Machine Learning in Measurement Part 2: Uncertainty Quantification

Hussein Al Osman and Shervin Shirmohammadi

In spite of the advent of Machine Learning (ML) and its successful deployment in measurement systems, little information can be found in the literature about uncertainty quantification in these systems [1]. Uncertainty is crucial for the adoption of ML in commercial products and services. Designers are now being encouraged to be upfront about the uncertainty in their ML systems, because products that specify their uncertainty can have a significant competitive advantage and can unlock new value, reduce risk, and improve usability [2]. In this article, we will describe uncertainty quantification in ML. Because there isn't enough room in one article to explain all ML methods, we concentrate on Deep Learning (DL), which is one of the most popular and effective ML methods in I&M [3]. Please note that this article follows and uses concepts from Part 1 [4], so readers are highly encouraged to first read that part. In addition, we assume the reader has a basic understanding of both DL and uncertainty. Readers for whom this assumption is false are encouraged to first read the brief introduction to DL and its applications in I&M presented in [3] as well as the uncertainty tutorial in [5].

Researchers have always understood that the reliability of predictions made by a DL model varies depending on the input; i.e., the DL model does not exhibit the same constant performance for different regions in the input space. Uncertainty can be used to quantify that, and this is beginning to gain attention and will undoubtedly continue to evolve in the coming years. In Part 1, we mentioned that variable-output ML models have randomness in their output. In ML literature, the variance of outputs is used as a measure of uncertainty, which is similar to Type A standard uncertainty in GUM [6] except that the latter uses the standard deviation; i.e., the square root of the variance, and not the variance itself. Please keep this small but important difference in mind when reading the sections below, and let us now see how an ML model can produce randomness, starting with an explanation of the concept of Monte Carlo Dropout which is used by most variable-output ML models.

Monte Carlo Dropout

Monte Carlo dropout uses the standard dropout approach, that was originally proposed for regularization (i.e., reducing overfitting), to estimate the ML uncertainty. So, let's first see how this dropout works.

Dropout

Deep Artificial Neural Networks (ANNs) tend to overfit the training data. This problem is exacerbated if the training dataset is small and/or the model is complex. Bagging is an ensemble training technique that can reduce overfitting [7]. The larger the ensemble, the less likely overfitting would occur. In fact, it is common to use an ensemble of up to ten models to achieve satisfactory results [8]. However, training an ensemble of ANNs can be tedious while the resulting ensemble is evidently more complex compared to a single model. Dropout [8] is a technique that produces similar results to bagging without increasing training time or the complexity of the model. Dropout trains subnetworks that are formed by randomly dropping nodes from the neural network's non-output layers according to a predefined probability. Hence, at each learning step, the output of each network node is multiplied by a Bernoulli distributed random variable. This is analogous to simultaneously training an exponentially large number of related ANNs. When dropout is used strictly for regularization, it is applied only during training. Once the network is trained, all network nodes are subsequently used for prediction.

Applying Dropout at Run Time

To estimate the ML uncertainty, the Monte Carlo Dropout approach stipulates the application of dropout during not only training but also run time. This introduces a degree of randomness into the prediction process. Hence, the network can produce different outputs for the same input depending on the nodes that are randomly cancelled out. Therefore, this results in a *variable-output* ML model. When dropout is applied at run time, the input can be applied to the model several times. Each time, the input is processed by a slightly different network as the dropout cancels some nodes according to a predefined probability. The set of predictions allow us to estimate an output distribution that renders information about uncertainty.

1094-6969/21/$25.00©2021IEEE

For simplicity, let's consider a network with a single hidden layer. We obtain the model's predictions as follows:

$$\hat{f}(x) = \sqrt{\frac{1}{K_1}} g\left(xZ_1W_1 + b_1\right)Z_2W_2 + b_2 \tag{1}$$

where $\hat{f}$ is the ANN model trained with the dataset, x is the input vector, g is the activation function, K_1 is the number of nodes at layer 1 (hidden layer), Z_1 and Z_2 are the dropout matrices, W_1 and W_2 are the weight matrices, and b_1 and b_2 are the bias vectors for layers 1 and 2. In fact, Z_1 and Z_2 are diagonal matrices with values sampled from a Bernoulli distribution according to a predefined dropout probability along the diagonal. Multiplying by the dropout matrices cancels certain rows in the weight matrix which is analogous to removing a node from the network during a forward pass.

