HYPERGAN: EXPLORING THE MANIFOLD OF NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce HyperGAN, a generative adversarial network that learns to generate all the parameters of a deep neural network. HyperGAN first transforms low dimensional noise into a latent space, which can be sampled from to obtain diverse, performant sets of parameters for a target architecture. We utilize an architecture that bears resemblance to adversarial autoencoders, but with the data term substituted to be classification loss, which is equivalent to minimizing the KL-divergence between the generated network parameter distribution with an unknown true parameter distribution. We apply HyperGAN to classification, showing that HyperGAN can learn to generate parameters which solve the MNIST and CIFAR-10 datasets with competitive performance to fully supervised learning, while learning a rich distribution of effective parameters. We also show that HyperGAN can also provide better uncertainty than standard ensembles. This is evaluated by the robustness of HyperGAN-generated ensembles to detect out of distribution data as well as adversarial examples. We see that in addition to being highly accurate on inlier data, HyperGAN can provide reasonable uncertainty estimates.

1 INTRODUCTION

Since the inception of deep neural networks, it has been found that it is possible to train from different random initializations and obtain networks that, albeit having quite different parameters, achieve quite similar accuracy (Freeman & Bruna, 2016). It has further been found that ensembles of deep networks that are trained in such a way have significant performance advantages over single models (Maclin & Opitz, 2011), similar to the classical bagging approach in statistics. Ensemble models also have other benefits, such as being robust to outliers and being able to provide variance or uncertainty estimates over their inputs (Lakshminarayanan et al., 2016).

Past work has shown that deep networks are often over-parameterized (Ulyanov et al., 2017; Arpit et al., 2017). It is possible then to hypothesize that there exists a low-dimensional manifold of network parameters, where all of them achieve similarly good generalization accuracy on the same dataset. In Bayesian deep learning, there is a significant interest in having a probabilistic interpretation of network parameters and modeling a distribution over them. Earlier approaches mostly utilize dropout as a Bayesian approximation, by randomly setting different parameters to zero and thus integrating over many possible networks. (Gal & Ghahramani, 2015) showed that networks with dropout following each layer are equivalent to a deep Gaussian process (Damianou & Lawrence, 2013) marginalized over its covariance functions. They proposed MCdropout as a simple way to estimate model uncertainty. These approximations are not well aligned with current training patterns of neural networks. Applying dropout to every layer results in over-regularization and underfitting of the target function. Moreover, dropout does not integrate over the full variation of possible models, only those which may be reached from one (random) initialization.

As another interesting direction, hypernetworks (Ha et al., 2016) are neural networks which output parameters for a target neural network. The hypernetwork and the target network together form a single model which is trained jointly. The original hypernetwork produced the target weights as a deterministic function of its own weights, but Bayesian Hypernetworks (BHNs) (Krueger et al., 2017), and Multiplicative Normalizing Flows (MNF) (Louizos & Welling, 2016) generate model parameters by sampling a Gaussian prior. However, these approaches use normalizing flows to

transform a simple prior into a sample of the more complicated posterior, which compose of only bijective, invertible functions. This limits their scalability and the variety of learnable functions.

In this paper we explore an approach which focuses on generating all the parameters of a neural network, without assuming any fixed noise models on parameters. To keep our method scalable, we avoid utilizing invertible functions as in Bayesian approaches, and instead utilize the ideas from generative adversarial networks (GANs). We especially observe recent adversarial autoencoder (Makhzani et al., 2015) approaches. These approaches have demonstrated an impressive capability to model complicated, multimodal distributions in an unsupervised manner. In our approach, a random noise vector is first encoded to a number of different random vectors, and then each random vector generates all parameters within one layer of a deep network. The generator is then trained with conventional maximum likelihood (classification/regression) on the parameters it generates, and an adversarial regularization keeps it from collapsing onto only one mode. In this way, it is possible to generate much larger networks than the dimensionality of the latent code, making our approach capable of generating all the parameters of a deep network with a single GPU. As an example, in our experiments on CIFAR-10 we start from a 256-dimensional latent vector and generate all $50,000+$ parameters in one pass, consuming only 4GB GPU memory. This shows that deep networks may indeed span a low-dimensional manifold, and could spur further thoughts and research.

1.1 Summary of Contributions

We propose HyperGAN, a novel approach for generating all the parameters for a target network architecture using a modified GAN, and we do so starting from a small Gaussian noise vector which scales well with the size of the output. Our approach is different from Bayesian approaches since we do not attempt to model the entire posterior. After our GAN is trained, one can directly generate many diverse, well-trained deep models without needing to further train or fine-tune them. The diversity of the models we can generate is beyond just adding dropout or scaling factors, which is shown by the superior performance of ensembles of the generated networks.

