Adversarial Distillation of Bayesian Neural Network Posteriors

Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, Richard Zemel

ICML 2018

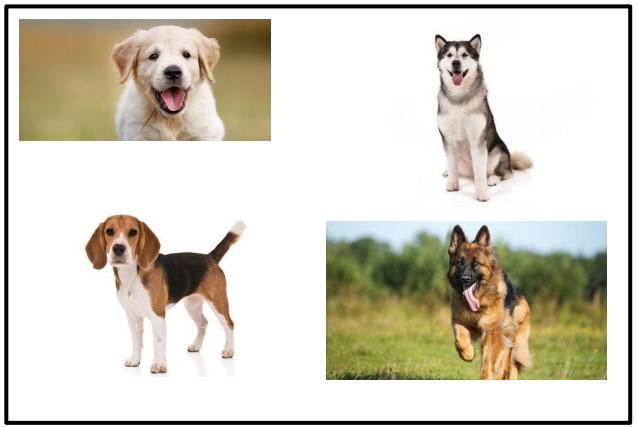
Slides by: Paul Vicol

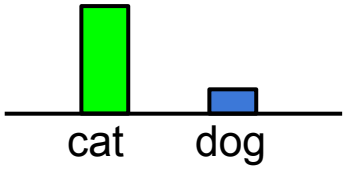
UNIVERSITY OF TORONTO

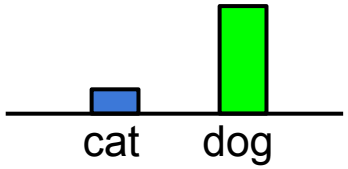
VECTOR INSTITUTE

Motivation: Why do we need uncertainty?

Training

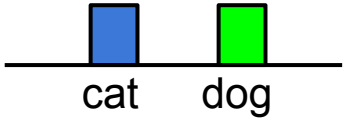






Test

- Try to extrapolate
- Indicate that the example is out-of-distribution



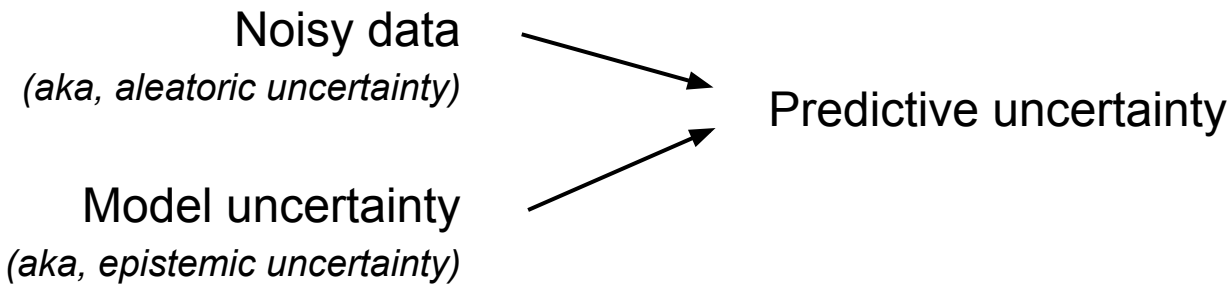
Motivation: Why do we need uncertainty?

- **Safety-critical systems**: want to have high confidence before taking action; otherwise defer to a human

- **Medical applications**: uncertainty is critical in automatic diagnosis systems

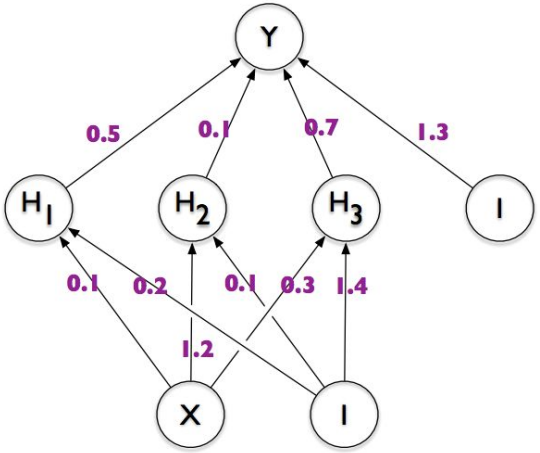
- Using uncertainty to **understand systems**
  - A basketball player's offensive skill can be judged/estimated by the amount of uncertainty he can induce in his movements. Here, *more uncertainty = better!*

Motivation: Why do we need uncertainty?



- Bayesian methods provide a *principled way to capture model uncertainty* through the posterior distribution $p(\theta|\mathcal{D})$ over model parameters
- Can reason about how different models from the posterior behave as a group
- Real-world applications of Bayesian methods (e.g., BNNs) include:
  - Efficient exploration in RL (Vlassis et al., 2012)
  - Active learning (Settles, 2010)
  - Defense against adversarial attacks (Feinman et al., 2017)

Standard Neural Net



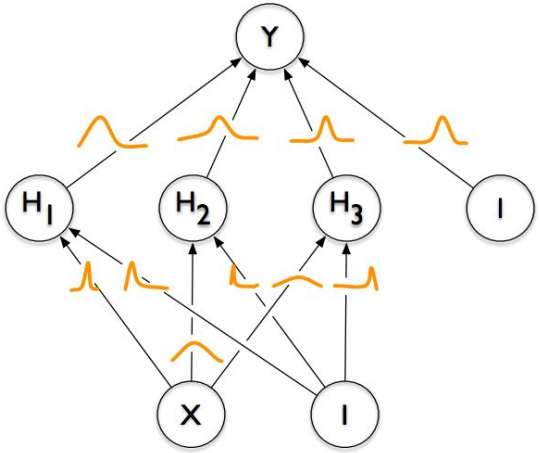
- Learn the parameters θ by minimizing a loss function $\mathcal{L}$

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta, \mathcal{D})$$

- Depending on the loss function, θ^* can be the MLE or MAP point-estimate
- Make predictions by computing

$$y = f_{\theta^*}(x)$$

Bayesian Neural Net



- Parameters represented by *distributions*
- Find $p(\theta|\mathcal{D})$ through *VI or MCMC methods*
- Make predictions by integrating over θ

$$p(y|x, \mathcal{D}) = \int p(y|\theta, x)p(\theta|\mathcal{D})d\theta$$

Motivation - Conventional Training == Approximation

Minimizing:

NLL (no regularization)

$$\frac{1}{N}\sum_{i=1}^{N} -\log p(y_i|x_i,\theta)$$

Is Equivalent To:

Maximum likelihood estimation

$$\theta^{MLE} = \arg\max_{\theta} p(\mathcal{D}|\theta)$$

NLL (+ L^2 regularization)

$$\frac{1}{N}\sum_{i=1}^{N} -\log p(y_i|x_i,\theta) + \lambda||\theta||_2^2$$

MAP estimation with a Gaussian prior

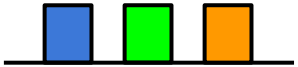
$$\theta^{MAP} = \arg\max_{\theta} p(\theta|\mathcal{D})$$

where $\theta \sim \mathcal{N}(0,\sigma^2)$

Uncertainty Measures

Predictive Entropy

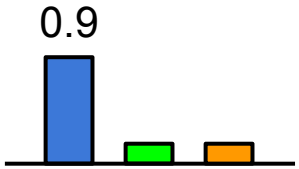
$$H(y|x,\mathcal{D}) = -\sum_c p(y=c|x,\mathcal{D}) \log p(y=c|x,\mathcal{D})$$



Max entropy

Variation Ratios (VR)

$$\mathrm{VR}[x] = 1 - \max_y p(y|x,\mathcal{D})$$



VR = 0.1

Bayesian Active Learning by Disagreement (BALD)

$$\mathbb{I}(y,\theta|x,\mathcal{D}) = H(y|x,\mathcal{D}) - \mathbb{E}_{\theta\sim p(\theta|\mathcal{D})}[H(y|x,\theta)]$$

Large when...

The model is uncertain about y...

But individual settings of the parameters are confident about y

Two Ways to Compute the Posterior

Variational Inference (VI)

- Introduce a parametric distribution $q_\phi(\theta)$ and minimize the KL divergence:

$$\phi^* = \arg\min_\phi \mathrm{KL}[q_\phi(\theta)||p(\theta|\mathcal{D})]$$

- VI *can* only produce samples from the *approximate posterior*
- Most VI approaches make strong assumptions about the structure of the posterior: assume that the posterior distribution factorizes as the product of univariate Gaussians
  - (Uses *~2x memory* to store a mean and variance for each parameter)

MCMC Methods

- In the limit, MCMC methods yield samples from the *true posterior* $p(\theta|\mathcal{D})$
- No assumption of factorizing posterior, but are not scalable to large datasets, because they require computation over the whole dataset
- SGLD and other sg-MCMC methods address the computational cost by operating on mini-batches

Stochastic Gradient Langevin Dynamics (SGLD)

SGLD bridges **optimization** and **Bayesian learning**

Transitions from optimization *to posterior sampling*

Mini Batch GD (SGD)

$$\Delta\theta^t = \frac{\epsilon^t}{2}\left(\nabla \log p(\theta^t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(y_i^t|x_i^t, \theta^t)\right)$$

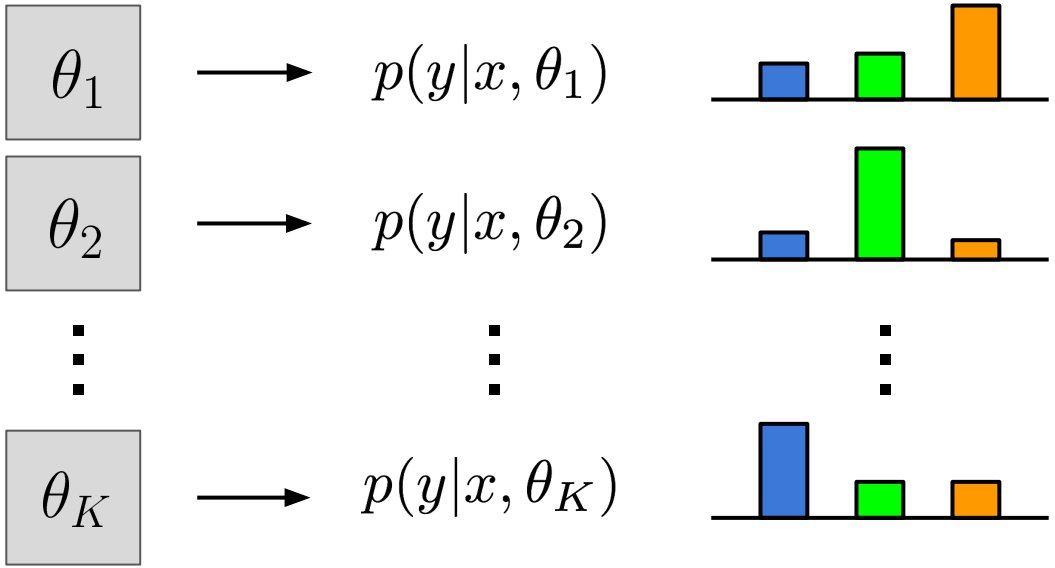
SGLD

$$\Delta\theta^t = \frac{\epsilon^t}{2}\left(\nabla \log p(\theta^t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(y_i^t|x_i^t, \theta^t)\right) + \eta^t$$

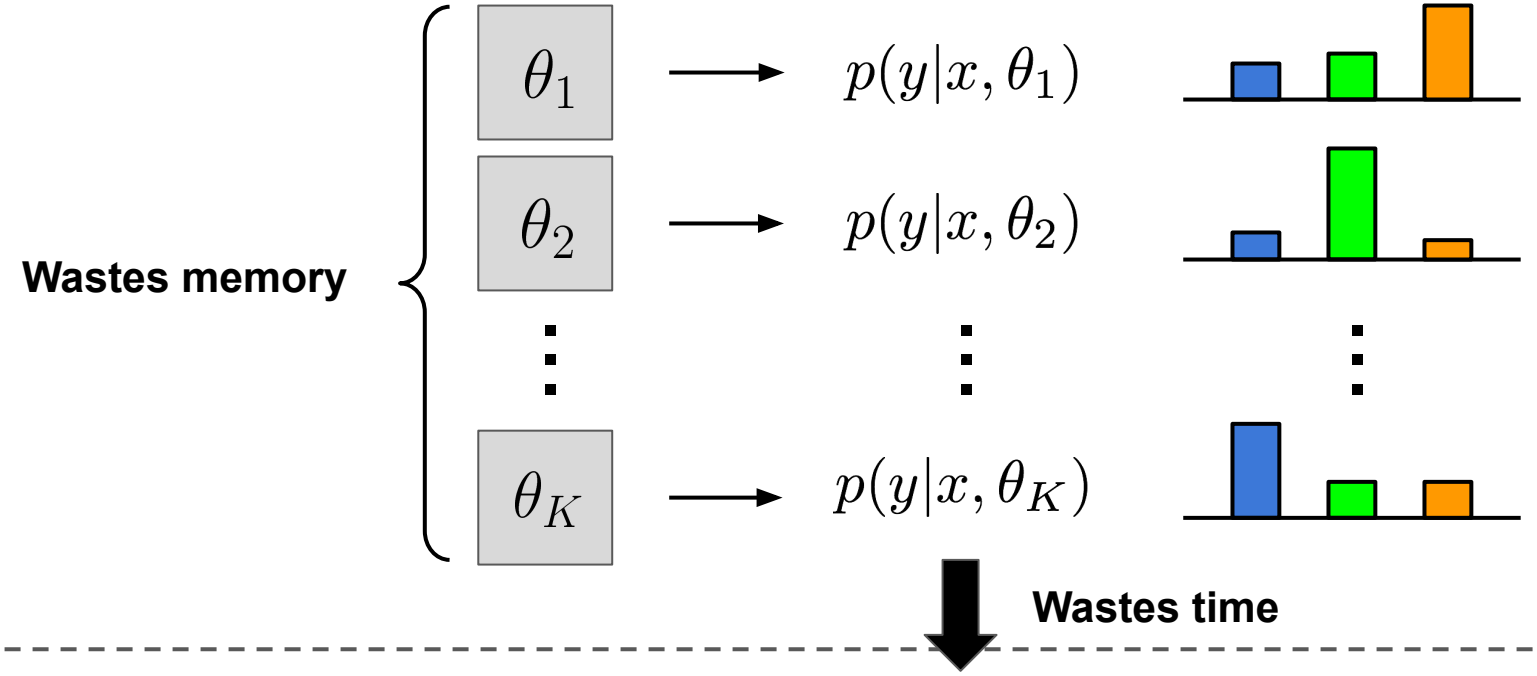
$$\eta^t \sim \mathcal{N}(0, \epsilon^t)$$

