Uncertainty estimates and out-of-distribution detection with Sine Networks

Hartmut Maennel [1]

Abstract

When a network makes predictions, how can we assess the (un)certainty of those predictions? How can we know when the network does not know? A simple approach uses ensembles of networks which give a set of predictions instead of only one. However, we find that different networks often produce similar outputs even for test inputs which have nothing to do with the training inputs.

A surprising remedy is replacing the usual ReLU (or sigmoid) activation functions by $sin(x)$ and adjusting the initialization. While this has little effect inside the training distribution, it greatly enhances the ability to detect model uncertainty outside of the training distribution.

1. Introduction

In this note we will focus on classification problems (but the method presented here applies equally to the regression case, see appendix A.1 for a toy example). The problem we investigate is how to estimate a useful uncertainty measure of such a prediction. We treat the training data set as fixed and consider only the model uncertainty.

We evaluate this uncertainty mainly with view to out-of-distribution detection: If we have meaningful uncertainties, the classifier should in particular "know when it does not know the answer". This means it should only concentrate all probability on one label for input that "belongs to the same type of input" as the training examples.

We also evaluate the method with respect to flagging of "difficult" examples inside the training distribution.

The "classical" approach to getting a distribution of possible labels instead of a single prediction is to use Bayesian Neural Networks ((Neal, 1996), (Barber & Bishop, 1998), (Blundell et al., 2015)). However, their use is computationally demanding.

On the other extreme, the simplest approach is to use the softmax output of a neural net as a confidence measure. It is well known that the softmax output tends to be "overconfident", so we cannot interpret it directly as a probability for the chosen class. However, it still tends to be more confident for samples from the correct distribution than for outliers, so setting a threshold for the softmax output can to some extent distinguish between "in distribution" and "out of distribution" samples. This is the "baseline method" for outlier detection suggested in (Hendrycks & Gimpel, 2017).

In (Lakshminarayanan et al., 2017) this baseline is improved by using an ensemble of classifiers (and adversarial training): Often each random initialization of a neural network gives an overconfident classifier, but different classifiers disagree - averaging the softmax output over an ensemble of classifiers then gives an improved signal.

Related is the suggestion of "MC dropout" (Gal & Ghahramani, 2016), here the ensemble of networks is replaced by one network, from which different predictions are produced by Dropout at prediction time.

2. Proposed method

A limiting factor for outlier detection with ensembles of networks seems to be that often outputs of different networks agree to an "unreasonable" extent on inputs outside of the training distribution. To illustrate this potential problem, we look at a 2-dimensional toy example, in which we can plot how the classifiers generalize from the training distribution to other points.

As input we use only a line, with labeled training samples given as blue dots in the middle and red dots at the end (Figure 1).

Figure 1. Input for the 2-dim example.

We will use a ReLU network with one hidden layer of 100 neurons to classify all points in the surrounding area of the plane. Colors encode the "probabilities" that the softmax computes from the output layer, with blue / red signifying

[1]Google Brain, Zurich, Switzerland. Correspondence to: Hartmut Maennel <hartmutm@google.com>.

Presented at the ICML 2019 Workshop on Uncertainty and Robustness in Deep Learning. Copyright 2019 by the author(s).

"probability 1". The result is in Figure 2.

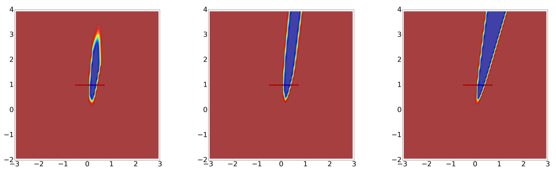

Figure 2. ReLU networks, extrapolating to 2d.

We see that most points (even far away from the training set) are either blue or red, i.e. the classifier is very confident; also there is only limited variation for classifiers initialized from different randomly chosen weights.

If we change the activation function of the hidden layer from to ReLU to $\sin(x)$, we obtain different results outside of the training distribution (Figure 3). While individual points are still mostly red or blue, i.e. the individual classifiers are still "overconfident", the different classifiers now tend to disagree.