We believe HyperGAN is widely applicable to a variety of tasks. One area where populations of diverse networks show promise is in uncertainty estimation and anomaly detection. We show through a variety of experiments that populations of networks sampled from HyperGAN are able to approximate the data distribution such that it can detect out of distribution samples. We show that we can provide a reasonable measure of uncertainty by calculating the entropy within the predictive distribution of sampled networks. Our method is straightforward, as well as easy to train and sample from. We hope that we can inspire future work in estimation of the manifold of neural networks.

2 Related Work

Generating parameters for neural networks has been framed in contexts other than the Bayesian approaches described above. The hypernetwork framework (Ha et al., 2016) has come to describe models where one network directly supervises the weight updates of another network. In computer vision this is often done with a data driven approach as seen in methods such as Spatial Transformer networks (Jaderberg et al., 2015), or Dynamic Filter networks (Brabandere et al., 2016). In these methods the filter parameters of the main network are conditioned on the input data, receiving contextual scale and shift updates from an auxiliary network. Our method instead generates weights of an entire network that can work on the entire learning problem. Furthermore our predicted parameters are highly nonlinear functions of the input, instead of simple affine transformations based on the input examples.

Recently, (Lakshminarayanan et al., 2016) proposed Deep Ensembles, where adversarial training was applied to the training of ensembles in order to smooth the predictive variance. However, adversarial training is a very expensive training process, where adversarial examples must be generated for each batch of data seen. We seek a method to learn a distribution over parameters which is does not requiring adversarial training.

Meta learning approaches use different kinds of weights in order to increase the generalization ability of neural networks. Perhaps the first proposed method is fast weights (Schmidhuber, 1992) which uses an auxiliary (slow) network to produce weight changes in the target (fast) network, acting as a short term memory store. Meta Networks (Munkhdalai & Yu, 2017) build on this approach by

using an external neural memory store in addition to multiple sets of fast and slow weights. In meta learning, the generation of each predicting network requires calling the base (slow) learner many steps, and many of the methods presented there, along with hyperparameter learning (Lorraine & Duvenaud, 2018), and the original hypernetwork, propose learning target weights which are deterministic functions of the training data. Our method instead captures a distribution over parameters, and provides a cheap way to directly sample full networks from a low dimensional manifold.

3 HYPERGAN

Taking a note from the original hypernetwork framework for generating neural networks from (Ha et al., 2016), we coin our approach HyperGAN. We learn a tractable distribution over neural network parameters, one that is simple to sample from and covers a non-trivial portion of the parameter space. We start by assuming that neural networks consist of a given architecture $\mathcal{F}$ with N layers, and a training set with inputs and targets $(x, y) = \{x_i, y_i\}_{i=1}^n$. The standard training regime currently consists of computing a loss function $\mathcal{L}(\mathcal{F}(x;\theta), y)$ and updating the parameters θ with backpropagation until $\mathcal{L}$ is minimized. This works fine if we want a point estimate of θ. However, if we want to generate more than one non-trivial network, we must instead model the distribution of θ so that we can sample diverse networks which each solve the same task with minimal loss.

To learn this latent space we observe recent work such as Adversarial Autoencoders (Makhzani et al., 2015) and Wasserstein Autoencoders (Tolstikhin et al., 2017) which use an encoder Q to learn a latent distribution $Q_z \in \mathbb{R}^d$ that matches the mean and covariance of some prior distribution $\mathcal{P}_z \in \mathbb{R}^m$, instead of modeling a single mode as commonly happens with GANs. In a conventional adversarial autoencoder for learning a latent representation of training examples, we have

$$\inf_{Q(z|x)\in\mathcal{Q}} \mathbb{E}_{P_x}\mathbb{E}_{Q(z|x)}\left[c(x, G(z))\right] + \lambda\mathcal{D}_z(P_z, Q_z) \tag{1}$$

where $x \sim P_x$, $Q(z|x)$ is an encoder, c is a measurable non-negative cost function that could be the Wasserstein distance between x and $G(z)$ (making it a Wasserstein autoencoder), and $\mathcal{D}_z$ is a divergence term which could be a Jensen-Shannon divergence or the maximum mean discrepancy divergence (Tolstikhin et al., 2017). P_z and Q_z are distributions in the latent space z, where P_z is a prior (usually Gaussian), and $Q_z \sim Q(z|x)$ is the distribution on input encodings.

