Stochastic Undirected Neural Networks

Ricardo Manuel Madeira Fraga Fernandes Simões
ricardo.fernandes.simoes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2023

Abstract

Recent developments in the fields of machine learning (ML) and deep learning (DL) have led to the discovery of new models capable of outperforming older models in various tasks, such as classification, object detection, and even sentiment analysis. This work presents a proof of concept for a type of neural network called stochastic undirected neural network (SUNN), which demonstrates the ability to perform multiple tasks, such as classifying an image or generating an image conditioned on its class, due to its property of not having a specific direction for the flow of internal information within the network. Additionally, this model also exhibits generative properties and can represent uncertainty due to the incorporation of stochastic nodes in the network. Both properties are explored simultaneously on a dataset commonly used as a benchmark, revealing promising results, when compared to some reference models for specific tasks. The focus of this work is on classification and prototype generation tasks. To achieve this, we use SUNNs with fully connected layers and convolutional layers, which are compared to non-stochastic and undirected neural networks, as well as other well-known neural network models for similar applications. The architectures of stochastic undirected neural network were optimized for these tasks, aiming to achieve the best possible results.

Keywords: Neural Networks, Deep Learning, Undirected, Stochasticity, Classification, Prototype Generation

1. Introduction

In recent times, the subjects of machine Learning (ML) and deep Learning (DL) have witnessed an array of remarkable advancements, largely catalyzed by innovative neural network (NN) architectures (see Chapter 5 in [11] and Chapters 1, 5 and 6 in [6]). However, many models adopt a monolithic approach, mapping inputs to outputs via a predetermined sequence of calculations. The challenges inherent in this paradigm were highlighted in the insightful work seen in [13], prompting the conception of the undirected neural network (UNN) model as a potential solution. This innovation model has the ability to encompass and unify well-established NN-based frameworks, like the convolutional neural network - CNN (see Chapter 9 in [6]), or the biaffine model (see [4] and [10]). The model remains poised for further refinement, especially in terms of introducing **generative properties** and establishing a mechanism for **quantifying uncertainty**, which are two pivotal challenges being tackled in this work. Therefore, this work introduces the stochastic undirected neural network (SUNN), an evolved version of the UNN framework, projected to possess the essential attributes for surmounting the dual challenges outlined earlier. This stochastic infusion equips the model to confront the two aforementioned predicaments, setting the stage for an innovative approach to address these challenges.

It is not hard to think about scenarios where it is important to measure uncertainty and scenarios where generating data is also valuable. A good example of the importance of uncertainty quantification can be found in Section 1.1 in [5], wherein a scenario unfolds of a passenger that commits to a generous tip for the taxi driver if the journey culminates within a 25-minute span. In such a situation, a navigation system that proffers a choice between two routes based on absolute time projection pales in utility compared to a probabilistic one. The latter one, rooted in probabilistic inference, produces results based on a distribution of arrival times. This dynamic approach not only furnishes a refined decision-making foundation but also enables the calculation of the probability associated with securing the previously mentioned tip.

The field of data generation also holds immense potential, spanning a spectrum that ranges from enhancing training data through augmentation to crafting synthetic samples for exploration and analysis. This unveils a novel era characterized by

data-driven creativity and profound insights. Notably, certain domains exhibit a rare disparity between different classes. For instance, in financial transactions, non-fraudulent occurrences are more prevalent than fraudulent ones, as elaborated in [2]. Imbalanced datasets can introduce bias towards the most frequent class, leading it to struggle to correctly classify the minority class. This is where data generation comes into play, enabling the creation of new data samples to achieve a more balanced distribution of samples per class. This approach helps prevent the machine learning model from becoming overly biased towards predicting the majority class.