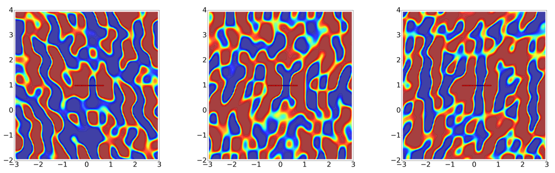

Figure 3. Sine Networks, extrapolating to 2d.

Averaging the softmax outputs over several of them we get much more realistic confidences for the Sine Networks than for the ReLU networks, see Figure 4.

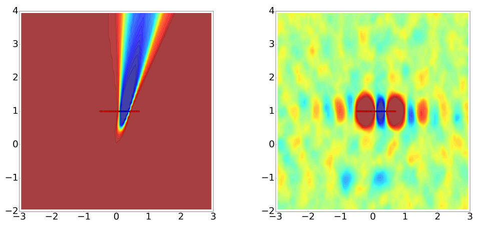

Figure 4. ReLU and Sine network ensembles, averaged.

In general, our proposed new method modifies the ensemble networks in two ways:

- Use higher initialization than usual for the first hidden layers.
- Use $\sin(x)$ as activation function in the last hidden layer.

We will see that this has little effect inside the training distribution, but significantly increases the observable uncertainty outside of the training distribution.

3. Main example

We look at fully connected networks with two hidden layers of 200 neurons, which we train on MNIST. For the baseline network, we use ReLU activations and the standard initialization of the weights ($\sigma = \sqrt{2/n}$ with $n = 784$ for the first layer, $n = 200$ for the second layer). For the Sine Network we change the activation function of the deeper layer to $\sin$ and use $\sigma_1 = 0.5$ and $\sigma_2 = 0.001$.

We train the networks on the training set of MNIST (60000 images), and evaluate them both on the test set of MNIST and on other data sets, here we use notMNIST (Bulatov, 2011) as the "outlier" data set (see the appendix A.2 for other data sets and the corresponding results).

By choosing a threshold for the "probability" (softmax output) given to the predicted label, we can measure which fraction of samples from MNIST (x-axis) and notMNIST (y-axis) are below the threshold, varying the threshold gives the ROC curve, see Figure 5. We get similar curves for other data sets, see appendix A.2.

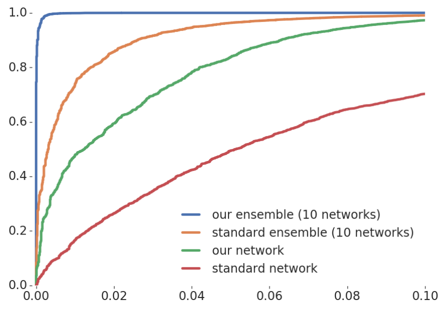

Figure 5. ROC curves for network ensembles trained on MNIST, evaluated on notMNIST.
x-axis: MNIST samples not recognized as belonging to MNIST,
y-axis: notMNIST samples recognized as not belonging to MNIST.

While using an ensemble of ReLU nets gives a very substantial improvement over using one ReLU net, the result is still not close to what humans would achieve: The classifiers used here have some 98% accuracy, if we allow not recognizing an image as MNIST digit in 2% of the cases, we might expect to recognize that the printed letters of the alphabet are no handwritten digits in most cases, but in the above graph 2% error rate for MNIST corresponds to "only" 80% success rate on notMNIST. However, the ensembles of Sine Networks recognizes most of notMNIST images (98%) if we allow 2% error rate on MNIST.

In the literature, the performance of such "out of distribution" detection is often measured by the area under the ROC curve, in our example this would be 97.4% for the ensemble of ReLU networks, and above 99.9% for the ensemble of Sine Networks. As comparison, (Hendrycks & Gimpel, 2017) report 97% in Table 1 for their "baseline" classifier (they use a slightly larger network and GELU activation functions), and (Liang et al., 2018) report 98.2% with their method that uses "outlier exposure", i.e. trains also on images that do not belong to MNIST (but also not on the "notMNIST" images that are used for evaluation).