An analogy of running an adversarial autoencoder to generate deep network parameters would be to first train many networks and then have the trained network parameters to be the x in eq.(1). We believe such an approach is too cumbersome and defeats the purpose of generating networks if many networks already need to be trained beforehand. Suppose the real parameters $\theta^* \sim \Theta$, we propose to replace the data term $c(x, G(z))$ with simply the training loss on the joint $P(x, y)$:

$$\inf_{Q(z)\in\mathcal{Q}} \mathbb{E}_{P_x}\mathbb{E}_{P(y|x)}\mathbb{E}_{Q(z)}\left[\mathcal{L}(\mathcal{F}(x; G(Q(z))), y)\right] + \lambda\mathcal{D}_z(P_z, Q_z) \tag{2}$$

where $z \sim P_z$, a prior distribution for which we use an m-dimensional isotropic Gaussian $\mathcal{N}(0; \sigma^2, I_m)$. Then we encode z to $Q(z) = [q_1, q_2, \ldots, q_K]$, a vector of concatenated d-dimensional embeddings from which each q_k will generate the weights of layer k via the generator $G(Q(z)$ (Fig. 1). The adversarial loss is measured between Q_z and P_z, where P_z is again a Kd-dimensional isotropic Gaussian prior. Here the encoder acts as a coupling function from the prior to the weight generators so that all the q_k (that will generate different layers) will be correlated, unlike dimensions of z which are drawn to be independent from each other. Empirical observations show that such coupling greatly improves the performance of the generated network.

To see the similarity between the data term in eq. (2) and eq. (1), we note:

$$\begin{aligned}\inf_\theta D_{KL}(P(y, x; \theta^*)||P(y, x; \theta)) &= \inf_\theta \mathbb{E}_{P_x}\mathbb{E}_{P(y|x;\theta^*)}\left[\log P(y|x;\theta^*) - \log P(y|x;\theta)\right] \\ &= \inf_\theta \mathbb{E}_{P_x}\mathbb{E}_{P(y|x;\theta^*)}\left[-\log P(y|x;\theta)\right] \end{aligned} \tag{3}$$

where θ^* is the vector of unknown true random parameters mapping x to y. Eq. (3) holds since x is independent from θ^* and θ. Hence, when we have negative log-likelihood as the loss function $\mathcal{L}$, our training loss (2) can be seen as equivalent to an adversarial autoencoder with a data term as a KL-divergence comparing the generated parameters $G(Q(z))$ against the distribution of the unknown true parameters θ^*. Hence, we do not need to sample many already-trained networks for training the

HyperGAN, and the theory in (Tolstikhin et al., 2017) also applies to the HyperGAN so that we do not suffer from mode collapse even if we only regularize in the latent space Q.

The job of the regularizer $\mathcal{D}_z(P_z, Q_z)$ is to force each embedding q_n to approximate $\mathcal{P}_z$. Intuitively, we want the encoder to provide a vector of embeddings with each having a distribution similar to $\mathcal{P}_z$ so that they span a considerable volume, but also are informative enough to generate parameters of a neural network. We utilize a GAN approach that trains a discriminator for $\mathcal{D}_z(P_z, Q_z)$. To do this we sample a fresh batch of N latent samples q, along with N samples p from $\mathcal{P}_z$ with the same dimensionality as q. The discriminator, as in the standard GAN, tries to predict which samples are real and which are fake. The loss on the discriminator is given by binary cross-entropy:

$$\mathcal{L}_{D_z} = \sum_{k=1}^{K} (\log D_z(p_k) + \log(1 - D_z(q_k)))$$

where the latent points q are the input to K parallel weight generators which output parameters $\theta_{1:K}$ for the corresponding layer in $\mathcal{F}$. The generators G themselves are neural networks, and are trained by backpropagating the loss of the target network on the training data x, y. This framework is general and can be adapted to a variety of tasks and losses, in this work we show that HyperGAN can operate in both classification and regression settings. For multi-class classification, the generators $G = \{G_1 \dots G_K\}$ and encoder Q are trained with the cross entropy loss function:

$$\mathcal{L} = N^{-1} \sum_{i=1}^{N} \mathcal{F}(x_i; \theta) \log(y_i) \quad \text{where} \quad \theta = \{G_1(q_1), \dots, G_K(q_K)\}$$

For regression tasks we simply replace the cross entropy loss with the MSE loss function:

$$\mathcal{L} = N^{-1} \sum_{i=1}^{N} (y_i - \mathcal{F}(x_i; \theta))^2 \quad \text{where} \quad \theta = \{G_1(q_1), \dots, G_K(q_K)\}$$