In this work, we use the *MNIST* dataset [3] for our research, in the realms of uncertainty measurement and data generation tasks, since it is used as a benchmark dataset in the research community. The present work makes contributions to both the field of uncertainty quantification and the subject of data generation. In particular, we introduce a novel model that exhibits both commonalities and distinctions when compared to classical approaches within these domains. This innovation not only expands the horizons of potential hybrid solutions but also underscores its applicability across both areas. Our contributions can be synthesized as:

- An introduction of our novel model, accompanied by a direct comparison to classical approaches.
- An application of the proposed model on the *MNIST* dataset, demonstrating its effectiveness through data synthesis and uncertainty assessment in predictions
- An open-sourced implementation in *Pytorch*, offering the research community access to our experiment code for future investigations, available in [16]
- A vast amount of future work ideas to develop SUNN models in terms of applications and training alternatives

2. Proposed Model

In general, the SUNN model has may similarities with the UNN one and differences from traditional NN models. We usually represent a SUNN model in a factor graph (FG), which can be useful for the explicit representation of modular components in NN architectures (see Section 8.4 in [1]). Additionally, instead of having a fixed order of computations, usually happening from an input to an output layer, we do some specific iterations that are slightly different. These iterations update the values of the nodes of the FG where each update is guaranteed to decrease a global energy function. The energy function is, of course, defined over the nodes of the network, accounting for different types of interactions. Updates can happen multiple times for the same node in a given pass. These updates, given by a coordinate descent algorithm update each representation node of the FG, leaving the remaining representations fixed as it will be seen further ahead.

Relating this model to some traditional ones, the Hopfield network (HN) [8], which is also an energy-based model, differs from the SUNN in training approach and determinism. While HN uses the Hebbian learning rule and is deterministic, SUNN employs gradient-based methods with backpropagation and has a stochastic component. The restricted Boltzmann machine (BM) introduces stochasticity to HN and trains via Gibbs sampling or contrastive divergence, sharing gradient-based characteristics with SUNN. Both aim for low-energy states during training and inference. The Bayesian neural network (BNN) uses a Gaussian distribution for training, balancing data fitting and prior adherence. Both have a dual loss function and share the one related to the classification task, as it will be presented later. The Perturb-and-MAP (PM) [15] aligns with SUNN in perturbation and energy minimization but varies in objective and application.

2.1. General form

We start by considering a SUNN model defined over an arbitrary FG, denoted as $\mathcal{G} = \{V, F\}$. This FG is composed by a set of variable nodes $V = \{X_i\}_{i=1}^{N}$ and a set of factor nodes F, where each factor node $f_s \in F$ is linked to a subset of variable nodes. Each variable node $X_i \in V$ is associated with a representation vector $x_i \in \mathbb{R}^{d_{X_i}}$. In Figure 1, seen below, we observe some examples of FGs that subsume some well-known architectures.

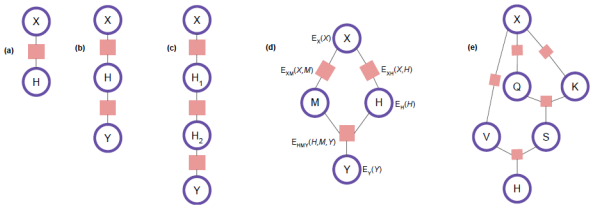

Figure 1: Different FGs for different NNs. From left to right: network without intermediate layers, undirected FFNNs with one , undirected FFNNs with two layers, undirected biaffine dependency parser, undirected self-attention. Energy labels were omitted for brevity with the exception of 4^{th} image. Figure adapted from [13]

The way we introduce stochasticity is by introducing extra nodes $W = \{N_i\}_{i=1}^{K}$ in the FG, which correspond to stochastic nodes, and each node $N_i \in W$ is associated with a representation vec-

tor $n_i \in \mathbb{R}^{d_{N_i}}$. In general, an iteration in a SUNN model is given by the formulation presented as:

$$\begin{aligned} n_1, ..., n_K &\sim p(n_1, ..., n_K), \\ (x_\pi)_* &= \arg\min_{x_\pi} E(x_1, \ldots, x_M, n_1, \ldots, n_K). \end{aligned} \tag{1}$$

