Analytically Tractable Inference in Deep Neural Networks

Abstract

Since its inception, deep learning has been overwhelmingly reliant on backpropagation and gradient-based optimization algorithms in order to learn weight and bias parameter values. The *Tractable Approximate Gaussian Inference* (TAGI) algorithm was shown to be a viable and scalable alternative to backpropagation for shallow fully-connected neural networks. In this paper, we are demonstrating how TAGI matches or exceeds the performance of backpropagation, for training classic deep neural network architectures. Although TAGI's computational efficiency is still below that of deterministic approaches relying on backpropagation, it outperforms them on classification tasks and matches their performance for information maximizing generative adversarial networks while using smaller architectures trained with fewer epochs.

1 INTRODUCTION

Since its inception, deep learning has been overwhelmingly reliant on backpropagation and gradient-based optimization algorithms in order to learn weight and bias parameter values. The *Tractable Approximate Gaussian Inference* (TAGI) algorithm [Goulet et al., 2020] was shown to be a viable and scalable alternative to backpropagation. Nevertheless, the method's performance was so far only demonstrated for shallow fully-connected neural networks, because its formulation is not compatible with existing libraries, e.g., TensorFlow or Pytorch. In this paper, we demonstrate how TAGI matches or exceeds the performance of backpropagation, for classic deep neural network architectures. Section 2 first presents the fundamentals of TAGI on which section 3 builds in order to adapt the approach to convolutional neural networks (CNN) [LeCun and Bengio, 1995] and generative adversarial networks (GAN) [Goodfellow et al., 2014]. In parallel, this same section presents how common tools such as hidden state normalization [Goodfellow et al., 2016], i.e. batch normalization [Ioffe and Szegedy, 2015] and layer normalization [Ba et al., 2016], and observation noise decay [Neelakantan et al., 2015] can be integrated into the TAGI's framework. Finally, section 4 compares the performance of TAGI with the state-of-the-art (SOTA) performance for image classification and image generation benchmarks.

2 TAGI

TAGI relies upon two key concepts [Goulet et al., 2020]. First, the joint distribution between the observations and a neural network's parameters is approximated by a multivariate Gaussian distribution. Learning the parameters, i.e., the mean vector and covariance matrix defining the joint probability density function (PDF) requires two simplifying assumptions; (1) the product of Gaussian random variables is also Gaussian with its moments, expected values, covariance, and cross covariances calculated analytically using moment generating functions, and (2) the non-linear activations such as rectified linear unit (ReLU), Tanh, and logistic sigmoid are locally linearized at the expected value of the input hidden unit, thus the output expected value, covariance, and cross-covariance for the activation unit can be computed analytically. The second key concept employed by TAGI has two folds; (1) it requires simplifying the covariance matrices for hidden layers and parameters to a diagonal structure, and (2) it performs the Gaussian conditional inference using a recursive layer-wise procedure which allows for an analytically tractable inference that scales linearly with respect to the number of weight parameters. Despite the simplifying assumptions mentioned above, TAGI was shown to match the performance of feedforward neural networks (FNN) trained with backpropagation for regression and classification benchmarks. In the following section, we will show how to apply TAGI to deep architecture formulations in order to demonstrate its superiority on benchmark

Submitted to the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021). **To be used for reviewing only**.

problems.

3 FORMULATIONS

The fundamental mathematical operations, i.e., additions, multiplications and non-linear activations, in deep neural networks are no different from those in FNN. Therefore, the mathematical formulation developed for defining the joint PDF for observations and parameters in FNN can be readily employed in deeper architectures. This section covers the specificities related to CNNs and GANs.

3.1 CNN

The main building blocks of CNNs are the convolution and pooling layers whereas the convolution operation is performed using fixed-size filters whose weights are learned from data. As the convolution operation [He and Sun, 2015] involves the same operations as a FNN, i.e., multiplication, addition and non-linear transformation, TAGI's mathematical developments can be directly applied to perform the operations in the convolution layer. Small changes occur in the pooling operations which reduce the output size following the convolutional layers. Additional changes also take place in two key tools that are employed with CNNs; hidden states' normalization, and observation noise decay.

3.1.1 Pooling layer

Existing methods such as the stochastic max pooling introduced by Peters and Welling [2018] can be directly be applied with TAGI. In addition to this method, we adapt an *average* pooling approach to the context of TAGI, where, as its name indicates, the output is the average of the activation units in the pooling kernel. For a $\mathtt{K}$-elements pooling kernel, the output Gaussian random variable is defined as

$$A^{\text{pool}} = \tfrac{1}{\mathtt{K}}(A_1 + A_2 + \cdots + A_{\mathtt{K}}), \tag{1}$$

3.1.2 Layer and batch normalization

In the context of TAGI, *layer normalization* [Ba et al., 2016] can be done using the following principles; For a layer of Gaussian activation units $\boldsymbol{A} = [A_1\ A_2 \cdots A_{\mathtt{A}}]^\intercal$, the normalization is defined as

$$\tilde{\boldsymbol{A}} = \frac{\boldsymbol{A}^{(j)} - \mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}, \tag{2}$$

where $\mu_{\boldsymbol{A}}$ and $\sigma_{\boldsymbol{A}}$ are the mean and standard deviation of a Gaussian mixture reduction [Runnalls, 2007] made from all activation units, each having a probability $1/\mathtt{A}$ so that

$$\begin{aligned} \mu_{\boldsymbol{A}} &= \tfrac{1}{\mathtt{A}} \textstyle\sum_{i=1}^{\mathtt{A}} \mu_{A_i} \\ \sigma_{\boldsymbol{A}} &= \sqrt{\tfrac{1}{\mathtt{A}} \left[\textstyle\sum_{i=1}^{\mathtt{A}} (\sigma_{A_i})^2 + \textstyle\sum_{i=1}^{\mathtt{A}} (\mu_{A_i} - \mu_{\boldsymbol{A}})^2 \right]}. \end{aligned} \tag{3}$$