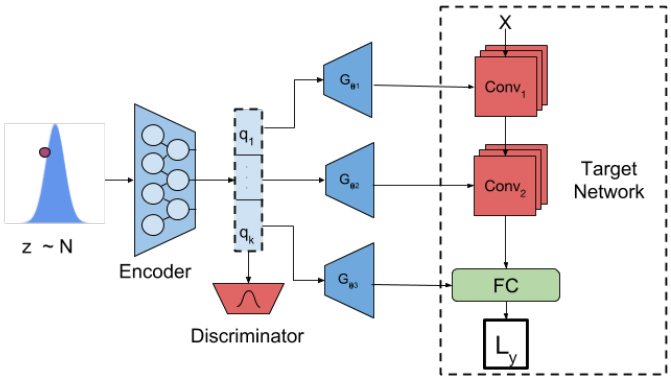


Figure 1: Example HyperGAN architecure. The encoder transforms $z \sim \mathcal{P}_z$ into a latent point Q_z. The generators each transform a latent subvector q_k into the parameters of the corresponding layer in the target network. The discriminator encourages points within Q_z to look like points from $\mathcal{P}_z$

4 Experiments

4.1 High Level Description and Experimental Setup

We conduct a variety of experiments to show HyperGAN's ability to achieve both high accuracy and obtain accurate uncertainty estimates. First we show classification performance on both MNIST and CIFAR-10 datasets. Next we examine HyperGAN's ability to learn the variance of a simple 1D dataset. We perform experiments on anomaly detection: testing HyperGAN on notMNIST, and 5 classes of CIFAR-10 which are hidden during training. We also examine adversarial examples as extreme cases of off-manifold data, and test our robustness to them.

In all experiments we report results with two HyperGANs, one trained on MNIST and another on CIFAR-10. Both of our models take a 256 dimensional noise vector as input, but have different sized latent spaces. The HyperGAN for the MNIST experiments consists of three weight generators, each using a 128 dimensional vector as input. Our HyperGAN trained on CIFAR-10 used 5 weight generators and latent points with dimensonality 256.

Table 1: MNIST HyperGAN Target Size

Layer	Latent size	Output Layer Size
Conv 1	128 x 1	32 x 1 x 5 x 5
Conv 2	128 x 1	32 x 32 x 5 x 5
Linear	128 x 1	512 x 10

Table 2: CIFAR-10 HyperGAN Target Size

Layer	Latent Size	Output Layer Size
Conv 1	256 x 1	16 x 3 x 3 x 3
Conv 2	256 x 1	32 x 16 x 3 x 3
Conv 3	256 x 1	32 x 64 x 3 x 3
Linear 1	256 x 1	256 x 128
Linear 2	256 x 1	128 x 10

HyperGAN Details

For our HyperGAN network architectures we use 2 layer MLPs with 512 units each and exponential rectifier activations (Clevert et al., 2015) for the encoder, weight generators, and discriminator. We found in a pilot study that larger networks in fact offered little performance benefit, and ultimately hurt scalability. In all experiments, we pretrain the encoder so that the mean and covariance of Q_z match $\mathcal{P}_z$. It should be noted that HyperGAN is flexible with respect to the exact architecture. The number of layers or the nonlinearity may be varied without harming HyperGANs ability to model the target distribution. We trained our HyperGAN on MNIST using less than 1.5GB of memory on a single GPU, while CIFAR-10 used just 4GB, making HyperGAN surprisingly scalable.

In Table 3 we show some statistics of the networks generated by HyperGAN on MNIST. We note that HyperGAN can generate very diverse networks, as the variance of network parameters generated by the HyperGAN is significantly higher than standard training from different random initializations.

	HyperGAN			Standard Training		
	Conv1	Conv2	Linear	Conv1	Conv2	Linear
Mean	7.49	51.10	22.01	27.05	160.51	5.97
σ^2	1.59	10.62	6.01	0.31	0.51	0.06

Table 3: 2-norm statistics on the layers of a population of networks sampled from HyperGAN, compared to 10 standard networks trained from different random initializations. Both HyperGAN and the standard models were trained on MNIST to 99% accuracy. Its easy to see that HyperGAN generates far more diverse networks

4.2 Classification

First we evaluate the classification accuracy of HyperGAN on MNIST and CIFAR-10. Classification serves as an entrance exam into our other experiments, as the distribution we want to learn is over parameters which can effectively solve the classification task. We test with both single network samples, and ensembles. For our ensembles we average predictions from N sampled models with the scoring rule $p(y|x) = \frac{1}{N}\sum_{n=0}^{N} p_n(y \mid x, \theta_n)$. Our target network for the MNIST experiments is a small two layer convolutional network, using leaky ReLU activations and 2x2 max pooling after each convolutional layer. Our target architecture for CIFAR-10 tests consists of three convolutional layers, each followed by leaky ReLU and 2x2 max pooling. The sizes of each layer can be found in tables 2 and 1. It should be noted that we did not perform fine tuning, or any additional training on the sampled networks. The results are shown in Table 4. We generate ensembles of different sizes and compare against both Bayesian (Louizos & Welling, 2016) (Krueger et al., 2017) and non-Bayesian (Lakshminarayanan et al., 2016) methods, as well as MC dropout Gal & Ghahramani (2015). We outperform all other methods by using a 100 network ensemble, across all datasets.