To compare our method also to the evaluation given in (Lakshminarayanan et al., 2017), we compute for each image the entropy of the softmax output and plot a histogram over the MNIST and the notMNIST data set for the ensemble sizes 1,5,10. The first three columns of Figure 6 (copied from Figure 3 in (Lakshminarayanan et al., 2017)) show the resulting entropy of the softmax output for ReLU networks.

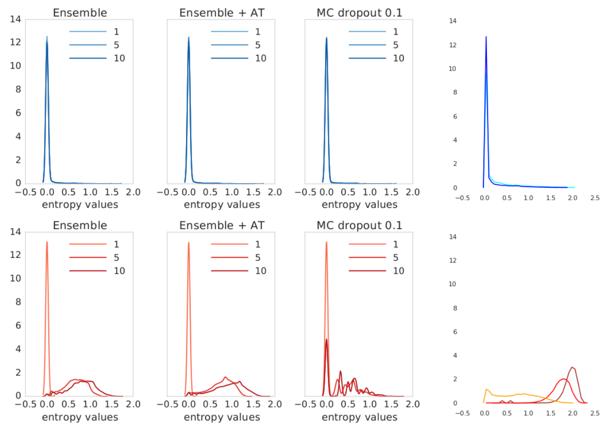

Figure 6. Histogram of the predictive entropy from Figure 3 in (Lakshminarayanan et al., 2017) (first three columns) compared to the corresponding Sine Networks (last column). Top row: Known classes (MNIST), bottom row: Unknown classes (notMNIST).

For input from the MNIST test set (first row, blue curves) the entropy is almost always close to 0, i.e. the classifier puts almost all probability on one label. For input from the notMNIST data set the classifier ensembles distribute the probability over several labels, resulting in much higher entropy. Use of adversarial training (second column) increases the entropy even a bit further for the notMNIST examples.

The fourth column shows the results for the corresponding Sine Networks. On the MNIST test set the behavior is basically the same as the usual network, but on notMNIST even one network often is unsure about the label, and an ensemble of 10 networks gives most of the time an entropy around 2 (the maximum would be $\log(10) \approx 2.3$ for the uniform distribution across all 10 labels).

So far we have used the "confidence" estimate only for detecting input which does not belong to the training distribution. Another application rather targets the low density parts of the input distribution: We can use it to flag "difficult" cases. We use the label with the highest averaged softmax output as the prediction, and the corresponding value (averaged softmax) as the confidence. Then we abstain for $x\%$ of the inputs with the confidence below a threshold. If the confidence is a good predictor of the classifier's ability to predict the correct label, the precision for the inputs for which we don't abstain should be high.

To get a significant number of "difficult" cases, we train only on the 10000 elements of the "test set" and evaluate on the 60000 elements of the "training set".

We get in this way:

Abstention	error rate, ReLU	error rate, sin
no abstention	4.9%	4.0%
abstain in 5%	2.5%	1.8%
abstain in 10%	1.3%	0.8%
abstain in 25%	0.30%	0.16%

4. What are the best initializations for ReLU and Sine networks?

In the case of two hidden layers it seems to be a good strategy to use a low standard deviation for the second (sine) layer, and then choose the standard deviation for the weights of the first (ReLU) layer as large as possible (such that training still converges).

In the case of one layer, we can only vary the σ for weights between the input and the first hidden layer (we keep the initialization of the output layer at the standard $\sigma = \sqrt{2/n}$).

On the x-axis we plot $\log(\sigma)$, on the y-axis the negative logarithm of the "error" := area above the ROC curve.

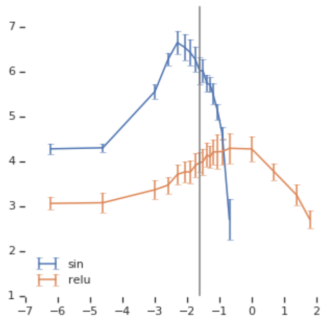

Figure 7. Influence of initialization
x-axis: $\log(\sigma)$, y-axis: $-\log(error)$, higher is better

Figure 7 shows these curves for the "test set" notMNIST, see Figure 17 in the appendix for the results for eight other "outlier" sets, and Figure 18 for results from training on fashionMNIST instead of MNIST.