The resulting expected value, covariance, and cross-covariance for the normalized vector of activation units $\tilde{\boldsymbol{A}}$ are given by

$$\begin{aligned} \boldsymbol{\mu}_{\tilde{\boldsymbol{A}}} &= \tfrac{\boldsymbol{\mu}_{\boldsymbol{A}} - \mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}} \\ \boldsymbol{\Sigma}_{\tilde{\boldsymbol{A}}} &= \tfrac{1}{\sigma_{\boldsymbol{A}}^2} \boldsymbol{\Sigma}_{\boldsymbol{A}} \\ \text{cov}(\tilde{\boldsymbol{A}}, \boldsymbol{A}) &= \tfrac{1}{\sigma_{\boldsymbol{A}}} \boldsymbol{\Sigma}_{\boldsymbol{A}}. \end{aligned} \tag{4}$$

By using the short-hand notation defined by Goulet et al. [2020] for hidden units at a layer $j+1$, i.e. $\boldsymbol{Z}^+ \equiv \boldsymbol{Z}^{(j+1)}$, and for the weights, bias, an hidden units at a layer j, i.e. $\{\boldsymbol{W}, \boldsymbol{B}, \boldsymbol{Z}\} \equiv \{\boldsymbol{W}^{(j)}, \boldsymbol{B}^{(j)}, \boldsymbol{Z}^{(j)}\}$, we can use the terms from equation 4 in the TAGI's formulation to update the covariance between hidden states at successive layers

$$\text{cov}(\boldsymbol{Z}, \boldsymbol{Z}^+) = \tfrac{1}{\sigma_{\boldsymbol{A}}} \boldsymbol{\Sigma}_{\boldsymbol{Z}} \mathbf{J} \boldsymbol{\mu}_{\boldsymbol{W}}, \tag{5}$$

where $\mathbf{J}$ is the diagonal Jacobian matrix of the activation function evaluated at $\boldsymbol{\mu}_{\boldsymbol{Z}}$, i.e., $\mathbf{J} = \text{diag}\big(\nabla_{\boldsymbol{z}} \sigma(\boldsymbol{\mu}_{\boldsymbol{Z}})\big)$. The covariance between the weight and bias parameters at a layer j and the hidden states at a layer $j+1$ are

$$\begin{aligned} \text{cov}(\boldsymbol{W}, \boldsymbol{Z}^+) &= \tfrac{1}{\sigma_{\boldsymbol{A}}} \boldsymbol{\Sigma}_{\boldsymbol{W}} \mu_{\boldsymbol{A}} - \tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}} \boldsymbol{\Sigma}_{\boldsymbol{W}} \\ \text{cov}(\boldsymbol{B}, \boldsymbol{Z}^+) &= \boldsymbol{\Sigma}_{\boldsymbol{B}}. \end{aligned} \tag{6}$$

The developments of Equations 5 and 6 are provided in Appendix A. These updated expected value, covariance, and cross-covariance terms can be directly employed in the TAGI's layer-wise recursive inference procedure.

The procedure for *batch normalization* [Ioffe and Szegedy, 2015] is analogous to the layer normalization except that the normalization is made for each hidden units, over a batch of $\mathtt{B}$ observations. The Gaussian activation units contributing to the normalization are $\boldsymbol{A}_j = [A_j^1\ A_j^2 \cdots A_j^{\mathtt{B}}]^\intercal$, so that the number of activation units $\mathtt{A}$ in Equations 3 and 4 is replaced by the number of observations per batch $\mathtt{B}$.

3.1.3 Observation noise decay

In the original TAGI formulation for FNN, the observation errors' standard deviation parameters σ_V were considered as constants during the training. Throughout empirical experimentation with CNNs, we noticed that we could improve the performance when using a decay equation

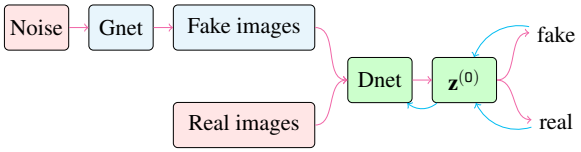
$$\sigma_V^e = \eta \cdot \sigma_V^{e-1}, \tag{7}$$

where e is the current epoch and $\eta \in (0, 1]$ is the decay factor hyperparameter that needs to be learned outside of the TAGI analytic inference procedure. This approach is similar to what is done in standard deep neural networks trained with backpropagation where noise is added to the gradient [Neelakantan et al., 2015] with a decay schedule. In the case of gradient-based learning, this noise consists in discrete samples added to the gradient itself whereas for

TAGI, it consists in additional variance on the output layer so that the update during the inference step will put more weight on the prior rather than on the likelihood, with this effect diminishing with time.