4.3 1-D Toy Regression Task

We next evaluate the ability of HyperGAN to fit a simple 1D function from noisy samples. This dataset was first proposed by (Hernández-Lobato & Adams, 2015), and consists of a training set of 20 points drawn uniformly from the interval $[-4, 4]$. The targets are given by $y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 3^2)$. We used the same target architecture as in (Hernández-Lobato & Adams, 2015) Lakshminarayanan et al. (2016) and (Louizos & Welling, 2016): a one layer neural network with 100 hidden units and ReLU nonlinearity. For HyperGAN we use two layer generators, and 128

Method	MNIST	MNIST 5000	CIFAR-5	CIFAR-10	CIFAR-10 5000
1 network	98.64	96.69	84.50	76.32	76.31
5 networks	98.75	97.24	85.51	76.84	76.41
10 networks	99.22	97.33	85.54	77.52	77.12
100 networks	**99.31**	**97.71**	**85.81**	**77.71**	**77.38**
Deep Ensembles	99.30		79.00		
MNFG	99.30		84.00		
BHN	98.63	96.51		74.90	
MC Dropout	98.73	95.58	84.00	72.75	

Table 4: Classification performance of HyperGAN on MNIST and CIFAR-10. In order to compare against MNF and Deep Ensembles, we also train a HyperGAN on only the first 5 classes of CIFAR-10, which we denote as CIFAR-5. In addition we examine our generalization ability by training on only 5000 examples of MNIST and CIFAR-10 with a small target network. We do not attempt to outperform state of the art, but we perform better than other probabilistic neural network approaches

hidden units across all networks. Because this is a small task, we use only a 64 dimensional latent space. MSE loss is used as our target loss function to train HyperGAN.

Results in figure 2 show that HyperGAN clearly learns the target function and captures the variation in the data well. In addition, it can be seen that sampling more networks to compose a larger ensemble improves predictive uncertainty as we sample farther from the mean of the training data.

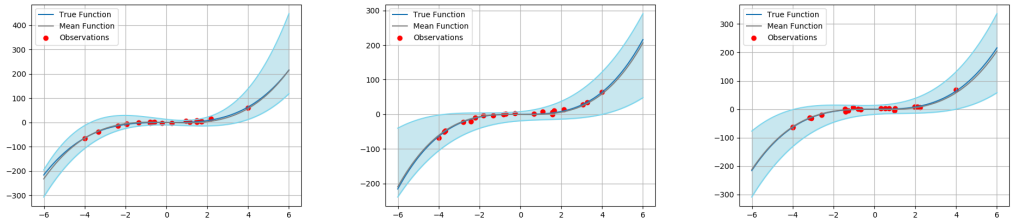


Figure 2: Results of HyperGAN on the 1D regression task. From left to right, we plot the predictive distribution of 10, 100, and 1000 sampled models from a trained HyperGAN. Within each image, the blue line is the target function x^3, the red circles show the noisy observations, the grey line is the learned mean function, and the light blue shaded region denotes ± 3 standard deviations

4.4 Anomaly Detection

To test our uncertainty measurements, we perform the same experiments as (Louizos & Welling, 2016), (Lakshminarayanan et al., 2016); we measure the total entropy in predictions from HyperGAN-generated networks. For MNIST experiments we train a HyperGAN on the MNIST dataset, and test on out-of-distribution notMNIST, which consists of 28x28 binary images of letters. In this setting, we want the softmax probabilities on inlier MNIST examples to have maximum entropy - a single large activation close to 1. On off-manifold data we want to have equal probability across predictions. We test our CIFAR-10 model by just training on the first 5 classes, and we use the latter 5 classes as out of distribution examples. To build an estimate of the predictive entropy we sample multiple networks from HyperGAN per example, and measure their predictive entropy.