The vertical line corresponds to $\sigma = 0.2$, which seems to be close to optimal for Sine Networks for all test sets. This corresponds to a standard deviation of around 1.4 for the input to the $\sin(x)$ activation function (for our encoding of the MNIST pixels, which we scale into the interval $[-1, 1]$).

We also see that "larger than usual" initialization alone already helps for ReLU networks, but using the $\sin(x)$ activation function gives an additional significant benefit.

5. Convolutional Networks

Since MNIST vs. notMNIST is too easy to profit from better classifiers, we use a more difficult problem: Train on classes 0-4, see whether we detect classes 5-9 as "out of distribution". We use two convolutional layers with 7×7 windows and max pooling in 3×3 windows with stride 2, and add two fully connected layers with 200 / 100 neurons. We also use data augmentation by randomly cropping to 24×24 pixels. For the sine network we change the activation function of the last layer to $\sin(x)$.

First experiments seem to indicate that a good way to choose the initialization for the different layers of the sine network is to start with small initializations for the first layers and the last layer, and then choose the second last initialization as large as possible.

With this setup we get for the area under the ROC curve:

Network	standard	ReLU, large init	$\sin(x)$
convolutional	95.9%	98.1%	99.1%
fully connected	95.2%	96.2	96.3%

For more complicated networks we probably need to modify the usual tricks of the trade, e.g. L2 regularization decreases the disagreement between different networks, so it should be replaced by a different method, e.g. a regularization that adds a loss depending not on the size of the weights, but on the distance to the initial initialization. Also, as further motivated in appendix A.3, we should use more general ways to encourage diversity in the lower layers than just using larger initialization. This is an area for future work.

6. How does this work?

At the moment a precise theory of how this works is still missing, but we can give some heuristics.

6.1. How large initialization increases diversity

We expect that Gradient Descent finds local optima "close to" the initial point. If we start with high random initializations, we may have a better chance that they are far apart and converge to different solutions.

This could be made more precise for ReLU networks with one hidden layer: (Maennel et al., 2018) shows that for given training data and infinitesimal initialization, there are only finitely many possibilities to which networks can converge (independent of network size). The arguments given in that paper should also show that with increasing number of weights we should with high probability end up at the same possibility (that is in a way "maximal"). So at least in this scenario we should avoid too small initialization.

We can also make a provable connection between high initialization and diversity of the network in the special case that the training input is in a lower dimensional subspace of all possible inputs. This actually happens when we train on MNIST, but evaluate on notMNIST: In the MNIST data set (but not in the notMNIST data set), some points at the corners are always white. While this does contribute to notMNIST being "easy" to detect, our proposed method works equally well for other data sets (like the "flipped MNIST" or the "masked notMNIST") so this effect does not seem to be the most important one for our method.

6.2. Non-monotonic activation functions

If we start from small initialization, the input to the sine neurons will be $x \approx 0$, and $\sin(x) \approx x$. As training progresses, $|x|$ usually increases. For $\sin(x)$ activation the increase will usually stop at the first maximum or minimum of $\sin(x)$, i.e. at $x \approx \pm\pi/2$. This means that on the training set, features (i.e. the output of the $\sin(x)$ neurons in the last hidden layer) get a large value by tuning the input to around $x \approx \pm\pi/2$, which is difficult to achieve "accidentally" for input that is not in the training set. By contrast, for ReLU or tanh activation, the inputs only have to achieve any high value.

A similar reasoning applies if the initialization is not necessarily small, but still such that the inputs of the $\sin(x)$ neurons do not vary too much compared to the distance of different minima / maxima of the $\sin(x)$ activation function. Indeed we observed that in the trained networks, applied to the training set, the inputs to $\sin$ tend to cluster around one maximum or minimum (depending on the label and the sign of the weight connecting the neuron to the corresponding output).

