Quantifying Pointwise Stability of Neural Networks

Anonymous Authors[1]

Abstract

1

1. Motivation

One of the surprising phenomena of deep learning is that DNNs rarely overfit. DNNs have been shown not to over-fit even when the number of parameters is much larger than the number of examples. This phenomenon cannot be explained using structural measures of complexity of the DNN, such as number of parameters or VC dimension. *Margins* based arguments have been used to explain the stability when the data distribution is well matched to the DNN (Bartlett et al., 2017).

These analysis methods provide stability guarantees that are *global* in nature. The bounds proven using this approach are on the difference between the true error rate and the training or test error rates. Global bounds can be used to bound the difference between the training error rate and the test error rate, which are global quantities. On the other hand, it provides no indicatin of where the errors are more likely to occur. In this paper we suggest a *local* stability measure which distinguishes between stable instances, on which our prediction can be confident, and unstable instances, on which our prediction is uncertain.

Our proposed algorithm follows the popular practice of independently training several DNNs and then combining them using a majority vote. Figure 1 is an intuitive explanation of the way the algorithm works in the binary case. The analysis is based on the interaction between two spaces: the sample space and the classifier space. We take advantage of the fact that the output of most learning algorithms, including DNNs, is sensitive to small changes in things such as reordering the training examples, changing the starting point, or bootstrapping the dataset. We call the set of all classifiers that are likely to be generated the **support set**. Each classifier in the support set defines a partitioning of the sample space into instances labeled +1 or -1. An instance is called **easy** if all of the classifiers in the support set agree on it's label. And example is called *hard* if about half of the classifiers in the support set predict +1 and half predict -1.

It is well known that taking the majority vote over the classifiers in an ensemble improves robustness and accuracy. The novelty of our approach is that we use the majority vote only when the fractions of rules in the ensemble that predict +1 / -1 on and example is far from 50%/50%. when the number is close to 50% / 50% our classifier outputs "I don't know", which is represented as predicting with the set $\{-1, +1\}$. When classifying into more than two classes there are many different types of "I don't know". We adopt the idea of *set predictions* from the work Conformal Prediction (**?**).

Set predictions provide a detailed quantification of prediction uncertainty. Here are a few examples for when set predictions are useful.

- **Different costs for different mistakes:** for a self-driving car, it might be more important to determine whether an object is a human, less important to determine gender or age. Thus it is enough to determine whether the object is in the set {man, woman, boy,girl}.
- **Detection cascade:** Detecting the location and size of objects in video requires processing a rapid stream of candidate windows to detect a small fraction which contain the object. In their work on face detection (**?**) Viola and Jones introduced the detection cascade as a method for reducing the *average* processing time by using a cascade of detectors, each filtering out windows that clearly contain no face. This can be expressed as predicting the sets {not a face} and {a face, not a face}.
- **Many Classes:** When the number of classes is large, it can be useful to reduce the number of possibilities using a cascade: First stage identifies a subset, second stage reduces the subset, ... until a single class is determined.

Contributions: Our main contribution is the use of an ensemble of NN to distinguish between stable and unstable instances. We develop the theoretical framework, extend