In Fig. 3 we show that the CIFAR-10 inlier and outlier examples are well separated. HyperGAN learns to be less certain about data it does not recognize, as the probability of a low entropy prediction is overall lower on outliers. On notMNIST we also show separation, though HyperGAN is also overall less confidant about inliers. Conventionally trained ensembles without the HyperGAN, referred to as L2 networks in the figure, are highly overconfident on outliers and cannot provide a notion of uncertainty. We have asked authors of (Louizos & Welling, 2016) and Lakshminarayanan et al. (2016) for exact values in their paper for those figures that we will include in the final draft. For now we refer the reader to their papers, but note that we never had any outlier with an entropy less than 0.8 while all theirs have significant portions of outliers having entropies less than 0.8.

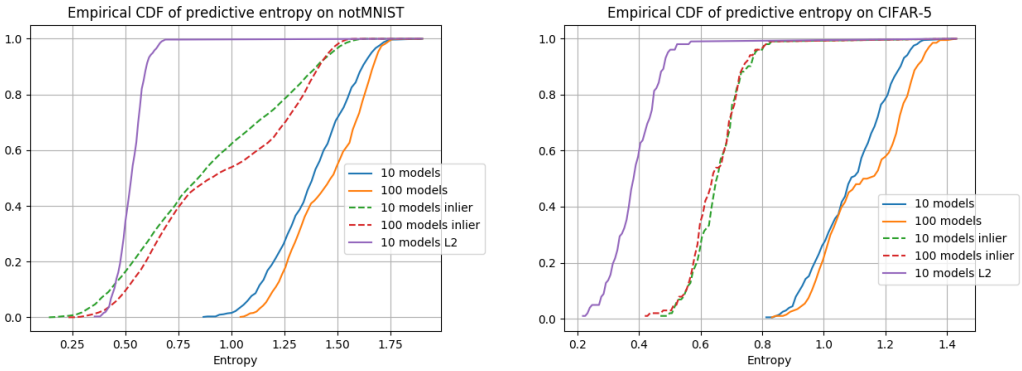


Figure 3: Empirical CDF of the predictive entropy on out of distribution datasets notMNIST, and 5 classes of CIFAR-10 unseen during training. Solid lines denote tests on the respective out of distribution data, while the dashed lines denote entropy on inlier examples (MNIST and CIFAR-10). L2 referred to conventional ensembles trained separately without a HyperGAN

4.5 Adversarial Detection

We employ the same experimental setup to the detection of adversarial examples, an extreme sort of off-manifold data. Adversarial examples are often optimized to lie within a small neighborhood of a classifier's decision boundaries. They are created by adding perturbations in the direction of the greatest loss with respect to the model's parameters. Because HyperGAN learns a distribution over parameters, it should be more robust to attacks. We generate adversarial examples using the Fast Gradient Sign method (FGSM) (Goodfellow et al., 2014) and Projected Gradient Descent (PGD) (Madry et al., 2017). FGSM adds a small perturbation ϵ to the target image in the direction of greatest loss. FGSM is known to underfit to the target model, hence it may transfer well across many similar models. In contrast, PGD takes many steps in the direction of greatest loss, producing a stronger adversarial example, at the risk of overfitting to a single set of parameters. This poses the following challenge: to detect attacks by FGSM and PGD, HyperGAN will need to generate diverse parameters to avoid both attacks. To detect adversarial examples, we first hypothesize that a single

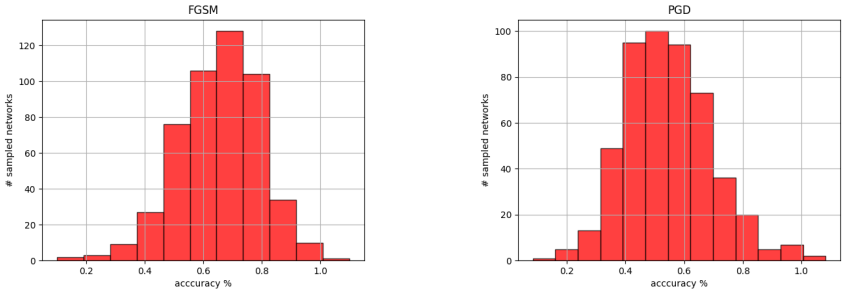


Figure 4: Diversity of predictions on adversarial examples. FGSM and PGD examples are created against a network generated by HyperGAN, and tested on 500 more generated networks. FGSM transfers better than PGD, though both attacks fail to cover the distribution learned by HyperGAN

adversarial example will not fool the entire space of parameters learned by HyperGAN. If we then evaluate adversarial examples against many generated networks, then we should see a high level of disagreement among predictions for any individual class. In this case, we define disagreement as a function of entropy in the softmax probabilities at the output of the network. Given the unnormalized outputs x of N models, we compute the disagreement d across K classes:

$$d = -\sum_i p_i \log p_i \quad \text{where} \quad p_i = N^{-1} \sum_{n=1}^{N} \frac{\exp f(x)_i}{\sum_{k=1}^{K} \exp f(x)_k}$$

where $f(x)_i$ refers to the logits of class i.