Given that T evaluations are performed on an input, the prediction and its uncertainty are obtained as the mean and variance:

$$Mean = \frac{1}{T}\sum_{i=1}^{T}\hat{f}_i(x) \tag{2}$$

$$Variance = \frac{1}{T}\sum_{i=1}^{T}\left(\hat{f}_i(x) - E(x)\right)^2 \tag{3}$$

Given that the Monte Carlo Dropout method allows us to realize a *variable-output ML model*, (3) estimates the ML uncertainty as defined in Table 2 of [4] (recall that the I&M uncertainty is the square root of that; i.e., the standard deviation). As opposed to the variance presented in (3), we calculate the variance on the output of the T predictions produced by the slightly different models generated through dropout in (3) (Fig. 1).

Although each input evaluation requires T forward passes through the model during prediction, the training process remains unchanged. To train the model, we use a gradient descent approach while re-populating the dropout diagonal matrices at every training step. If we consider a Euclidean loss, then the loss function can be expressed as:

$$Loss = \frac{1}{N}\sum_{i=1}^{N}\left\|\hat{f}(x_i) - y_i\right\|^2 + \lambda\sum_{l=1}^{L}\left(p_a\left\|W_l\right\|_2^2 + \left\|b_l\right\|_2^2\right) \tag{4}$$

where $\hat{f}(x_i)$ is the estimation output for input x_i, y_i is the true value at observation i, N is the number of instances in the