- Obtain samples $\theta \sim p(\theta|\mathcal{D})$ by adding Gaussian noise to SGD updates
- Save the iterates $\{\theta^t\}$ while training the model - simultaneous training & posterior sampling

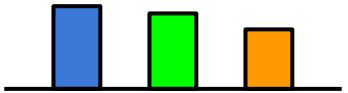
Problems with SGLD



Problems with SGLD



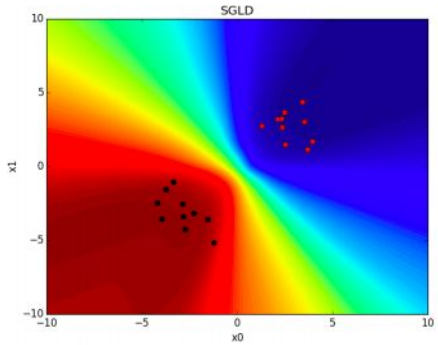
$$p(y|x,\mathcal{D}) \approx \frac{1}{K}\sum_{k=1}^{K} p(y|x,\theta_k)$$

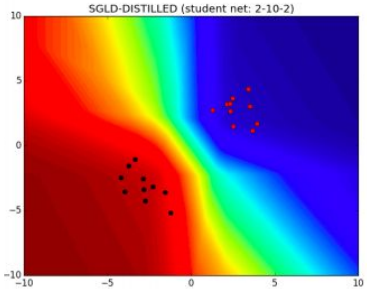


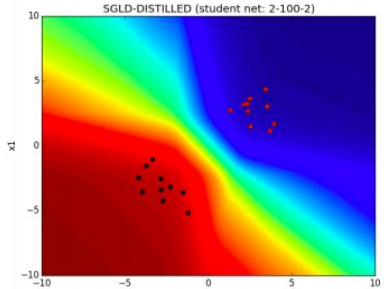
Related Work: Bayesian Dark Knowledge

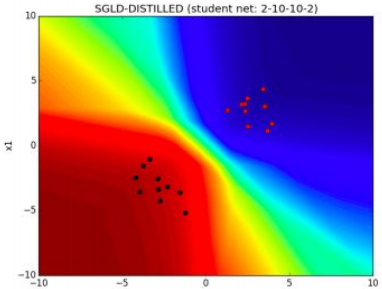
- Train a student network to approximate the posterior predictive distribution of the teacher (= Monte Carlo ensemble)
- Denote the prediction of the teacher by $\mathbb{E}_{p(\theta|\mathcal{D})}[p(y|x,\theta)]$

Goal: minimize $\mathrm{KL}[\mathbb{E}_{p(\theta|\mathcal{D})}[p(y|x,\theta)]||\mathcal{S}(y|x,w)]$









SGLD predictive distribution

Various student network architectures

Related Work: Bayesian Dark Knowledge

- Bayesian Dark Knowledge distills the posterior predictive produced by SGLD into a single network
  - \+ Saves computation time (no integrating over posterior samples)
  - \+ Saves storage (no need to keep posterior samples around)
  - \- **The posterior is lost at test-time**

Cannot compute quantities like BALD:

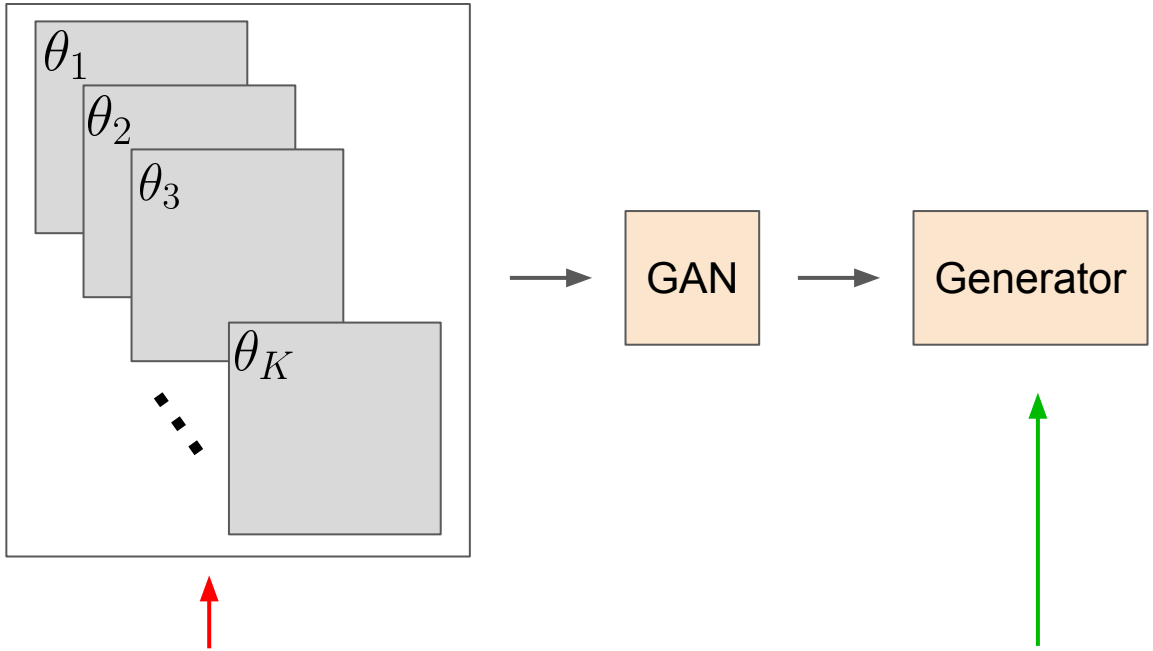
$$\mathbb{I}(y, \theta | x, \mathcal{D}) = H(y|x, \mathcal{D}) - \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}[H(y|x, \theta)]$$