This reasoning also applies to other functions with maxima and minima, and indeed some of the benefit of the $\sin(x)$ activation function can be replicated by using e.g $ReLU(x) \cdot e^{-x^2}$. However, in general the $\sin(x)$ activation functions still seem to give better results for ensembles of networks, which we might attribute to more disagreement for large activations (as illustration, see the graphs of A.1 in the appendix).

References

Barber, D. and Bishop, C. Ensemble learning in bayesian neural networks. In *Generalization in Neural Networks and Machine Learning*, pp. 215–237. Springer Verlag, January 1998.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. In *https://arxiv.org/abs/1505.05424*, 2015.

Bulatov, Y. notmnist dataset. 2011. URL http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Hendrycks, D. and Gimpel, K. A baseline for detecting misclassified and out-of-distribution examples. In *ICLR*, 2017.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

Liang, S., Li, Y., and Srikant, R. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Machine Learning (ICML)*, 2018.

Maennel, H., Bousquet, O., and Gelly, S. Gradient descent quantizes relu network features. In *https://arxiv.org/abs/1803.08367*, 2018.

Neal, R. M. *Bayesian Learning for Neural Networks*. Springer, 1996.

Sirignano, J. and Spiliopoulos, K. Mean field analysis of neural networks: A central limit theorem. In *https://arxiv.org/abs/1808.09372*, 2018.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

A. Appendix

A.1. Regression in 1D

Maybe the simplest toy example to study the difference between ReLU, tanh, and sin activation functions is one dimensional regression (using the standard $L2$ loss).

If we start a ReLU network with one hidden layer

$$f(x) = \sum_i w_i' \cdot ReLU(w_i \cdot x + b_i)$$

from small initialization $b_i, w_i, w_i' \approx 0$ and optimize it with Gradient Descent to fit n points $(x_1, y_1), ..., (x_n, x_n)$, then (see (Maennel et al., 2018)) the kinks $\mu_i := -b_i/w_i$ will initially cluster at at most $2n$ points and give contributions of the form

$$\lambda_j \cdot ReLU(x - \mu) \tag{1}$$

During the Gradient Descent the number of these kink clusters does not change, and since a sum of such terms with the same kink μ will again have the form (1), this gives a simple piecewise linear function with at most $2n$ kinks.

However, for the activation function $\sin(x)$ it is no longer true that the sum of contributions

$$w_i' \cdot \sin(w_i \cdot x + b_i)$$

with the same $\mu := -b_i/w_i$ can be expressed by one such function, and also Gradient Descent will change them in different ways. This makes it plausible that even for small initialization, for which a ReLU network will converge to a simple function, the sin-Network will converge to a far more complex function. Indeed this is what we can observe experimentally, see Figure 9 and 10. The activation function tanh is sort of "in between" these two cases: For small $x \ll 0$ or large $x \gg 0$ the sum of

$$w_i' \cdot \tanh(w_i \cdot x + b_i)$$

only depends on the sum of $w_i' \cdot sign(w_i)$, see Figure 11.

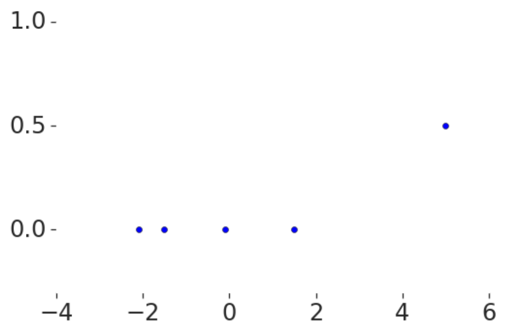

Figure 8. Regression input

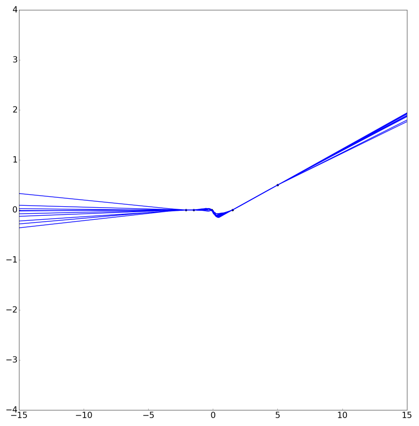

Figure 9. Sample regression functions (activation function ReLU)