3.2 GENERATIVE ADVERSARIAL NETWORKS

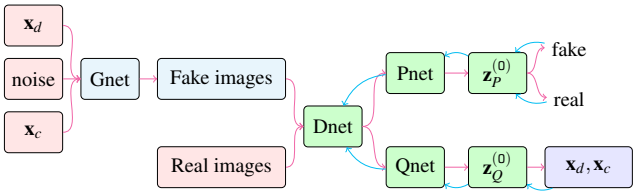
GANs build on a generative approach to create realizations from the input covariates distribution $f(\boldsymbol{x})$. In practice, the input $\boldsymbol{x}$ typically takes the form of images. The GAN architecture consists in a discriminator (Dnet) and a generator (Gnet) network as depicted in Figure 1. The generator network is used to create synthetic data and the discriminator is a binary classification network that allows distinguishing synthetic (i.e. fake) and real data. The generator takes random noise as input. Figure 1 presents the schematic representation of a GAN where the magenta arrows depict the dependencies in the network and correspond to the flow of information during the forward propagation of uncertainty through the network. The blue arrows running backward represent the flow of information during the inference procedure. In Figure 1a, the Dnet takes as input either a fake image generated from an input noise sample that is passed to the Gnet, or a real image; once passed through the Dnet, that image is either classified as fake or real in the output layer $\boldsymbol{z}^{(0)}$. The parameters of the Dnet are then updated in the inference step by relying on the true label for that image. During this inference step, the parameters of the Gnet are not updated. For the generator presented in Figure 1b, an input noise sample is passed to the Gnet in order to return a fake image that is then passed through the Dnet. We then update the Gnet's parameters by forcing the observation for the output layer of the Dnet, $\boldsymbol{z}^{(0)}$, to be classified as a real image. Note that although the inference procedure passes



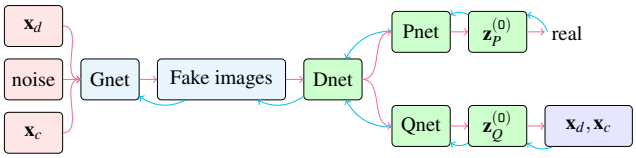
(a) Discriminator

(b) Generator

Figure 1: Schematic representation for (a) the discriminator and (b) the generator network of a GAN. The magenta arrows correspond to the flow of information during the forward propagation of uncertainty through the network and the blue arrows represent the flow of information during the inference procedure.



(a) Discriminator



(b) Generator

Figure 2: Schematic representation for (a) the discriminator and (b) the generator network of an infoGAN. The magenta arrows correspond to the flow of information during the forward propagation of uncertainty through the network and the blue arrows represent the flow of information during the inference procedure.

through the Dnet, only the parameters from the Gnet are updated for the generator.

Information maximizing generative adversarial networks (InfoGAN) [Chen et al., 2016] allow controlling the latent space by disentangling interpretable data representations such as classes, rotation, and width. As a GAN, an infoGAN consists in two networks: discriminator and generator. With infoGAN, the discriminator network has two purposes: (1) distinguishing the fake and real data, and (2) learn the interpretable representations. Figure 2a shows the architecture of the discriminator where the Pnet is employed to classify the real and fake data just like in normal GANs, while the Qnet allows predicting the latent variables associated with an input image. Both the Pnet and Qnet are outputs from the shared Dnet taking as input either a real or fake image that has been created by the Gnet. During inference, the parameters of the Dnet, Pnet and Qnet are updated by relying on the true label for that image as well as the latent variables in the case of fake images. The inputs of the generator networks consist in an input noise sample, a categorical variable vector ($\mathbf{x}_d$), and continuous variable vector ($\mathbf{x}_c$). For example, in the case of the MNIST dataset [LeCun et al., 1998], the discrete variables represent the digits from 0 to 9 and the continuous variable can be the width and rotation of each digit. In practice, $\mathbf{x}_d$ and $\mathbf{x}_c$ are randomly generated during training and can be specifically queried once the training is completed. Note that again only the Gnet's parameters are updated in Figure 2b, even if during inference, the information has to flow backward through the Pnet, Qnet and Dnet.

For image datasets, GANs architectures commonly employ

both CNNs and FNNs which involve the operations such as multiplication, addition and non-linear transformation. Therefore, TAGI's mathematical developments can be directly applied to perform the operations in GAN and infoGAN architectures without having to rely on any gradient whatsoever, or to develop any new equations other than those presented in Section 3.1 for CNNs.

4 BENCHMARKS

In this section, we compare the performance of TAGI for deep neural networks with existing deterministic or Bayesian neural networks trained with backpropagation. In the case of TAGI, the hyperparameters are kept the same for all benchmarks with a batch size $\mathtt{B} = 16$, a maximal number of epochs $\mathtt{E} = 50$, a noise discount factor $\eta = 0.975$, and all weights are initialized using the He's approach [He et al., 2015]. These hyperparameters are kept constant across all benchmarks to demonstrate the robustness of the approach towards them. The only hyperparameter specific to each experiment is the initial observation noise standard deviation σ_V^0. Additionally, for the MNIST [LeCun et al., 1998] and CIFAR10 [Krizhevsky et al., 2009] datasets, the input data was first normalized between 0 and 1 and then the mean of the normalized data was subtracted. For the celebA dataset [Liu et al., 2015], we preprocess the input data by subtracting the mean and dividing by the standard deviation of the input data.