Requires access to the posterior distribution

- Our approach distills the posterior distribution such that we can draw samples from it at test-time

Distilling Posterior Samples using a GAN

$\theta \sim p(\theta|\mathcal{D})$ Model parameters saved during training

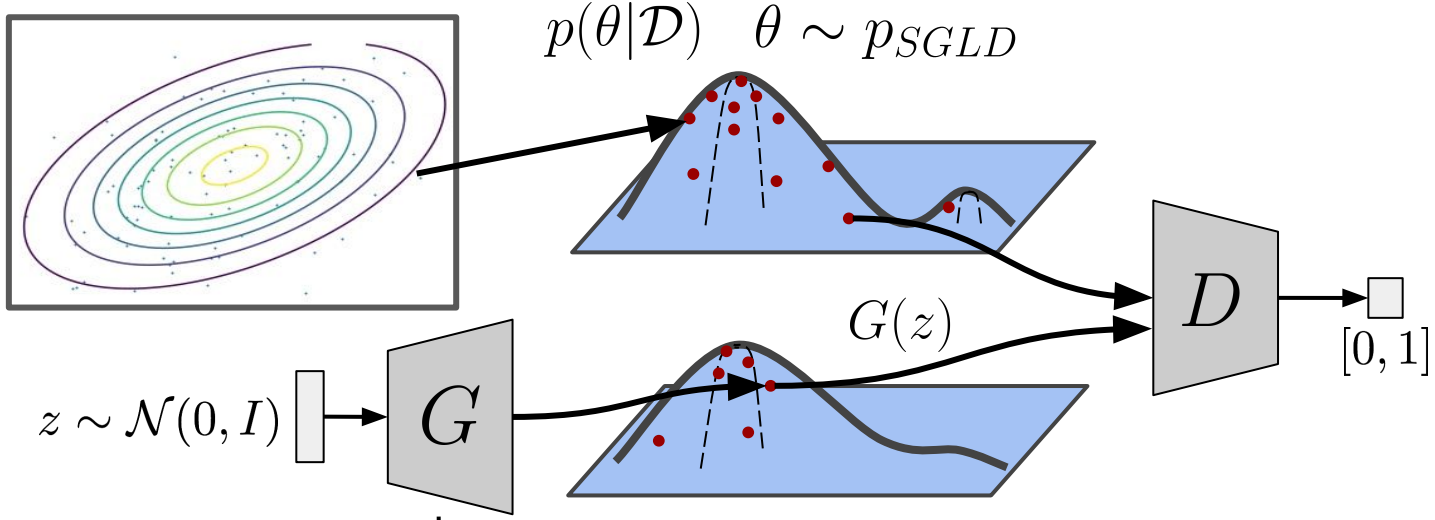


More samples = more storage

Fixed storage cost for any # samples

Adversarial Posterior Distillation (APD)

Distillation



Inference

SGLD

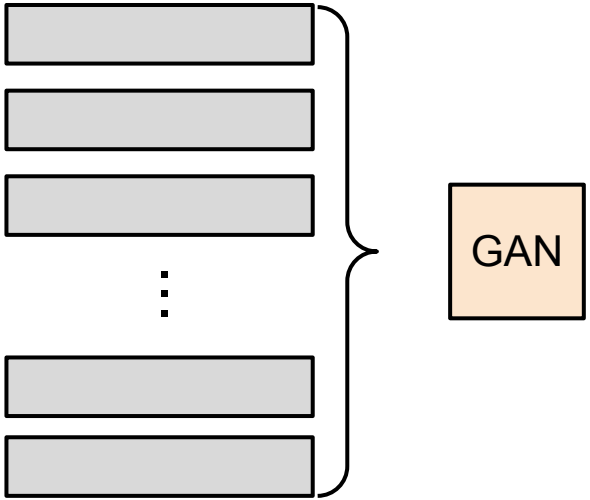
$$p(y|x, \mathcal{D}) = \frac{1}{T} \sum_t p(y|x, \theta^t), \theta^t \sim p(\theta|\mathcal{D})$$

APD (Ours)

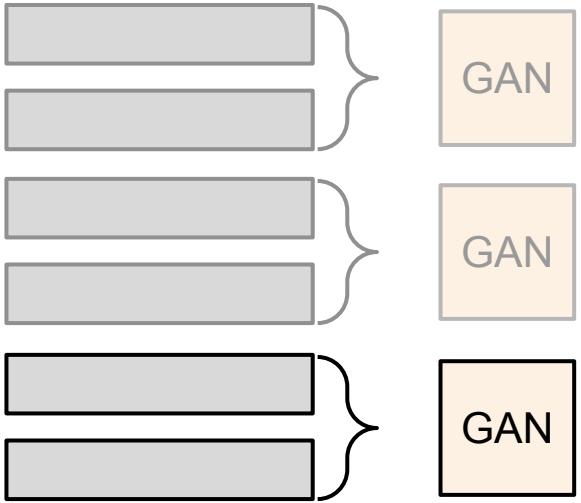
$$p(y|x, \mathcal{D}) = \frac{1}{T} \sum_t p(y|x, G(z^t)), z^t \sim \mathcal{N}(0, I)$$

Offline and Online APD

Offline APD



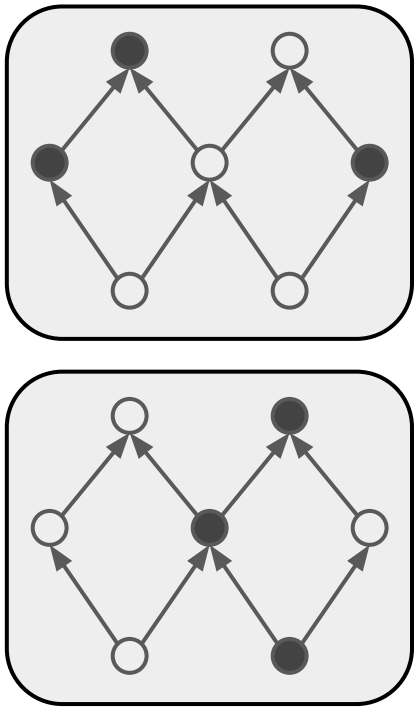
Online APD



- Two distinct stages:
  1. Obtain posterior samples $\{\theta^t\}_{t=1}^T$
  2. Train a GAN on those samples