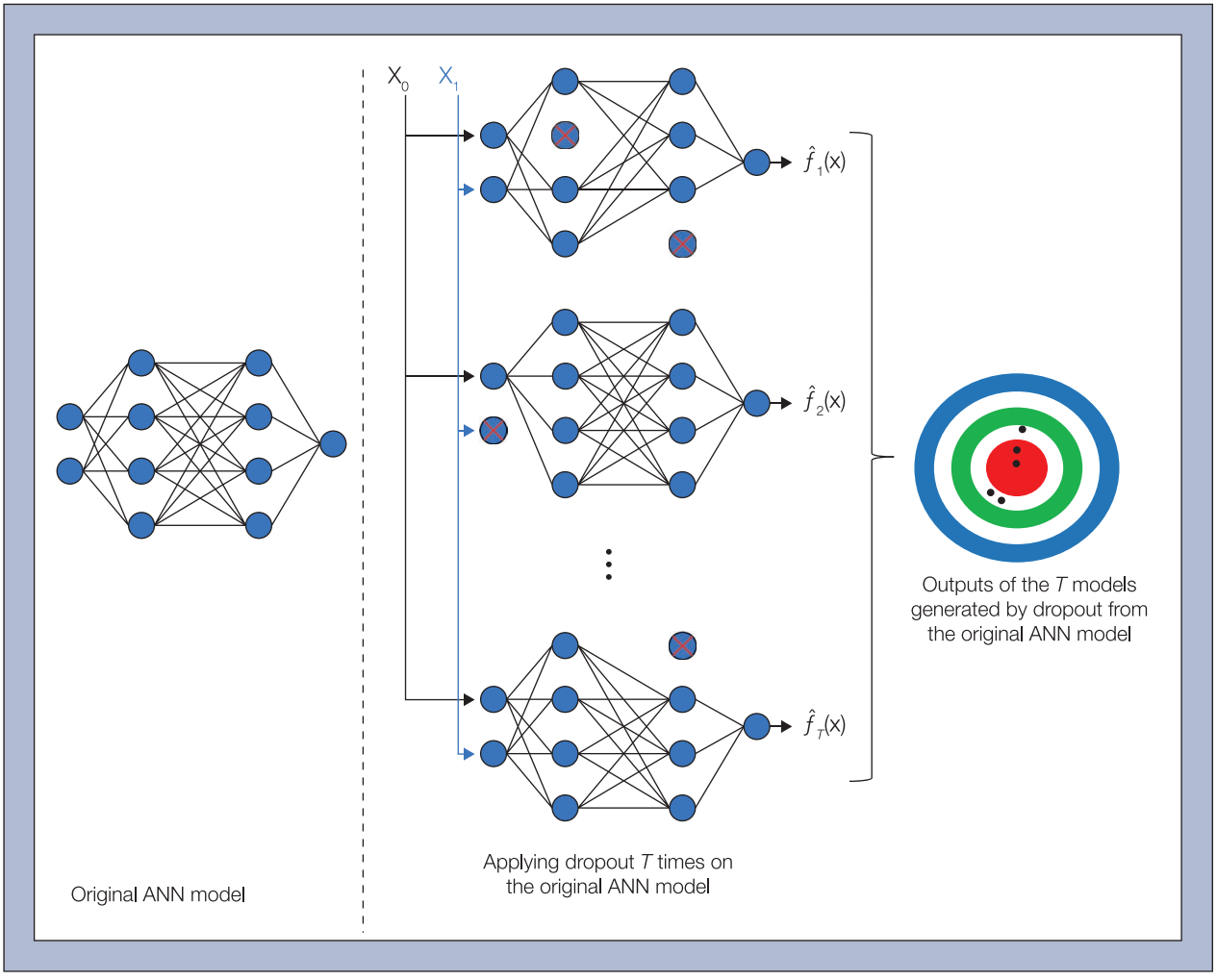

Fig. 1. *T* slightly different models are generated by the dropout process. Each model evaluates the same input and produces its output. The mean of the outputs corresponds to the prediction, and the variance corresponds to the uncertainty.

training batch, λ is the decay factor used for regularization, p_d is the dropout probability, and L is the number of layers. The dropout probability in (4) scales the weight matrices to correct for the rows removed through the dropout process.

Monte Carlo Batch Normalization

The Monte Carlo dropout approach can be shown to estimate a Bayesian model where ML uncertainty is estimated as the variance of multiple predictions [9]. Hence, this approach introduces stochasticity into the network that permits the estimation of uncertainty. Inspired by this approach, Teye *et al*. [10] noticed that batch normalization, a technique that reduces training time [11], has a regularization effect that introduces randomness as well, a property that makes such network useful for uncertainty estimation. The idea in batch normalization is to normalize the output of neurons in hidden layers on a mini batch of the training samples during training by subtracting from the batch's mean and dividing by its standard deviation. The weights are updated for every mini batch during training. At run time, the output of neurons in hidden layers is subtracted from the training set's mean and divided by its standard deviation. They argue that such normalization reduces the covariate shift in the output of the hidden layers. However, more recent research suggests that batch normalization renders the optimization landscape traversed during gradient descent smoother [12]. Hence, the optimization process is less likely to linger in somewhat flat areas that increase training time. Given that mini batches are randomly chosen during training, batch normalization introduces stochasticity that results in regularizing the network. However, batch normalization is seldom employed as the sole regularization approach and is often combined with dropout. By combining both approaches, we can afford to reduce the probability of dropping out nodes in the network as dropout is also not the sole regularization approach employed.

Teye *et al*. [10] show that similar to Monte Carlo dropout, a network trained with batch normalization approximates a Bayesian model. For prediction, the input is evaluated several times, hence resulting in a *variable-output ML model* which enables the estimation of uncertainty as described in Table 2 of [4]. However, the run time normalization is not performed using training set probability distribution parameters (mean and standard deviation). It is performed using the stochastic parameters of a randomly sampled mini-batch. Similar to the Monte Carlo dropout, this results in output variability that enables the estimation of the uncertainty.

Deep Ensembles

Deep Ensembles [13] refers to a technique that permits the estimation of two types of uncertainties: epistemic and aleatoric uncertainty. As mentioned in Table 2 of [4], aleatoric uncertainty is associated with randomness in the training data while the epistemic uncertainty relates to the model uncertainty resulting from a lack of correct training data in some areas of the input space. This is shown in Fig. 2, where region A has a higher aleatoric uncertainty compared to regions B and C,