4.1 MNIST

The first benchmark is the MNIST digit classification task. Table 1 compares the classification accuracy with the results from Wan et al. [2013] where both approaches use the same CNN architecture with 2 convolutional layers (32-64) and a fully connected layer with 150 hidden units. For this experiment, the initial observation noise standard deviation $\sigma_V^0 = 1$. Results show that TAGI can already reach a classification error $< 2\%$ after the first epoch and after $\mathtt{E} = 50$ epochs, the performance is slightly better than the results obtained with backpropagation trained with over 20 times as many epochs.

Table 1: Comparison of the classification performance on MNIST between a network trained with TAGI and backpropagation (BP). (HP: hyperparameters; E, number of epochs, B, batch size)

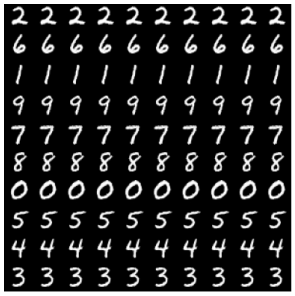
	Error Rate [%]		HP	
	$e = 1$	$e = \mathtt{E}$	E	B
TAGI	1.88	0.64	50	16
BP [Wan et al., 2013]	-	0.67	1000	128

In a second experiment on the same dataset, we employ an infoGAN (see §3.2) in order to test the performance of TAGI on unsupervised learning tasks. The hyperparameters specific to this experiment are the two observation noise initial standard deviations for the output of the Pnet and Qnet that are taken as $\sigma_V^{0,P} = \sigma_V^{0,Q} = 3$. As a reference, we employ the input latent space and infoGAN architecture described in Chen et al. [2016]. The input latent space includes 62 Gaussian noise variables, one categorical variable x_d containing ten classes, and two continuous variables for representing the width and rotation of the digit. In the case of TAGI, we employ the same input latent space with a modification to the architecture. Specifically, for the Gnet, we use half as many convolutional filters and half as many hidden units in the output's fully connected layer. For the Qnet, we use a fully connected layer containing 300 hidden units instead of 128. Overall, our architecture contains ≈ 2.1M parameters versus 13.3M for the network used as reference [Chen et al., 2016], and we train over 50 epochs whereas they train with 100 epochs. The complete architecture's details are provided in Appendix B.2.1.

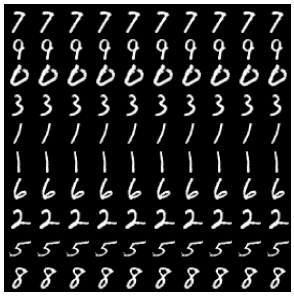
The results obtained using TAGI are presented in Figures 3a–c while the results obtained with backpropagation are shown in Figures 3d–f. Note that the results for the MNIST trained with backpropagation were obtained using the code from Mohit [2019]. Overall, we denote an advantage for the network trained with backpropagation as TAGI can only disentangle 9 out of 10 digits. Moreover, for TAGI, the two continuous variables forming the latent space do not have an effect as clear as in the case of backpropagation. The reason for this sub-par performance is that for TAGI, the output layer uncertainty is homoscedastic whereas, in the case of Chen et al. [2016], it is heteroscesastic so that they optimize both the expected values and variances for the two continuous variables on the output layer. With its current formulation TAGI can only handle homoscedastic output uncertainty which jeopardizes its capacity at disentangling the latent space with the unsupervised setup where we learn both the categorical and continuous variables at once.

4.2 CIFAR-10

The second benchmark is the CIFAR-10 image classification task. Experiments are conducted for a simple network with three convolutional layers, as well as for the Resnet18 architecture [He et al., 2016]. For both cases, we also experimented with the usage of layer and batch normalization. The details regarding the 3-layers convolutional network are presented in Appendix B.1.2. Note that we only apply the data augmentation techniques including random cropping and horizontal flipping [Osawa et al., 2019] to the Resnet18 architecture for the comparison purpose with the neural networks trained with backpropagation. In the case of TAGI, the initial observation noise standard deviation was set to the

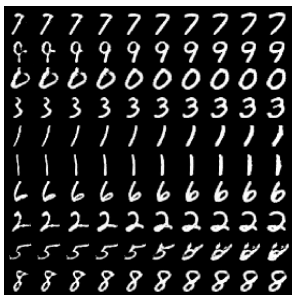


(a) Digits – Backprop



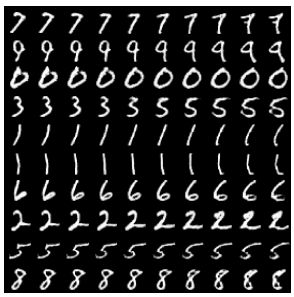
(b) Digits– TAGI

(c) Width – Backprop



(d) Width – TAGI

(e) Rotation – Backprop



(f) Rotation – TAGI

Figure 3: Comparison of the performance of a backpropagation and TAGI trained InfoGAN applied to the MNIST dataset. Results present reference digit generation (a,d), and their control with a latent space consisting of the width (b,e) and rotation (c,f).

same value as for the MNIST experiment, i.e., $\sigma_V^0 = 1$. Table 2 presents the classification error rate for TAGI-trained networks as well as the results from several papers using deterministic as well as Bayesian methods for the same network architectures. Note that in the case of Resnet18 network, the results presented are the average from three runs.

First, for the 3-layer convolutional architecture, the results show that TAGI matches the SOTA performance while using a lower number of epochs for the experiments with the batch normalization and without normalization. TAGI outperforms the SOTA performance for the experiment with the layer normalization. Second, for the Resnet18 architecture, TAGI outperforms deterministic and Bayesian approach with a third of the epochs for training. Note that the performance of TAGI still matches the performance of MC-dropout and VOGN even if we employ half as many convolutional filters in the Resnet18 architecture, i.e., Resnet18half.