- Alternate between posterior sampling and GAN training

Baseline: MC Dropout

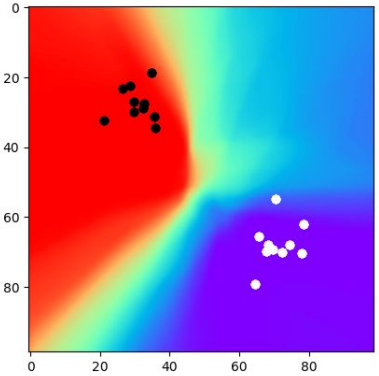


- **Monte Carlo dropout** = applying dropout at test-time to obtain predictions made by an ensemble of models

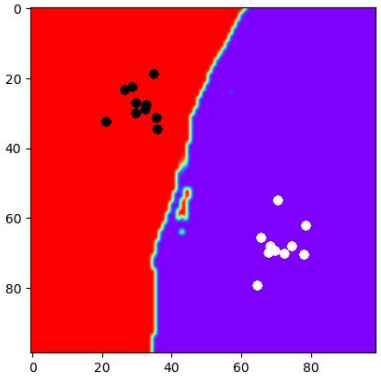
\+ Simple way to obtain uncertainty estimates; does not require any additional storage; just the standard point-estimate model

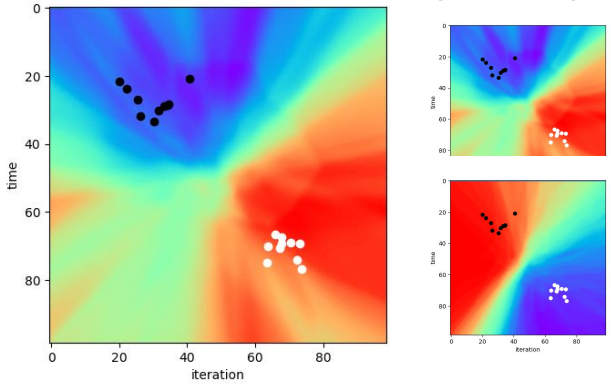
Experiments: Toy 2D Classification

SGLD Samples

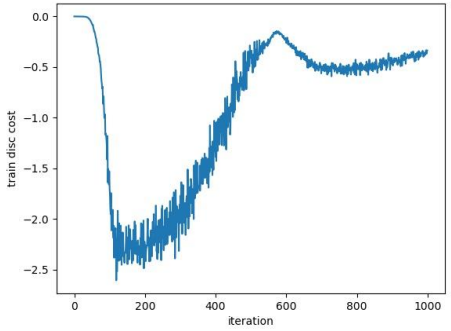


Non-Bayesian



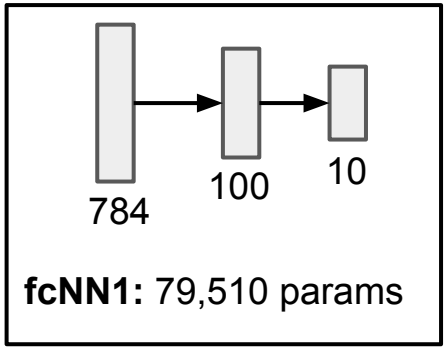


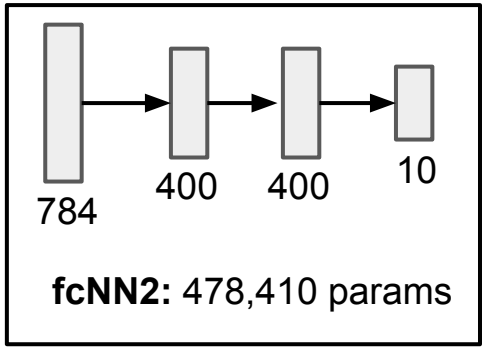
WGAN-GP loss



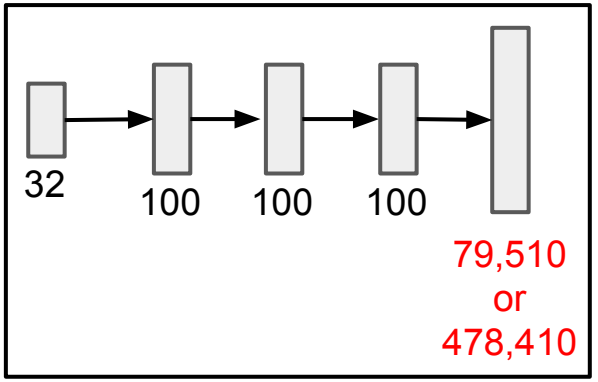
Predictive Performance and Uncertainty

Two fully-connected architectures





Generator/discriminator architecture

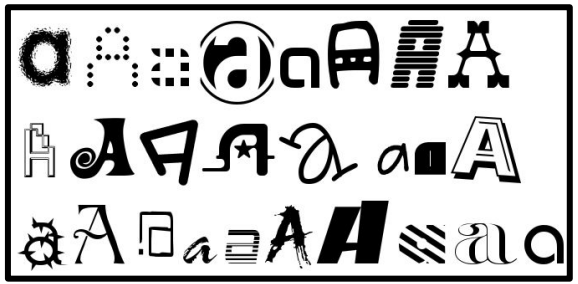


- Classification accuracy on MNIST

Dataset	Model	SGD	MC-Drop	SGLD	APD (Ours)
MNIST	fcNN1	0.981	0.973	0.979	0.978
MNIST	fcNN2	0.981	0.983	0.980	0.981

Anomaly Detection

- **Anomaly detection:** detecting out-of-distribution (OOD) data given a BNN trained on in-distribution data
- Should be more uncertain about OOD data (e.g., when trying to classify an OOD example into one of K classes that all correspond to in-distribution data)



notMNIST

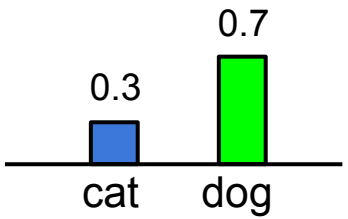
Omniglot

CIFAR-10 bw

Anomaly Detection

- Method:
  1. Train a classifier on MNIST
  2. At test-time, input 50% MNIST data and 50% OOD data

- Uncertainty indicates whether an example is OOD - how much uncertainty is enough?