Adversarial examples have been shown to successfully fool ensembles (Dong et al., 2017), but with HyperGAN one can always generate significantly more models that can be added to the ensemble for the cost of one forward pass, making it hard to attack against. In Fig. 5 we test HyperGAN against adversarial examples generated to fool one network. It is shown that while those examples can fool $50\% - 70\%$ of the networks generated by HyperGAN, they usually do not fool all of them.

We compare the performance of HyperGAN with ensembles of $N \in \{5, 10\}$ models trained on MNIST with normal supervised training. We fuse their logits (unnormalized log probabilities) together as $l(x) = \sum_{n=0}^{N} w_n l_n(x)$ where w_n is the nth model weighting, and l_n is the logits of the nth model. In all experiments we consider uniformly weighted ensembles. For HyperGAN we simply sample from parameter space to create as many models as we need, and similarly fuse their logits together. Specifically we test ensembles with $N \in \{5, 10, 100, 1000\}$ members each. Here adversarial examples are generated by attacking the ensemble directly. For HyperGAN, we attack an ensemble of networks, but test with a new ensemble of equal size.

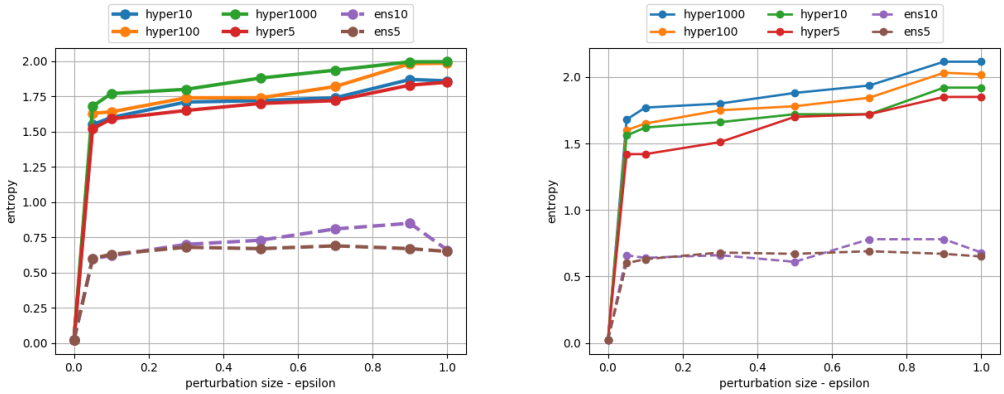


Figure 5: Entropy of predictions on FGSM and PGD adversarial examples. HyperGAN generates ensembles that are far more effective than standard ensembles even with equal population size. Note that for large ensembles, it is hard to find adversarial examples with small norms e.g. $\epsilon = 0.01$

For the purposes of detection, we compute the entropy within the predictive distribution of each of the ensemble members to score the example on the likelihood that it was drawn from the training distribution. Figure 5 shows that HyperGAN easily identifies adversarial examples as being out-of-distribution. HyperGAN is especially suited to this task as adversarial examples are optimized against parameters - parameters which HyperGAN can change. We find that we can successfully detect over 97% of adversarial examples, with a low false positive rate for both attacks just by thresholding the entropy.

5 Discussion and Future Directions

We have proposed a generative, non-Bayesian solution to parameter generation which performs strongly on detecting out-of-distribution samples, as well as classification. Training a GAN to learn a probability distribution over parameters allows us to non-deterministically sample diverse, performant networks which we can use to form ensembles that can give good uncertainty estimates. Our method is ultimately scalable to any number of networks in the predicting ensemble, requiring just one forward pass to generate a new set of parameters and a low GPU memory footprint. We showed that we can generate models with significant variation over the learned distribution and thus provide uncertainty estimates on outlier data. Our HyperGAN can be readily extended to generate parameters for a variety of architectures such as MLPs, CNNs, etc. We hope that this will encourage the community to consider other generative approaches to learning the manifold of neural networks. There is still much room for exploration, we believe that learning a low dimensional manifold of performant neural networks could be useful for a variety of domains including meta learning and reinforcement learning. In the future we wish to explore agent curiosity and exploration policies aided by uncertainty measurements from HyperGAN, or explore transfer learning by learning a manifold of neural networks which can solve more than one task. We will strive to make the code available as soon as possible.