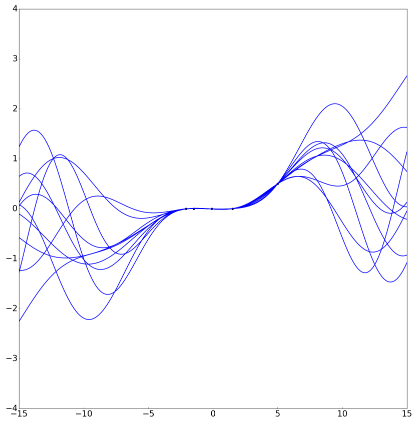

Figure 10. Sample regression functions (activation function sin)

So we see that extrapolating from the training points, ReLU and tanh networks display much less uncertainty than Sine Networks.

A.2. Test data sets

As input distributions that are "clearly different" from MNIST we can use the fashionMNIST (Xiao et al., 2017) and notMnist (Bulatov, 2011) data sets, see Figure 12.

We also use some constructed data sets:

- "Masked fashionMNIST":

  In MNIST some pixels are always or almost always white, see Figure 14. Out of fashionMNIST we derive this data set by setting all pixels to 0 (white) that have

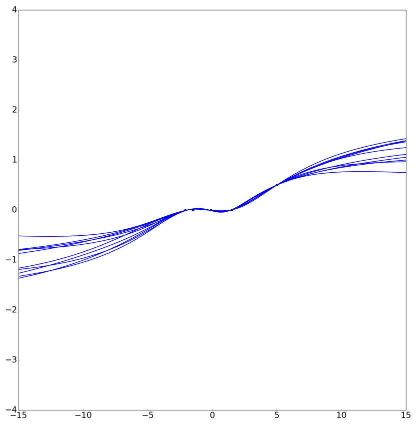

Figure 11. Sample regression functions (activation function tanh)

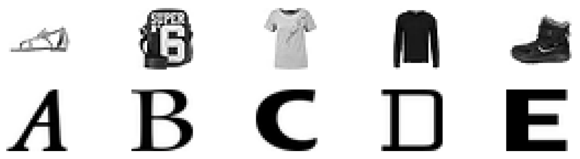

Figure 12. FashionMNIST (top) and notMNIST (bottom) data sets.

  an average gray value of below 0.01.

- "Masked notMNIST":

  This is derived from notMNIST in the same way.

- Circles and lines:

  Two lines and two circles in random positions.

- Flipped MNIST:

  Derived from MNIST by mirroring the digits 3,4,6,7,9 horizontally and the digits 4,6,7,9 vertically.

- IID noise:

  Uniform iid pixel values from [0.256].

- Independent pixels:

  Each pixel is randomly sampled from pixels at the same position at MNIST training images, independent of the other pixel values.

- PseudoMNIST: Here we try to mimic not only the value distribution for each pixel, but also local correlations. We start with an image in which each pixel is independently sampled like in the previous distribution. Then we perform 1000 (PseudoMNIST1) or 1500 (PseudoMNIST2) updates of the following form: We randomly select a 3×3 window, and find the closest match in 1000 randomly sampled images from MNIST, where "closest" is with respect to the L2 distance of the 8 outer pixels of the 3×3 window. Then we replace the middle pixel in our image with the value of the middle pixel in this "closest match" from MNIST.

We will also use a "masked version" with a mask from MNIST: see figure 14 for the origin of the mask, and figure 13 for examples.

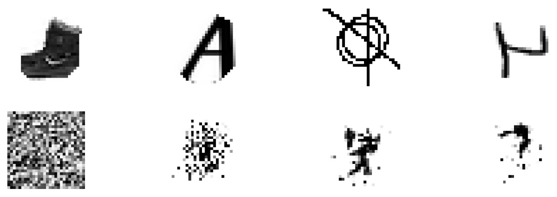

Figure 13. Constructed data sets.
Top row: Masked fashionMNIST, masked notMNIST, circles and lines, flipped MNIST. Bottom row: IID noise, independent pixels, pseudoMNIST1, pseudoMNIST2.