$$\mathrm{VR}(x) = 1 - \max_y p(y|x, \mathcal{D}) = 0.3$$

- We report performance using the *area under the receiver operating characteristic curve* (AU-ROC), which is a *threshold-independent measure* (based on TPR and FPR)

Anomaly Detection Results

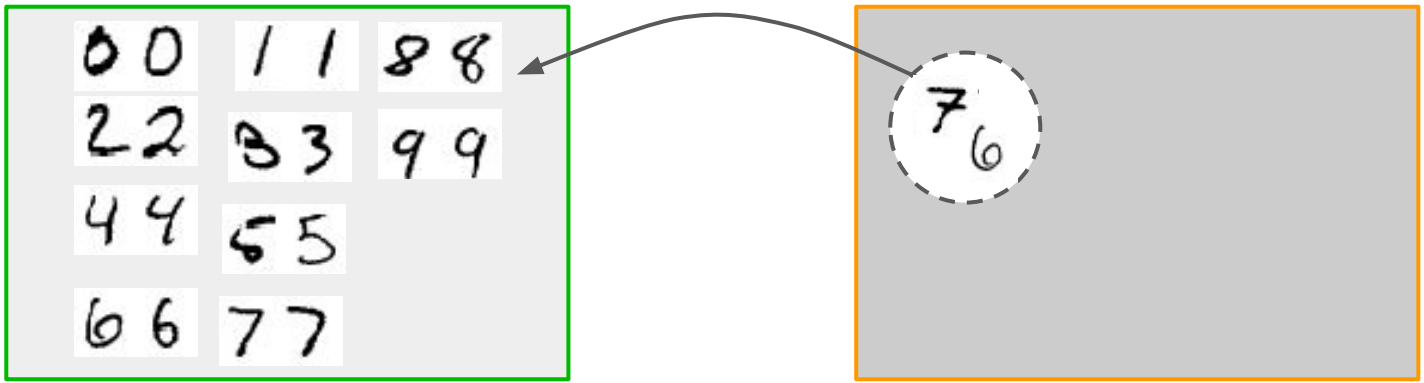
	Dataset	SGD			MC-Dropout			SGLD			APD (Ours)		
Det.	**area under**	**ROC**	**PR+**	**PR-**	**ROC**	**PR+**	**PR-**	**ROC**	**PR+**	**PR-**	**ROC**	**PR+**	**PR-**
VR	**notMNIST**	55.4	60.9	48.3	88.0	87.2	82.1	98.1	97.8	98.3	97.8	97.4	98.1
	OmniGlot	85.0	85.5	79.6	91.5	90.8	90.3	99.0	98.8	99.1	98.8	98.6	99.1
	CIFAR10bw	59.0	65.0	50.4	90.1	88.5	86.5	97.4	97.0	97.5	96.9	96.5	96.7
	Gaussian	64.3	68.0	54.6	91.3	89.8	89.0	99.6	99.6	99.7	99.6	99.5	99.6
	Uniform	81.2	79.2	80.4	93.6	91.2	94.8	99.8	99.8	99.9	99.8	99.7	99.8
BALD	**notMNIST**	-	-	-	87.0	85.0	81.0	99.7	99.8	99.6	99.6	99.7	99.5
	OmniGlot	-	-	-	91.4	90.7	90.5	99.9	100.0	99.9	99.9	99.9	99.9
	CIFAR10bw	-	-	-	89.3	86.2	86.0	99.4	99.4	99.2	99.1	99.3	98.3
	Gaussian	-	-	-	90.9	88.6	89.3	100.0	100.0	100.0	100.0	100.0	100.0
	Uniform	-	-	-	97.3	96.6	97.9	100.0	100.0	100.0	100.0	100.0	100.0

- Anomaly detection results using fcNN2 (478,410 params)
- Our method outperforms SGD and MC dropout, and *almost matches SGLD*

Experiments: Active Learning

- When labeling data is expensive, we should make sure we label informative examples

In each acquisition iteration, the model *chooses 10 images* from the pool set to have labelled (by a human or oracle)



Initial Training Set

Unlabeled Pool Set $\mathcal{D}_{pool}$

Intuitively: Choose to label the points we're most uncertain about

Experiments: Active Learning

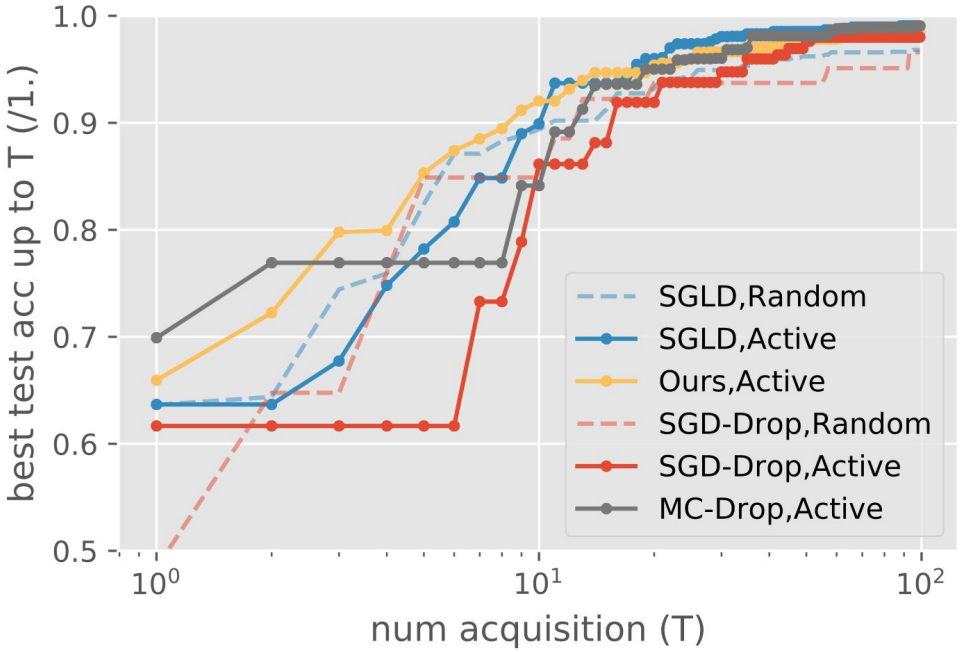
- An acquisition function $a(x)$ is used to decide *which points to query next*, e.g.,

$$x^* = \arg \max_{x \in \mathcal{D}_{pool}} a(x)$$

- Most acquisition functions are based on uncertainty, e.g., *entropy, variation ratios, BALD*

- Sort the examples in $\mathcal{D}_{pool}$ by decreasing $a(x)$
- Choose the top K to add to the training set in one iteration

