most non-convex optimization methods achieved by regularizing the model parameters. In our case introduction of the Oracle dataset creates a kind of "goodness" measure between all the possible solution in the river of classifiers. We see that the dotted lines in Figure 3 may not be the max-margin classifiers but we can surely see that they are more robust than the non-dotted lines.

6 Conclusion and Future Work

There are several extensions of the above work that are crucial and needed to verify the hypothesis further. Following are the avenues where possible contributions can be made in future.

- Extending to higher dimensions. Currently we have only demonstrated how the method works on 2D data and the corresponding visualizations. We would like to extend the above approach to higher dimensions but we expect the curse of dimensionality to play an important role. We also believe that searching through all possible solutions would be not as trivial as explained above.
- Extending the work to data-sets which are not linearly separable. Our proof of concept has only been showed on a set of datas which are linearly separable which is similar to the assumption that SVMs make. We believe that ideas from kernel trick can be straight away applied to our method with a small change to the architecture. Instead of learning to predict weights of the classifier predict the weights for various data-points in the in-hand data-set.
- Generating parameters of a neural network is what has been explored by us, but there have been several papers that explore these ideas like [3, 13, 7]. Regularization is a serious issues with neural networks and over-fitting to the train-set and lack of generalization has encouraged researchers to constantly look for techniques and engineering hacks to avoid over-fitting. [9, 5]

References

[1] Christopher M Bishop. Bayesian neural networks. *Journal of the Brazilian Computer Society*, 4(1), 1997.

[2] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[3] Lior Deutsch. Generating neural networks with neural networks. *arXiv preprint arXiv:1801.01952*, 2018.

[4] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. 2017.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

[8] Radford M Neal. Bayesian training of backpropagation networks by the hybrid monte carlo method. Technical report, Citeseer, 1992.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[11] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[12] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. *arXiv preprint arXiv:1711.08393*, 2017.

[13] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.