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

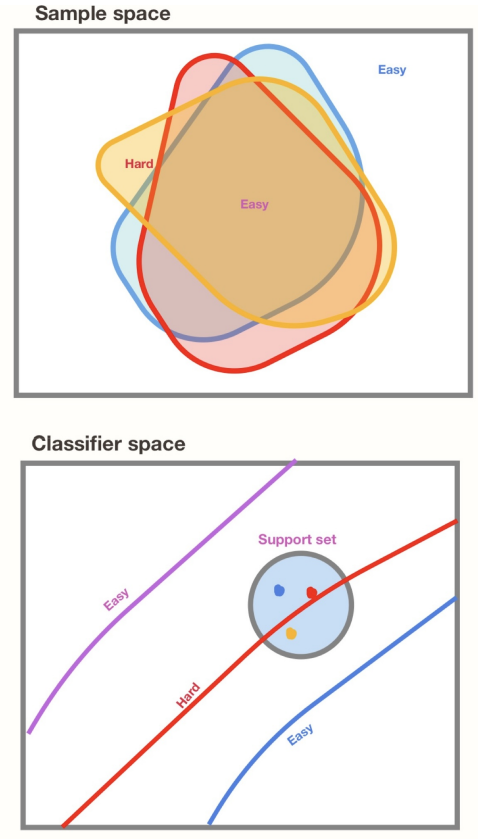

Figure 1. **support sets, easy and hard examples:** We use the dual relationship between binary classifiers and examples. We denote an instance by $x \in X$ and a classifier by $h \in \mathcal{H}$. In the Instance space X each point is an instance $x \in X$ and a classifier $h \in \mathcal{H}$ defines a subset of X: $\{x|h(x) = 1\}$. Similarly in the classifier space each point is an instance $h \in \mathcal{H}$, and an instance defines a subset of $\mathcal{H}$: $\{h|h(x) = 1\}$. The three named points in the sample space correspond to the three boundaries in the classifier space with the matching colors. On the dual side, the three points in the classifier space correspond to the three sets in the example space.

it to the multi-class setup and demonstrate it's utility on CIFAR-10 and CIFAR-100.

2. Comparison to other work

Not like calibration: does not depend on labels.

3. Support sets: definition and estimation

Learning algorithms are randomized. The first unavoidable source of randomness is the choice of the training set, denoted **sample**. DNNas are usually trained using gradient-based methods. This results in two additional sources of randomness: the choice of the starting point, denoted **start** and the choice of the order of the training examples, denote **order**. Many other sources of randomness can be useful, here, we restrict ourselfves to the three above plus an additional one that will be defined for the sake of analysis.

In what follows we fix the architecture of the DNN. We denote by w the weight vector of the DNN and by W the space of all weight vectors. Let R denote a source of randomness. We denote by D^R the distribution over W produced by R.

Our focus here is on K-class classification and we adopt the convention that the output layer of the DNN consists of K units whose activations are non-negative and sum to 1, i.e. a distribution, or a point on the K dimensional simplex Δ^K. We denote by $w(x) \in \Delta^K$ the prediction of the network with weights w on the instance x. Fixing x and choosing the weight vector at random $w \sim D^R$ defines a distribution over Δ^k that we denote by $D^R(x)$.

The predictions we use are subsets of $[K] = \{1, \ldots, K\}$. We denote this set of subsets by $[[K]]$. We now define a mapping from the ensemble outputs $w_1(x), \ldots, w_m(x)$ to $[[K]]$. The mapping is parametrized by a threshold $\theta \in [0, 1]$. and defined as follows (could be made into a figure):

1. Let order

We denote the mean of this distribution by $\mu(x) \in \Delta^K$. We define the "ideal" prediction as the set of maximal coordinates in $\mu(x)$.

$$p(x) = \{i \in [K] | \forall j \in [K], j \neq i,\ \mu(x)_i \geq \mu(x)_j\}$$

Recall that $D^R(x)$ and $\mu(x)$ are not observable. The only observable quantities are the predictions made by the ensemble members $w_1(x), \ldots, w_k(x)$ which we can think of as samples from the distribution $D^R(x)$ over Δ^K.

We can formulate the

4. Multi-label classification

we predict with a set of labels L with the property that, with high probability, the true label is in L. The size of the set L quantifies the degree of our uncertainty. If $|L| = 1$ this is the standard prediction of a single class. If $L = (g, d)$ our prediction is that the label is either g or d. If L contains all possible labels, then predicting L conveys no information. This approach, which is similar to confidence sets in regression, was developed in (**?**).

5. The confidence-rated algorithm

Putting all of this together, we arrive at the following algorithm:

1. **Create ensemble:** Start with an empty ensemble and repeat until new classifiers are added only rarely:

(a) Run a perturbation of the base learning algorithm to generate a classifier.

(b) If the classifier has at least ϵ disagreement with each of the classifiers in the ensemble, add it to the ensemble.

2. **weigh classifiers** each classifier in the ensemble according to $w(h) \doteq e^{-\eta\hat{\varepsilon}(h)}$.