Table 3 presents the predictive uncertainty calibration metrics [Guo et al., 2017] for the Resnet18 and Resnet18half architecture. The results show that for the Resnet18, TAGI performs poorly on the NLL and ECE criteria, yet, for the same criteria, it performs on par with other methods for the Resnet18half. One of reason behind this limitation is that TAGI is currently limited to using homoscedastic observation noise parameters σ_V, which is not the case for existing methods relying on backpropagation. Despite this limitation, none of the backpropagation-based methods match the AUROC criterion obtained with TAGI.

4.3 CELEBA

In the last experiment, we applied infoGAN to the CelebA dataset for which we downsampled the original images from $3 \times 224 \times 224$ to $3 \times 32 \times 32$. The hyperparameters specific to this experiment are the two observation noise initial standard deviations for the output of the Pnet and Qnet that are taken as $\sigma_V^{0,P} = 3$ and $\sigma_V^{0,Q} = 8$. As a reference, we use the input latent space and the infoGAN architecture described in Chen et al. [2016]. The input latent space contains a Gaussian noise vector including 128 variables along with ten categorical variables $\mathbf{x_d}$ each having ten classes. The specifications of the architecture used with TAGI is presented in Appendix B.2.2. Overall the TAGI-trained architecture contains approximately 0.7M parameters in comparison with 4.1M for the architecture which serves here as reference. Results for the infoGan trained with backpropagation were obtained using the code from Mohit [2019].

Figure 4 compares a sample of the training images with those generated with both the architecture from Chen et al. [2016] trained with backpropagation over 100 epochs, and with an architecture containing six times fewer parameters trained with TAGI over 50 epochs. These results show that despite using a smaller network, TAGI performs at par with the larger network trained with backpropagation.

Figure 5 presures the variations in images for the latent variable controlling the hair colour. Again, TAGI performs at par with the larger network trained with backpropagation. Other latent variables such as the gender, azimuth, hairstyle, and contrasts are presented in Appendix D

5 DISCUSSION

For the infoGAN experiments, the results have shown that TAGI can reach a performance at par with a network trained with backpropagation while using significantly smaller architectures and a lower number of epochs. We opted for this

Table 2: Comparison of the classification performance on CIFAR-10 between a network trained with TAGI and backpropagation (BP), BP with layer normalization (-LN), BP with batch normalization (-BN), MC-dropout, and VOGN. (HP: hyperparameters; E: number of epochs; B: batch size ; DA: data augmentation).

Architecture	Method	Error rate [%]		HP	
		$e = 1$	$e = \mathtt{E}$	E	B
3 conv.	**TAGI**	44.3	23.6	50	16
	TAGI-LN	41.4	22.6	50	16
	TAGI-BN	42.6	22.0	50	16
	BP [Wan et al., 2013]	-	23.5	150	128
	BP-LN [Ren et al., 2016]	-	27.9	100	100
	BP-BN [Ren et al., 2016]	-	21.9	100	100
Resnet18 (DA)	**TAGI**	65.9	13.8	50	16
	BP [Osawa et al., 2019]	-	14.0	160	128
	MC-dropout [Osawa et al., 2019]	-	17.2	161	128
	VOGN [Osawa et al., 2019]	-	15.7	161	128
Resnet18half (DA)	**TAGI**	65.9	14.9	50	16

Table 3: Comparison of the predictive uncertainty calibration performance on CIFAR-10 between a network trained with TAGI and backpropagation (BP), BP, MC-dropout, and VOGN. (HP: hyperparameters; E: number of epochs; B: batch size ; NLL: Negative Log-Likelihood; ECE: Expected Calibration Error; AUROC: Area Under the Receiver Operating Characteristics). See Table 15 for standard deviations.

Architecture	Method	NLL	ECE	AUROC	HP	
		$e = \mathtt{E}$	$e = \mathtt{E}$	$e = \mathtt{E}$	E	B
Resnet18 (DA)	**TAGI**	0.588	0.143	0.980	50	16
	BP [Osawa et al., 2019]	0.55	0.082	0.877	160	128
	MC-dropout [Osawa et al., 2019]	0.51	0.166	0.768	161	128
	VOGN [Osawa et al., 2019]	0.477	0.04	0.876	161	128
Resnet18half (DA)	**TAGI**	0.541	0.046	0.978	50	16

approach because TAGI's implementation is not yet optimized for computational efficiency so the current computing time is approximately one order of magnitude higher than when training the same deterministic neural network using backpropagation.

Throughout the experiments, TAGI employs far fewer hyperparameters than traditional neural networks trained with backpropagation. Moreover, TAGI's robustness toward these hyperparameter values allows keeping most of them constant across various tasks. The main hyperparameters that still need to be adapted with the size of the network employed are the initial output noise standard deviations, σ_V^0. In the future, these hyperparameters could be passed as hidden states that are learned online like other weights and bias, while simultaneously adding the capacity to model heteroscedastic output noise variance.

In addition, we can improve the classification performance with the same architecture by using smaller batch sizes, which, however, decreases the potential for parallelization. The same applies, for larger batch sizes where the losses in classification performance is compensated by an increase in computation parallelization. A systematic study of the effect of batch sizes remains to be explored.