REFERENCES

D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. A Closer Look at Memorization in Deep Networks. *ArXiv e-prints*, June 2017.

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. *CoRR*, abs/1605.09673, 2016. URL `http://arxiv.org/abs/1605.09673`.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. URL `http://arxiv.org/abs/1511.07289`.

Andreas Damianou and Neil Lawrence. Deep Gaussian processes. In C. Carvalho and P. Ravikumar (eds.), *Proceedings of the Sixteenth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, AISTATS '13, pp. 207–215. JMLR W&CP 31, 2013.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Discovering adversarial examples with momentum. *CoRR*, abs/1710.06081, 2017. URL `http://arxiv.org/abs/1710.06081`.

C. D. Freeman and J. Bruna. Topology and Geometry of Half-Rectified Network Optimization. *ArXiv e-prints*, November 2016.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *ArXiv e-prints*, June 2015.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *ArXiv e-prints*, December 2014.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016.

J. M. Hernández-Lobato and R. P. Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. *ArXiv e-prints*, February 2015.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. URL `http://arxiv.org/abs/1506.02025`.

D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian Hypernetworks. *ArXiv e-prints*, October 2017.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *ArXiv e-prints*, December 2016.

Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *CoRR*, abs/1802.09419, 2018.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *CoRR*, abs/1605.09673, 2016. URL `http://arxiv.org/abs/1605.09673`.

Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *CoRR*, abs/1106.0257, 2011.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *ArXiv e-prints*, June 2017.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015. URL `http://arxiv.org/abs/1511.05644`.

Tsendsuren Munkhdalai and Hong Yu. Meta networks. *CoRR*, abs/1703.00837, 2017.

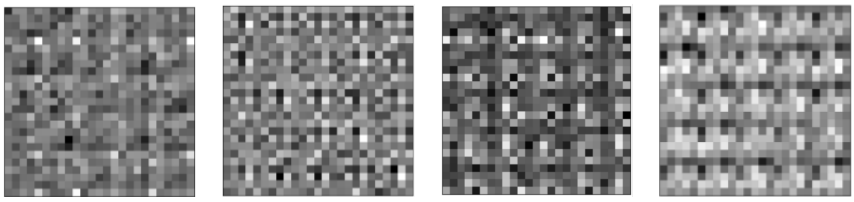
Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein Auto-Encoders. *ArXiv e-prints*, November 2017.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Deep image prior. *CoRR*, abs/1711.10925, 2017.

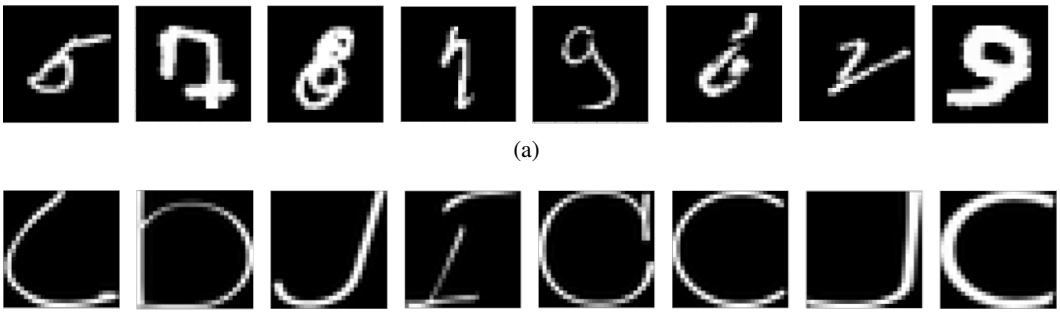
A APPENDIX

We show the first filter in 25 differernt networks generated by the HyperGAN to illustrate their difference in Fig. 7. It can be seen that qualitatively HyperGAN learns to generate classifiers with a variety of filters.



(a) (b) (c) (d)

Figure 6: Convolutional filters from MNIST classifiers sampled from HyperGAN. For each image we sample the same 5x5 filter from 25 separate generated networks. From left to right: figures a and b show the first samples of the first two generated filters for layer 1 respectively. Figures c and d show samples of filters 1 and 2 for layer 2. We can see that qualitatively, HyperGAN learns to generate classifiers with a variety of filters.



(a)

(b)

Figure 7: Images of examples which do not behave like most of their respective distribution. On top are MNIST images which HyperGAN networks predict to have high entropy. We can see that they are generally ambiguous and do not fit with the rest of the training data. The bottom row shows notMNIST examples which score with low entropy according to HyperGAN. It can be seen that these examples look like they could come from the MNIST training distribution, making HyperGAN's predictions reasonable