Experiments: Active Learning



- Using *entropy as the acquisition function* is better than random
- BNNs outperform point-estimate counterparts consistently
- *APD performs best early, <10 acquisitions*, due to either better regularization or better uncertainty for active learning

Adversarial Detection - Method

- **Fast Gradient Sign Method (FGSM)** - Relatively easy attack to defend against

- **Projected Gradient Descent (PGD)** - Strong attack

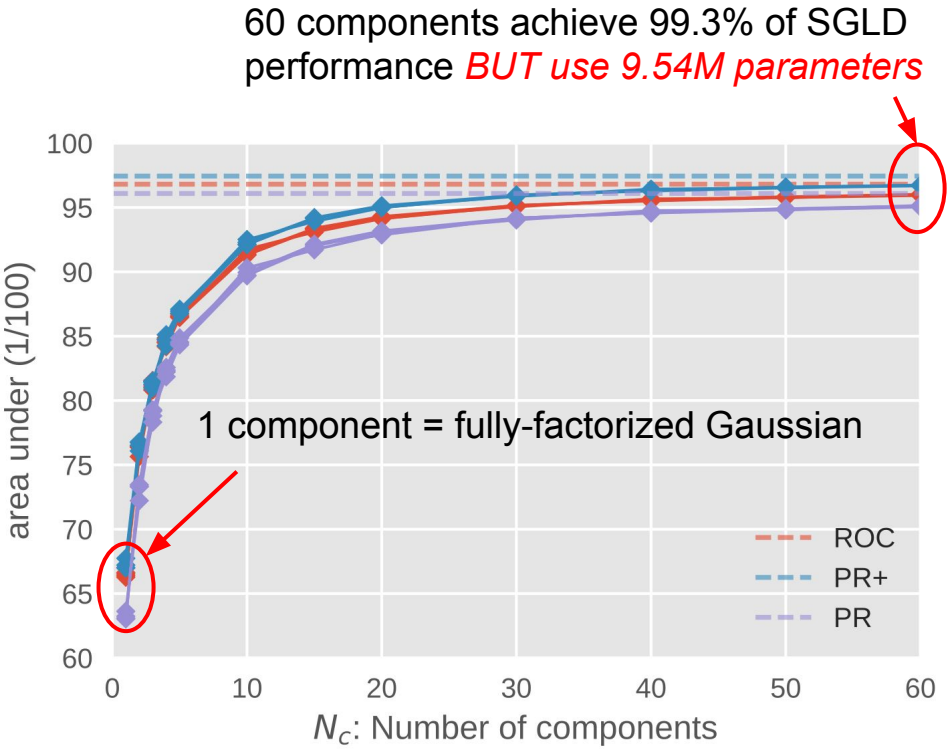
- The adversary has access to the network architecture and *a single posterior sample*
  - Gray-box attack because the adversary does not have access to the full posterior
- We generated 6000 adversarial examples from the validation set images using the single posterior as fixed network weights
- Generated attacks using samples from the source model (MC dropout, SGLD, or APD)

Adversarial Detection Results

- All approaches were effective when detecting adversarial examples *crafted with their own networks*
- When *transferring attacks* between networks:
  - MC dropout performed near random (50%)
  - SGLD and APD were able to detect transferred attacks
  - But SGLD outperformed our method on the transferred attacks

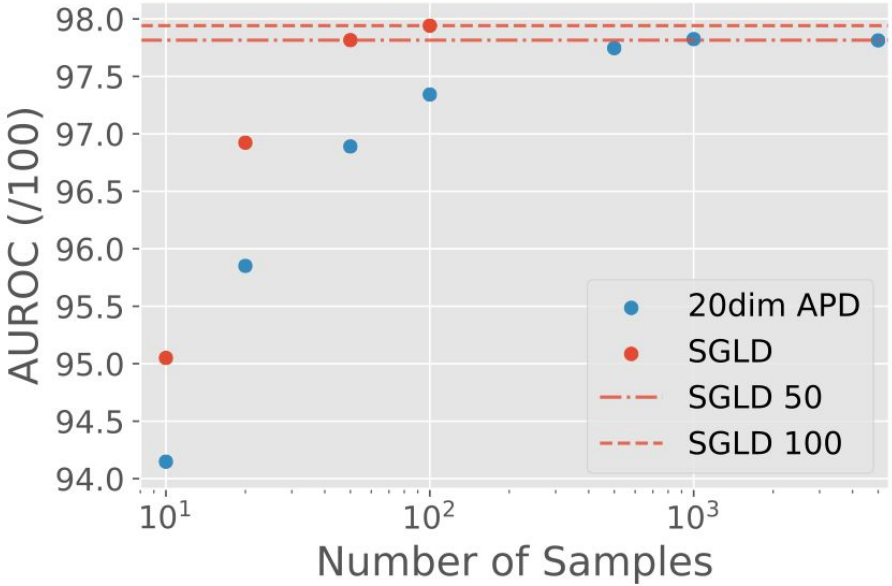
Source	Attack Type	MC-Drop	SGLD	Ours
MC-Drop	**FGSM**	89.53	**94.01**	91.70
	PGD	88.37	**93.95**	91.63
SGLD	**FGSM**	54.99	**83.76**	75.93
	PGD	56.91	**84.98**	82.80
Ours	**FGSM**	54.51	83.05	**86.02**
	PGD	54.98	88.01	**93.15**

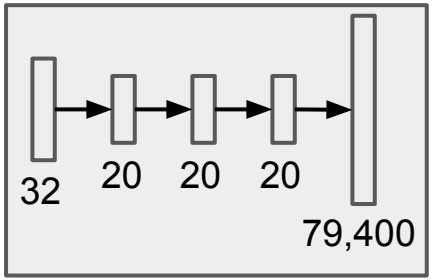
Analysis: Do we need a GAN?