6 CONCLUSION

This paper demonstrates the applicability of TAGI to deep neural networks. The results obtained from different tasks

(a) Sample training images from the CelebA dataset.

(b) Backpropagation (100 epochs)

(c) TAGI (50 epochs)

Figure 4: Comparison of a sample of training images (a) with the synthetic images generated using infoGAN with (b) the architecture presented in Chen et al. [2016] trained with backpropagation, and with (c) a network six times smaller trained with TAGI.

(a) Backpropagation (100 epochs)

(b) TAGI (50 epochs)

Figure 5: Comparison of the capacity to control the latent space for hair colour for (a) the architecture presented in Chen et al. [2016] and trained with backpropagation, and (b) a network six times smaller trained with TAGI.

confirm that TAGI is able to perform analytical inference for the parameters of deep neural network. Although the computational efficiency is still below that of standard approaches relying on backpropagation, TAGI outperforms these approaches on classification tasks and match the performance for infoGANs while using smaller architectures trained with fewer epochs. Future code developments oriented toward efficiency should reduce that gap.

References

J.L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv:1606.03657*, 2016.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680, 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016.

J.-A. Goulet, L. H. Nguyen, and S. Amiri. Tractable approximate gaussian inference for Bayesian neural networks. *arXiv*, (2004.09281, cs.LG), 2020.

C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1321–1330. JMLR, 2017.

K. He and J. Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

A. Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

J. Mohit. InfoGAN-PyTorch. `https://github.com/Natsu6767/InfoGAN-PyTorch`, 2019.

A. Neelakantan, L. Vilnis, Q.V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. *CoRR*, abs/1511.06807, 2015.

K. Osawa, S. Swaroop, M. E. E. Khan, A. Jain, R. Eschenhagen, R. E. Turner, and R. Yokota. Practical deep learning with bayesian principles. In *Advances in neural information processing systems*, pages 4287–4299, 2019.

J.W.T Peters and M. Welling. Probabilistic binary neural networks. *arXiv preprint arXiv:1809.03368*, 2018.

M. Ren, R. Liao, R. Urtasun, F. H. Sinz, and R. S. Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. *arXiv preprint arXiv:1611.04520*, 2016.

A. R. Runnalls. Kullback-leibler approach to gaussian mixture reduction. *IEEE Transactions on Aerospace and Electronic Systems*, 43(3):989–999, 2007.

L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

APPENDIX

A CROSS-COVARIANCES FOR LAYER NORMALIZATION

Assuming that $\boldsymbol{Z}$, $\boldsymbol{W}$, and $\boldsymbol{B}$ are independent on each other,

$$\begin{aligned}
\operatorname{cov}(\boldsymbol{Z},\boldsymbol{Z}^{+}) &= \operatorname{cov}\left(\boldsymbol{Z}, \tfrac{1}{\sigma_{\boldsymbol{A}}}\boldsymbol{W}\boldsymbol{A} - \tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\boldsymbol{W} + \boldsymbol{B}\right) \\
&= \tfrac{1}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{Z},\boldsymbol{W}\boldsymbol{A}) - \cancelto{0}{\tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{Z},\boldsymbol{W})} + \cancelto{0}{\operatorname{cov}(\boldsymbol{Z},\boldsymbol{B})} \\
&= \tfrac{1}{\sigma_{\boldsymbol{A}}}\left[\operatorname{cov}(\boldsymbol{Z},\boldsymbol{A})\boldsymbol{\mu}_{\boldsymbol{W}} + \cancelto{0}{\operatorname{cov}(\boldsymbol{Z},\boldsymbol{W})\boldsymbol{\mu}_{\boldsymbol{A}}}\right] \\
&= \tfrac{1}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{Z},\boldsymbol{Z})\mathbf{J}\boldsymbol{\mu}_{\boldsymbol{W}},
\end{aligned} \tag{8}$$

where $\boldsymbol{A} = \mathbf{J}(\boldsymbol{Z} - \boldsymbol{\mu}_{\boldsymbol{Z}}) + \sigma(\boldsymbol{Z})$ with $\sigma(.)$ being the activation function.

$$\begin{aligned}
\operatorname{cov}(\boldsymbol{W},\boldsymbol{Z}^{+}) &= \operatorname{cov}\left(\boldsymbol{W}, \tfrac{1}{\sigma_{\boldsymbol{A}}}\boldsymbol{W}\boldsymbol{A} - \tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\boldsymbol{W} + \boldsymbol{B}\right) \\
&= \tfrac{1}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{W},\boldsymbol{W}\boldsymbol{A}) - \tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{W},\boldsymbol{W}) + \cancelto{0}{\operatorname{cov}(\boldsymbol{W},\boldsymbol{B})} \\
&= \tfrac{1}{\sigma_{\boldsymbol{A}}}\left[\cancelto{0}{\operatorname{cov}(\boldsymbol{W},\boldsymbol{A})\boldsymbol{\mu}_{\boldsymbol{W}}} + \operatorname{cov}(\boldsymbol{W},\boldsymbol{W})\boldsymbol{\mu}_{\boldsymbol{A}}\right] - \tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{W},\boldsymbol{W}) \\
&= \tfrac{1}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{W},\boldsymbol{W})\boldsymbol{\mu}_{\boldsymbol{A}} - \tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{W},\boldsymbol{W}).
\end{aligned} \tag{9}$$

$$\begin{aligned}
\operatorname{cov}(\boldsymbol{Z},\boldsymbol{Z}^{+}) &= \operatorname{cov}\left(\boldsymbol{B}, \tfrac{1}{\sigma_{\boldsymbol{A}}}\boldsymbol{W}\boldsymbol{A} - \tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\boldsymbol{W} + \boldsymbol{B}\right) \\
&= \cancelto{0}{\tfrac{1}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{B},\boldsymbol{W}\boldsymbol{A})} - \cancelto{0}{\tfrac{\mu_{\boldsymbol{A}}}{\sigma_{\boldsymbol{A}}}\operatorname{cov}(\boldsymbol{B},\boldsymbol{W})} + \operatorname{cov}(\boldsymbol{B},\boldsymbol{B}) \\
&= \operatorname{cov}(\boldsymbol{B},\boldsymbol{B}).
\end{aligned} \tag{10}$$