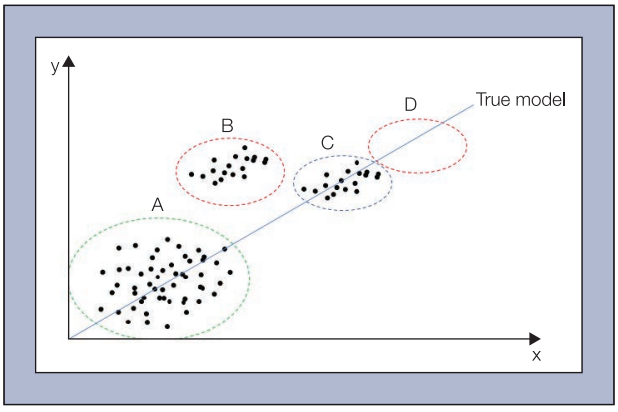

Fig. 2. Uncertainty in the training dataset. Region A has the highest aleatoric uncertainty, while region D has the highest epistemic uncertainty.

due to the larger variance in A's data noise. Region B and C have about the same aleatoric uncertainty as each other, although region B has a higher epistemic uncertainty because its data seems to have a systematic offset. Finally, region D has the highest epistemic uncertainty because its data are missing. In Deep Ensembles, the ANN model is trained to approximate the epistemic uncertainty as part of its output (see "ANN with Uncertainty Output" section). Moreover, the aleatoric uncertainty is estimated through the use of an ensemble of similar models making predictions on the same input (see "Deep Ensemble Dropout" section). The approach further applies adversarial training to improve model performance.

ANN with Uncertainty Output

Lakshminarayanan *et al*. [13] propose an elegant approach that incorporates the uncertainty into the output of the ANN. Hence, given an input x, instead of predicting an output, the ANN outputs a predictive distribution $\hat{p}(x)$ (Fig. 3b). The goal of the training process is to find the parameters of the predictive distribution which fully describe the ANN's prediction along with its associated uncertainty.

For regression, the network parameters are usually trained through the minimization of a Mean Square Error (MSE) loss function. Nonetheless, this allows the ANN to predict the output without any regard to the probabilistic distribution associated with it. If the ANN needs to output the parameters of a predictive distribution, then the loss must incorporate these parameters so that they can affect the ANN's weight optimization process achieved through gradient descent. Therefore, Lakshminarayanan *et al*. [13] propose a negative log-likelihood loss expressed in terms of a normal distribution, as it is assumed that the observation is sampled from a Gaussian distribution:

$$Loss = -\frac{\log\left(Variance\left(x\right)\right)}{2} - \frac{\left(y - \text{Mean}\left(x\right)\right)^2}{2 \times \text{Variance}\left(x\right)} + c \qquad (5)$$

where c corresponds to all constant terms that do not affect the loss minimization process, and $Mean(x)$ is the mean and $Variance(x)$ is the variance (i.e., ML uncertainty) associated with the Gaussian distribution of the prediction for input

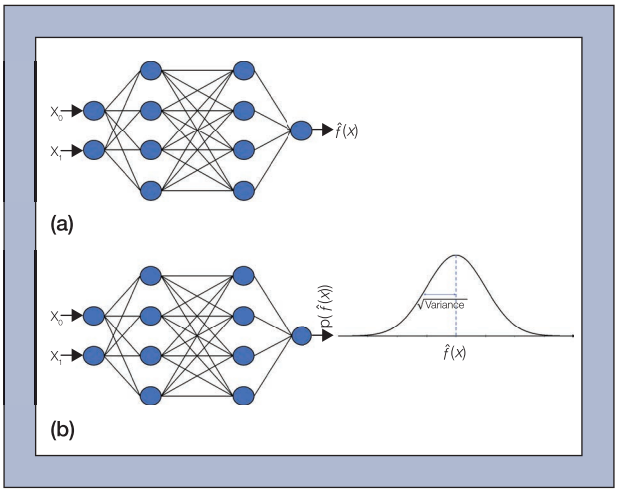

Fig. 3. (a) Typical ANN model that produces a single prediction for an input. (b) ANN model as proposed by Lakshminarayanan *et al.* [13] that produces a probability distribution for an input.

x. The log-likelihood function enables the calculation of the most likely parameters or maximum-likelihood of the joint probability of independent events. Given that observations are assumed to be independent, the maximum-likelihood reflects the most likely probabilistic distribution the observations were sampled from. The log-likelihood simplifies the estimation of the maximum-likelihood parameters as their calculation requires a differentiation that is difficult to compute. In (5), the second term produces high loss values if the difference between the estimated mean and the observation is large, which pressures the network to adjust the weights to reduce this difference. However, when this difference cannot be further reduced, then the variance in the denominator increases to compensate and reduce the loss.