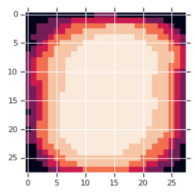

Figure 14. Average pixel value $\leq$ thresholds 0, 0.0001, 0.001, 0.01, 0.1 on MNIST

With these data sets we can again compute the ROC curves that show how well we can distinguish the training set from the outlier set by setting a threshold on the "confidence". As confidence we take again the averaged softmax output of the predicted label over an ensemble of 10 classifiers, in which the weights are randomly sampled from a normal distribution with mean 0 and standard deviation σ. We gave in Figure 5 the results for training fully connected networks with two hidden layers of 200 neurons on MNIST, and detecting notMNIST as outliers. Figure 15 shows the corresponding figure for detecting fashionMNIST as outliers. Similarly, Figure 16 shows the corresponding figures when training in fashionMNIST and detecting notMNIST / MNIST.

For fully connected networks with one hidden layer we have only one parameter for the initialization and can thus directly plot the influence of this parameter: We consider the

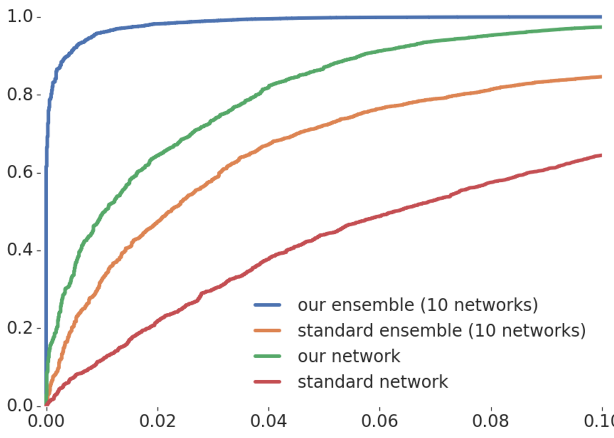

Figure 15. ROC curves for training on MNIST, detecting fashion-MNIST

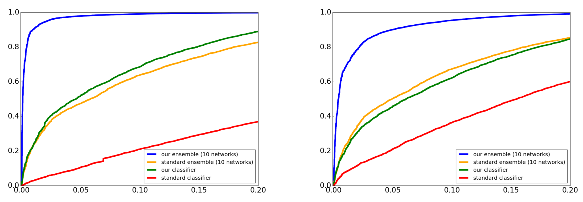

Figure 16. ROC curves for training on fashionMNIST, detecting notMNIST / MNIST

area above the ROC curve as "error" and plot $-\log(error)$ against the initialization σ used for the networks. The resulting graphs are in Figure 17.

To obtain the error bars, we train 50 networks, and randomly select 10 networks for an ensemble. Each ensemble selected in this way gives one data point, we use the average y–value to draw the curve, and ± 2 times the standard deviation to draw the error bar.

We can also train on fashionMNIST and try to detect images from MNIST or notMNIST or circles & lines, it again looks similar to the results we got when training on MNIST, see Figure 18,

A.3. Influence of the distribution from which weights are sampled

Our method for getting a variety of networks relies on different randomly sampled initializations giving different networks. According to (Sirignano & Spiliopoulos, 2018) this actually no longer happens if the number of neurons is very large: Then the networks converge to an idealized network