B MODEL ARCHITECTURES

This appendix contains the specifications for each model architecture in the experiment section. D refers to a layer depth; W refers to a layer width; H refers to the layer height in case of convolutional or pooling layers; K refers to the kernel size; P refers to the convolutional kernel padding; S refers to the convolution stride; σ refers to the activation function type; ReLU refers to rectified linear unit; lReLU refers to leaky rectified linear unit.

B.1 CLASSIFICATION

B.1.1 MNIST

Table 4: Configuration details for the CNN applied to the MNIST classification problem.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$1 \times 28 \times 28$	-	-	-	-
Convolutional	$32 \times 27 \times 27$	4×4	1	1	ReLU
Pooling	$32 \times 13 \times 13$	3×3	0	2	-
Convolutional	$64 \times 9 \times 9$	5×5	0	1	ReLU
Pooling	$64 \times 4 \times 4$	3×3	0	2	-
Fully connected	$150 \times 1 \times 1$	-	-	-	ReLU
Output	$11 \times 1 \times 1$	-	-	-	-

B.1.2 CIFAR10

Table 5: Configuration details for the CNN applied to the CIFAR10 classification problem.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$3 \times 32 \times 32$	-	-	-	-
Convolutional	$32 \times 32 \times 32$	5×5	2	1	ReLU
Pooling	$32 \times 16 \times 16$	3×3	1	2	-
Convolutional	$32 \times 16 \times 16$	5×5	2	1	ReLU
Average pooling	$32 \times 8 \times 8$	3×3	1	2	-
Convolutional	$64 \times 8 \times 8$	5×5	2	1	ReLU
Average pooling	$64 \times 4 \times 4$	3×3	1	2	-
Fully connected	$64 \times 1 \times 1$	-	-	-	ReLU
Output	$11 \times 1 \times 1$	-	-	-	-

B.2 INFOGAN

B.2.1 MNIST

Table 6: Configuration details for Dnet in the experiment on the MNIST Dataset. The leaky rate for lReLU is set to 0.2.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$1 \times 28 \times 28$	-	-	-	-
Convolutional	$32 \times 28 \times 28$	3×3	1	1	lReLU
Batch normalization	$32 \times 28 \times 28$	-	-	-	
Average pooling	$32 \times 14 \times 14$	3×3	1	2	-
Convolutional	$64 \times 14 \times 14$	3×3	1	1	lReLU
Batch normalization	$64 \times 14 \times 14$	-	-	-	
Average pooling	$64 \times 7 \times 7$	3×3	1	2	-
Output	$512 \times 1 \times 1$	-	-	-	lReLU

Table 7: Configuration details for Pnet in the experiment on the MNIST Dataset.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$512 \times 1 \times 1$	-	-	-	-
Output	$1 \times 1 \times 1$	-	-	-	-

Table 8: Configuration details for Qnet in the experiment on the MNIST Dataset.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$512 \times 1 \times 1$	-	-	-	-
Fully connected	$300 \times 1 \times 1$	-	-	-	ReLU
Output	$13 \times 1 \times 1$	-	-	-	-

Table 9: Configuration details for Gnet in the experiment on the MNIST Dataset.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$75 \times 1 \times 1$	-	-	-	-
Fully connected	$3072 \times 1 \times 1$	-	-	-	ReLU
Transposed convolutional	$64 \times 7 \times 7$	3×3	1	1	ReLU
Transposed convolutional	$32 \times 14 \times 14$	3×3	1	2	ReLU
Output	$1 \times 28 \times 28$	3×3	1	2	-

B.2.2 CelebA

Table 10: Configuration details for Dnet in the experiment on the CelebA Dataset. The leaky rate for lReLU is set to 0.2.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$3 \times 32 \times 32$	-	-	-	-
Convolutional	$32 \times 32 \times 32$	3×3	1	1	lReLU
Batch normalization	$32 \times 32 \times 32$	-	-	-	
Average pooling	$32 \times 16 \times 16$	3×3	1	2	-
Convolutional	$32 \times 16 \times 16$	3×3	1	1	lReLU
Batch normalization	$32 \times 16 \times 16$	-	-	-	
Average pooling	$32 \times 8 \times 8$	3×3	1	2	-
Convolutional	$64 \times 8 \times 8$	3×3	1	1	lReLU
Batch normalization	$64 \times 8 \times 8$	-	-	-	
Average pooling	$64 \times 4 \times 4$	3×3	1	2	-
Output	$256 \times 1 \times 1$	-	-	-	lReLU

Table 11: Configuration details for Pnet in the experiment on the CelebA Dataset.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$256 \times 1 \times 1$	-	-	-	-
Output	$1 \times 1 \times 1$	-	-	-	-

Table 12: Configuration details for Qnet in the experiment on the CelebA Dataset.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$256 \times 1 \times 1$	-	-	-	-
Fully connected	$256 \times 1 \times 1$	-	-	-	ReLU
Output	$110 \times 1 \times 1$	-	-	-	-

Table 13: Configuration details for Gnet in the experiment on the CelebA Dataset.