- Using MCMC allows us to *avoid making simplifying assumptions* about the structure of the posterior
- Series of *mixture of Gaussians with increasing numbers of components*
  - Fit to 2000 SGLD samples
- The multimodal posterior distribution induced by SGLD samples cannot be completely represented with simple, fully-factorized approximations
- A *single mode is not sufficient* to model the posterior

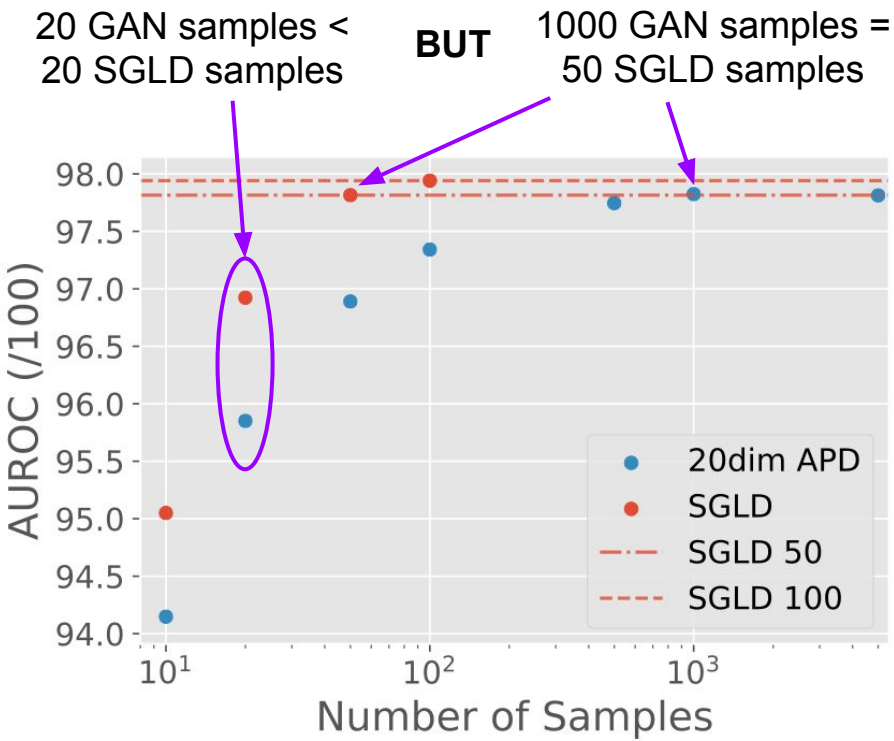
APD Storage Savings

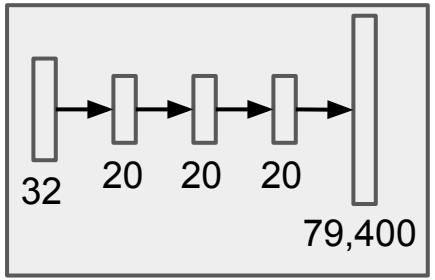




20-dim GAN

APD Storage Savings





20-dim GAN

- APD yields 2.5x storage savings compared to SGLD
- APD attains 99.8% of SGLD performance using 1.67M parameters (vs 9.54M for MoG)

A Way to Evaluate GANs?

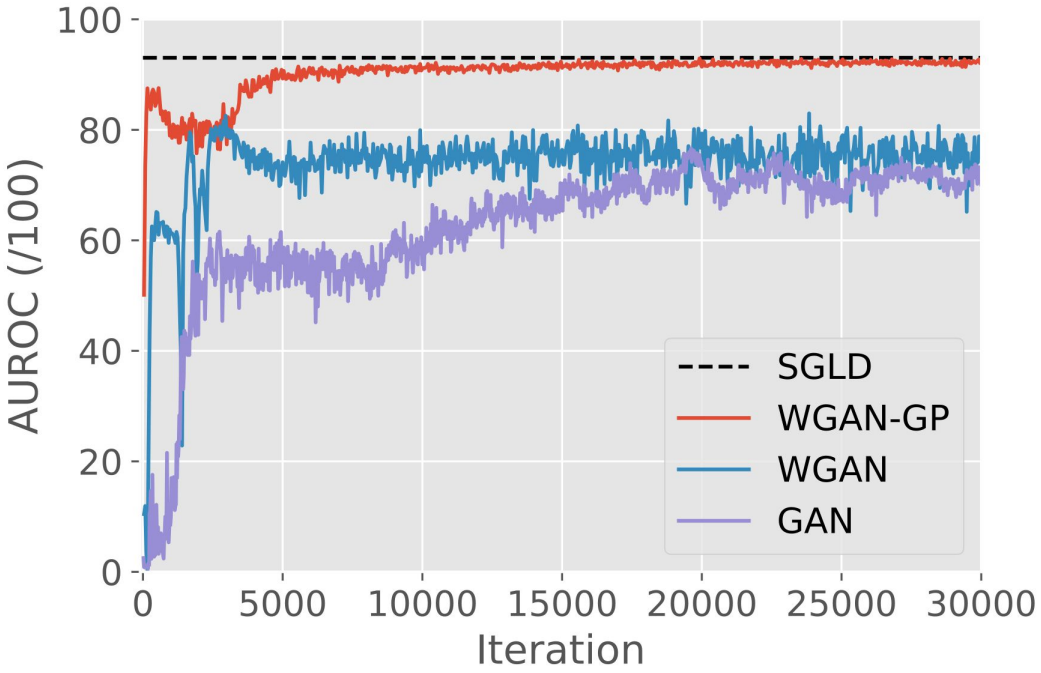
Vanilla GAN

WGAN

WGAN-GP

- Evaluating GANs is challenging
- Difficult to quantify performance
- Some metrics are based on visual quality, including InceptionScore

Analysis: Comparing GAN Formulations



- We use GANs to generate *network parameters*
- We can *indirectly evaluate GAN formulations* by the performance of the parameters they generate
- WGAN-GP converges much faster and is more stable than the original GAN or WGAN

One Slide Summary

- Uncertainty is useful in many scenarios
- BNNs provide a principled way to model parameter and prediction uncertainty
- BNNs can be learned using variational inference (VI) or MCMC methods
  - VI methods make assumptions about the structure of the posterior, and can only yield samples from the approximate posterior
  - MCMC methods draw samples from the true posterior
  - Drawback of MCMC: computational and storage cost
    - Computational cost addressed through the intro of sg-MCMC (e.g., SGLD)
- We can apply Bayesian methods by storing samples drawn using MCMC
  - But this requires a lot of memory, especially for large networks
- Our method alleviates the cost of storing MCMC samples by training a GAN to generate such samples
  - Replace the cost of storing samples with the cost of storing the GAN generator
- We evaluate MCMC-based BNNs on modern applications

Conclusion & Future Work

- We introduced a framework for *distilling BNN posterior samples drawn using SGLD*
- *APD is able to retain the characteristics of the original SGLD samples*, as measured by the performance of generated samples on downstream tasks
- APD outperforms MC dropout on all our tasks
- MCMC methods have not been widely used due to computational cost
  - *APD reduces the storage overhead* involved in maintaining posterior samples
- It is also worthwhile to explore drawing orders of magnitude more samples from SGLD
  - Most of the theoretical results hold in the infinite limit
  - *Online APD* is well-suited for this
- Since we generate model parameters, we can consider this a new, indirect method to evaluate GAN formulations by measuring the performance of the parameters they generate on downstream tasks

Thank you!