Adversarial Training

Researchers have remarked that ANNs can produce widely divergent predictions for very similar inputs. For instance, objects appearing in nearly identical images may be differently classified. Hence, Szegedy *et al*. [14] proposed augmenting the training data with examples that are close to those in the training data to address this issue. These are called adversarial examples and are chosen strategically to be similar to examples from the training data yet result in increasing the loss during training. To this end, Goodfellow *et al*. [15] introduced the fast gradient sign method as a computationally fast approach to generate adversarial examples. Lakshminarayanan *et al*. [13] propose incorporating adversarial training into their probabilistic distribution prediction approach. They posit that adversarial training improves the predictive accuracy of their method.

Deep Ensemble Dropout

Ensemble learning involves training multiple models on the training dataset to improve predictive accuracy and in some cases reduce overfitting. At run time, the results of these models are fused using a variety of schemes. In addition to adversarial training, Lakshminarayanan *et al*. [13] propose using ensemble learning to further improve uncertainty estimation. To evaluate an input, every member of the ensemble makes a prediction. All predictions are averaged to obtain a Gaussian mixture distribution (given that every member predicts a Gaussian distribution). This results in a *variable-output ML model* where each member of the ensemble produces a Gaussian distribution ($\hat{p}(x)$). Hence, the aleatoric uncertainty is estimated as the average of all variance values estimated by the members of the ensemble. The epistemic uncertainty corresponds to the variance in the means estimated by every member of the ensemble. Both components can be combined to produce an overall estimation of the model's uncertainty.

Dropout Ensemble

Bachstein [16] presents an approach that combines aspects of the Monte Carlo dropout and deep ensemble methods. Just like the deep ensemble method, Bachstein proposes to modify ANNs to predict probabilistic distributions, and uses the negative log-likelihood as a loss function. However, instead of using adversarial training, the proposed approach relies on dropout to build in robustness into the network.

Uncertainty Estimation without Re-Training

The methods surveyed above require modification to the ANN design and/or training process. However, in practice there are many instances when developers employ pre-trained networks for classification or regression tasks. These networks are typically trained on large datasets, and their retraining may require vast computational resources and access to the training data, which is not always possible. For this purpose, Mi *et al*. [17] introduce an approach for uncertainty estimation on pre-trained networks that were not designed and/or trained with uncertainty approximation in mind. They define two scenarios: black-box and gray-box uncertainty estimation. In the black-box scenario, the developer has access to the trained model which however is impractical to modify or retrain. In the gray-box scenario, the developer has access to intermediate layers in the network but is unable to modify the weights by retraining. In the latter case, Mi *et al*. access feature maps produced by the layers of Convolutional Neural Networks (CNNs). CNNs can automate the process of feature engineering by automatically extracting useful features for a particular regression or classification task. Hence, the network learns which features are important for the problem in question through the training process. The feature extraction process is performed progressively through multiple convolutional layers. The output of each convolutional layer corresponds to features deemed useful through the training process. While the early layers produce general features that may generally be applicable to numerous tasks, the last layers generate specific features that are optimized for the problem under consideration. To estimate uncertainty, Mi *et al*. [17] impose tolerable perturbation on the

input for the black-box scenario or feature maps for the gray-box scenario. This technique bears similarities to adversarial training. However, instead of attempting to identify slightly different inputs that result in a large loss, they perform transformations on the inputs that only marginally modify the model's behavior.

In their study, Mi *et al*. [17] consider image processing regression tasks. Hence, they apply input transformations on the images that do not drastically affect the behavior of the model. For instance, CNNs tolerate image transformations such as rotations and flips. Therefore, these are the tolerable perturbations they consider. They were able to achieve uncertainty estimates that are comparable to those produced by the Monte-Carlo dropout method.

For the gray-box scenario, perturbations are not directly introduced at the input level. Instead, they are applied to feature maps. Hence, they apply evenly distributed Gaussian noise to feature maps to introduce tolerable perturbation. The noise is randomly sampled at run time to induce a different output upon repeating execution of the model on the same inputs. Moreover, they propose a dropout approach where features are randomly dropped from the feature maps. Intermediate layers of CNNs often carry redundant information, hence applying dropout introduces stochasticity without drastically changing the model's behavior. The ML uncertainty is calculated for both scenarios from the variance in the output that results from introducing perturbation to the inputs or feature maps.