Layer	$D \times W \times H$	$K \times K$	P	S	σ
Input	$238 \times 1 \times 1$	-	-	-	-
Fully connected	$1024 \times 1 \times 1$	-	-	-	ReLU
Transposed convolutional	$64 \times 4 \times 4$	3×3	1	1	ReLU
Transposed convolutional	$64 \times 8 \times 8$	3×3	1	2	ReLU
Transposed convolutional	$32 \times 16 \times 16$	3×3	1	2	ReLU
Transposed convolutional	$32 \times 32 \times 32$	3×3	1	2	ReLU
Transposed convolutional	$32 \times 32 \times 32$	3×3	1	1	ReLU
Output	$3 \times 32 \times 32$	3×3	1	2	-

C SUPPLEMENTARY RESULTS FOR CLASSIFICATION

Table 14: Comparison of the classification performance on CIFAR-10 between a network trained with TAGI and backpropagation (BP), BP, MC-dropout, and VOGN on residual networks over three runs. (HP: Hyperparameters; E: number of epochs; B: batch size ; DA: data augmentation).

Architecture	Method	Error rate [%]		HP	
		$e = 1$	$e = \mathtt{E}$	E	B
Resnet18 (DA)	**TAGI**	65.9 ± 0.667	13.8 ± 0.921	50	16
	BP [Osawa et al., 2019]	-	14.0 ± 0.257	160	128
	MC-dropout [Osawa et al., 2019]	-	17.2 ± 0.208	161	128
	VOGN [Osawa et al., 2019]	-	15.7 ± 0.195	161	128
Resnet18half (DA)	**TAGI**	65.9 ± 0.4	14.9 ± 0.7	50	16

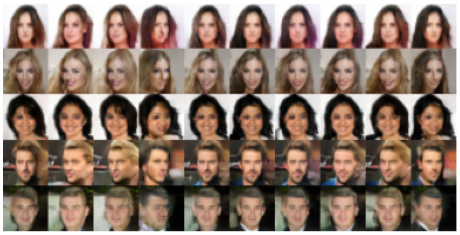
Table 15: Comparison of the predictive uncertainty calibration performance on CIFAR-10 between a network trained with TAGI and backpropagation (BP), BP, MC-dropout, and VOGN over three runs. (HP: Hyperparameters; E: number of epochs; B: batch size ; NLL: Negative Log-Likelihood; ECE: Expected Calibration Error; AUROC: Area Under the Receiver Operating Characteristics).

Architecture	Method	NLL	ECE	AUROC	HP	
		$e = \mathtt{E}$	$e = \mathtt{E}$	$e = \mathtt{E}$	E	B
Resnet18 (DA)	**TAGI**	0.588 ± 0.09	0.143 ± 0.08	0.98 ± 0.001	50	16
	BP [Osawa et al., 2019]	0.55 ± 0.01	0.082 ± 0.002	0.877 ± 0.001	160	128
	MC-dropout [Osawa et al., 2019]	0.51 ± 0	0.166 ± 0.025	0.768 ± 0.004	161	128
	VOGN [Osawa et al., 2019]	0.477 ± 0.006	0.04 ± 0.002	0.876 ± 0.002	161	128
Resnet18half (DA)	**TAGI**	0.541 ± 0.04	0.046 ± 0.001	0.978 ± 0.003	50	16

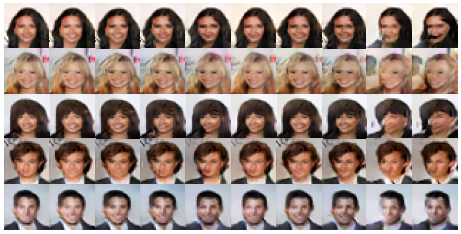
D CELEBA LATENT SPACE VARIATIONS

(a) Gender – Backpropagation

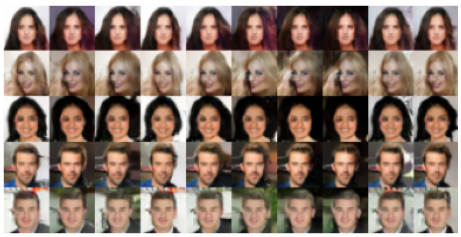
(b) Gender – TAGI



(c) Azimuth – Backpropagation



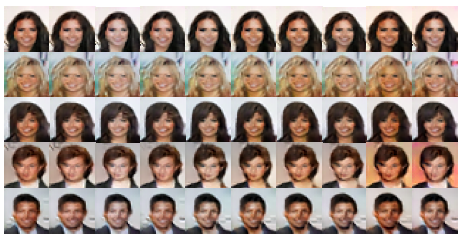
(d) Azimuth – TAGI



(e) Hair style – Backpropagation

(f) Hair style – TAGI

(g) Contrast – Backpropagation



(h) Constrast – TAGI

Figure 6: Comparison of the capacity to control the latent space for the architecture presented in Chen et al. [2016] and trained with backpropagation, and a network six times smaller trained with TAGI over half as many epochs.