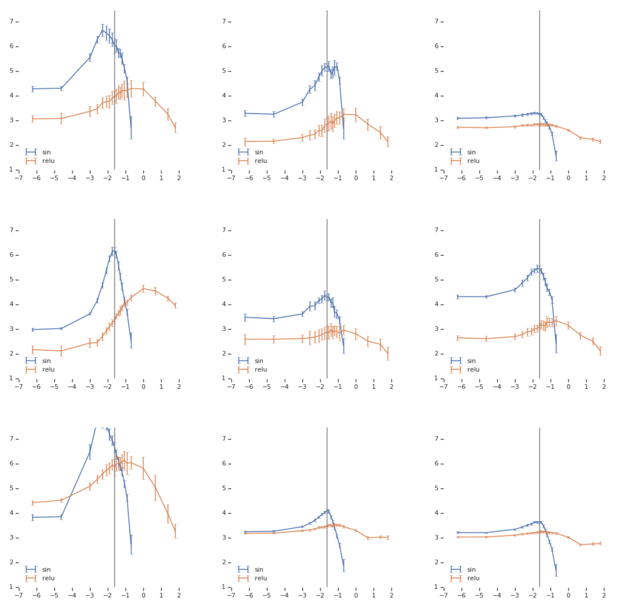

Figure 17. Influence of initialization
x-axis: $\log(\sigma)$, y-axis: $-\log(error)$, higher is better
Top row: notMNIST, fashionMNIST, flipped MNIST,
Middle row: Independent pixels, masked fashionMNIST, masked notMNIST
Bottom row: IID noise, pseudoMNIST1, pseudoMNIST2.

in which there are infinitely many neurons that have their weights distributed according to the probability distribution from which we sample. We can actually see this effect in our 2-dimensional example: When we use 20000 neurons (instead of the 100 we used in the above pictures), the results of different initializations become more similar to each other: See figure 19

So in this case, we can no longer rely on different sampling from the same distribution, but need to change the distribution itself. For higher dimensional input, this effect may be more theoretical because the number of neurons required to get into that regime is no longer realistic.

However, even for a moderate number of neurons it makes sense to vary the distribution of initial weights from which we sample to obtain a greater diversity. Here is an example in our case of 2-dimensional input, 100 neurons, ReLU activation:

The weights in the 2-dimensional case have 3 coordinates: Two for the direction and one for the bias. We sample the "direction" weights according to a Normal Distribution with standard deviation σ_1, and the "bias" weight according to a Normal Distribution with standard deviation σ_2. Depending on σ_1, σ_2 we observe different outcomes (Figure 20).

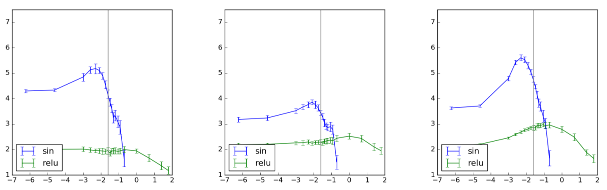

Figure 18. Influence of initialization, networks trained on fashion MNIST
x-axis: $\log(\sigma)$, y-axis: $-\log(error)$, higher is better
Test sets: notMnist, MNIST, circles and lines.

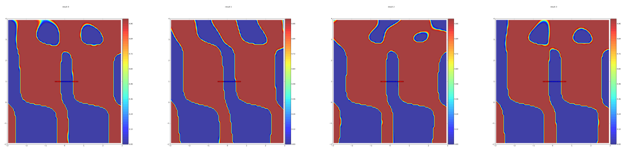

Figure 19. Different Sine networks with weights initialized by samples from same distribution, 20000 hidden sin-neurons

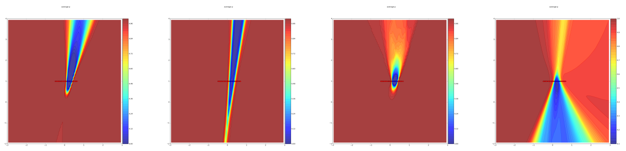

Figure 20. Ensemble average, 100 hidden ReLU-neurons
Left to right: $\sigma_1, \sigma_2 = (3, 1),\ (0.01, 0.01),\ (5, 5)\ (0.01, 1)$

In particular, the networks initialized by $\sigma_1, \sigma_2 = (5, 5)$ could in theory contain the same initializations as the networks initialized by $\sigma_1, \sigma_2 = (0.01, 1)$. However, the latter give blue cones extending downwards most of the time, whereas such a result is not among the 50 samples I obtained for $\sigma_1, \sigma_2 = (5, 5)$.

So it would make sense to restrict the distribution of weights from which we sample in different ways to obtain a more diverse set of networks.