Conclusion

The use of ML in I&M will only increase with the advent of the former. It is therefore crucial to understand how ML contributes to measurement error and how to quantify its associated uncertainty. The latter subject has only recently been studied and needs more investigation to provide sufficient confidence in future ML-based measurement instruments and methods.

References

[1] M. Vallejo, C. de la Espriella, J. Gómez-Santamaría, A. F. Ramírez-Barrera, and E. Delgado-Trejos, "Soft metrology based on machine learning: a review," *Meas. Sci Technol*., vol. 31, no. 3, Mar. 2020.

[2] M. Myslín, "Machine learning has uncertainty. design for it," Towards Data Science, Mar. 16, 2020. [Online]. Available: https://towardsdatascience.com/machine-learning-has-uncertainty-design-for-it-f015a249a444.

[3] M. Khanafer and S. Shirmohammadi, "Applied AI in instrumentation and measurement: the deep learning revolution," *IEEE Instrum. Meas. Mag*., vol. 23, no. 6, Sep. 2020.

[4] S. Shirmohammadi and H. Al Osman, "Machine learning in measurement, part 1: error contribution and terminology confusion," *IEEE Instrum. Meas. Mag.,* vol. 24, no. 2, 2021.

[5] A. Ferrero and S. Salicone, "Measurement uncertainty," *IEEE Instrum. Meas. Mag.,* vol. 9, no. 3, pp. 44-51, Jun. 2006.

[6] JCGM 100:2008, *Evaluation of measurement data – Guide to the expression of uncertainty in measurement, (GUM 1995 with minor corrections)*, Joint Committee for Guides in Metrology, 2008.

[7] L. Breiman, "Bagging predictors," *Mach. Learn*., vol. 24, pp. 123-140, 1996.

[8] C. Szegedy *et al*., "Going deeper with convolutions," in *Proc. 2015 IEEE Conf. Computer Vision and Pattern Recognition (CVPR),* 2015.

[9] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: representing model uncertainty in deep learning," in *Proc. Int. Conf. Int. Conf. Mach. Learn*., pp. 1050-1059, 2016.

[10] M. Teye, H. Azizpour, and K. Smith, "Bayesian uncertainty estimation for batch normalized deep networks," in *Proc. Int. Conf. Mach. Learn*., pp. 4907-4916, 2018.

[11] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn*., pp. 448-456, 2015.

[12] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" *Advances in Neural Info. Processing Sys*., pp. 2483-2493, 2018.

[13] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in Neural Info. Processing Sys*., pp. 6402-6413, 2017.

[14] C. Szegedy *et al*., "Intriguing properties of neural networks," in *Proc. Int. Conf. Learning Representations*, 2014.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," arXiv preprint arXiv:1412.6572, 2015.

[16] S. Bachstein, "Uncertainty Quantification in Deep Learning," Master's thesis, Universitat Ulm, 2019.

[17] L. Mi, H. Wang, Y. Tian, and N. Shavit, "Training-free uncertainty estimation for neural networks," arXiv preprint arXiv:1910.04858, 2019.

Hussein Al Osman (M'12) (Hussein.AlOsman@uottawa.ca) is currently an Associate Professor with the School of Electrical Engineering and Computer Science, University of Ottawa, Canada, where he directs the Multimedia Processing and Interaction Group and his research focuses on novel ideas in the realm of Human Computer Interaction, with particular attention to Applied AI. He received the Ph.D. degree in electrical engineering from the University of Ottawa in 2014.

Shervin Shirmohammadi (M '04, SM '04, F '17) (shervin@ieee.org) is currently a Professor with the School of Electrical Engineering and Computer Science, University of Ottawa, Canada, where he is Director of the Distributed and Collaborative Virtual Environment Research Laboratory and his research focuses on multimedia systems and networks, including measurement techniques and applied AI for networking, video streaming, and health systems. He received his Ph.D. degree in electrical engineering from University of Ottawa, Canada and currently serves as the Editor-in-Chief of *IEEE Transactions on Instrumentation and Measurement*.