3. **Predict** the label of example x to be $+1, ?, -1$ according to equation (**??**)

6. Experimental Results

Error rates for different scoring methods				
size of prediction set	Margins		random starting points	
	Incorrect	Correct	Incorrect	Correct
1	1.7	37.8	1.8	46.2
2	1.2	18.7	2.5	26.1
3	0.9	13.4	1.7	13.2
4	0.7	10.6	0.8	5.5
5	0.5	7.5	0.3	1.7
6	0.2	4.0	0.0	0.3
7	0.1	2.0	0.0	0.0
8	0.0	0.7	0.0	0.0
9	0.0	0.1	0.0	0.0
10	0.0	0.0	0.0	0.0

The results we have are for CIFAR which has 10 categories. The notion of "I don't know" becomes complex.

We can overcome this by looking at particular pairs of labels, some that are easy to discriminate and some that are harder. Another approach can be to partition the set of labels into two sets and distinguish between those.

In order to have publishable experiments on drifting, we need a good drifting dataset, preferably a binary one.

We might be able to report performance on Brand Safety. But I (yoav) find irreproducible results frustrating.

ScoreHist-unbounded.png

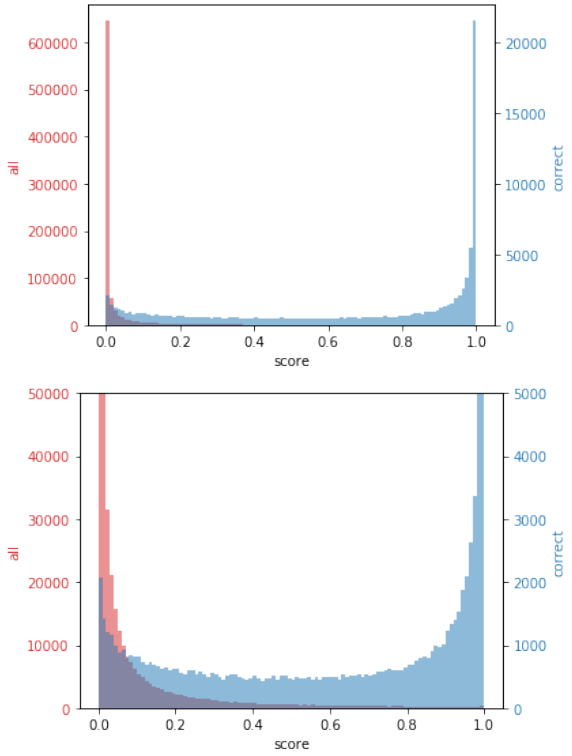

Figure 2. **Score Histograms**

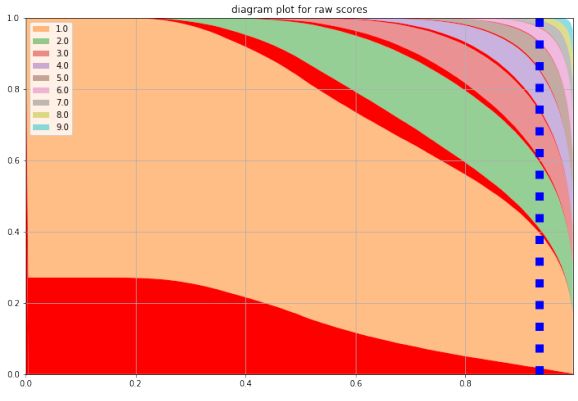

Figure 3. **Margins**

References

Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. Spectrally-normalized margin bounds for neural networks. *Advances in Neural Information Processing Systems*, 30: 6240–6249, 2017.

Freund, Y., Mansour, Y., and Schapire, R. E. Generalization bounds for averaged classifiers. *The annals of statistics*, 32(4):1698–1722, 2004.

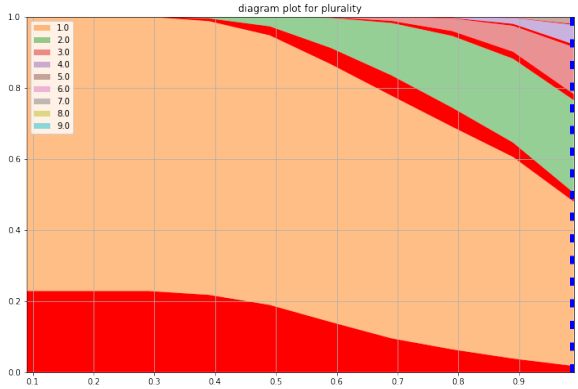

Figure 4. **RandomStartEnsemble**