In the previous equation, π refers to a particular order of updating the energy values of the non-stochastic terms. In this work, we apply some restrictions over the FG and the stochastic terms. First and foremost, we also consider one stochastic term for each hidden and output layer of the network, which means $K = M - 1$. Lastly, we assume that the global energy function is composed of three types of energy terms: the unary energy potentials and the pairwise energy terms which are assumed to be the same ones adopted for the UNN, given as (we use the same notation as in [13]):

$$\begin{aligned} E_{X_i}(x_i) &= \Psi_{X_i}(x_i) - \langle b_{X_i}, x_i \rangle, \\ E_f(x_f) &= -\langle W_f, \bigotimes_{X_j \in f} x_f \rangle. \end{aligned} \tag{2}$$

where, b_{X_i} refers to the bias term associated with the node X_i and W_f to the weights associated with a specific factor. The term Ψ_{X_i} refers to a strictly convex regularizer. It happens that with these conditions, a coordinate descent algorithm guarantees the global energy to decrease at every iteration, learning rate free [17]. Moreover, the update equations result in a chain of affine transformations followed by non-linear activations that yield the traditional computations that happen in regular NN-based models. In general, there can be many nodes linked to a factor in the network. In this work, we will focus solely on pairwise factors $f = \{X_i, X_j\}$, since this will be the scenario in the examples we will address. For the more general expression refer to Equation (21), seen in Appendix A in [13].

The final energy term is a linear energy term for the interaction of the stochastic terms with the nodes, where the stochastic terms are assumed to be independent and standard Gaussian distributed with known σ^2 variance, given below as:

$$\begin{aligned} E_{X_i N_{X_i}}(x_i) = -\langle n_{X_i}, x_i \rangle, \quad n_{X_i} \sim N(0, \sigma^2 I_{d_{X_i}}), \\ n_{X_i} \perp n_{X_j}, \forall\, i \neq j, \quad \sigma \text{ known}. \end{aligned} \tag{3}$$

Handling stochastic nodes within a NN structure has to be done carefully. In variational autoencoders (VAEs) models, it is not possible to do backpropagation without first doing what is called of reparametrization trick [9]. These models have stochastic nodes in the network that do not allow the computation of gradients, necessary to calculate and update the weights during training. In consequence, this trick has to be implemented to allow for differentiability, and it leverages a change of variable approach followed by the application of the LOTUS rule (see Section 3.8 in [6]). In the proposed model, both the sampling procedure and the noise introduction utilize an approach similar to the final result of applying the reparametrization trick, avoiding compromising the differentiation process in the network. It is crucial to emphasize that, based on our assumptions, the network does not have any stochastic nodes inside the network (between layers). Instead, and as depicted in the example seen in the second image of Figure 2, the stochasticity is parallelized. Therefore, we have some stochasticity added to a deterministic component, which is similar to the result of applying the reparametrization trick to VAEs. With this approach, the backpropagation in the SUNN model is still "forcing" low energy to align with the desired behaviour by adjusting the model's parameters, just like it happened with the UNN. It uses gradient information to navigate the energy landscape and create configurations of states that correspond to low-energy regions for correct predictions and high-energy regions for incorrect ones. This optimization process ensures that the energy function captures the relationships and constraints necessary for successful task execution. Another perspective is to recognize that the stochastic elements have shifted from the random nodes directly to the bias term present in the initial unary energy potential function, given below as:

$$\begin{aligned} E_{X_i}(x_i) + E_{X_i N_{X_i}}(x_i) = -\langle b_{X_i} + n_{X_i}, x_i \rangle \\ + \Psi_{X_i}(x_i), \quad n_{X_i} \sim N(0, \sigma^2 I_{d_{X_i}}). \end{aligned} \tag{4}$$

In the previous equation, b_i refers to the bias term of node X and $\Psi_{X_i}(x_i)$ to the regularize applied to node X. Introducing stochasticity inside the network when adding stochastic nodes in the FG is equivalent to directly injecting noise in the bias term of the unary potentials. When b_i, x_i and n_{X_i} are vectors, we can now derive the mean and variance of this sum of energy terms $E_{X_i}(x_i) + E_{X_i N_{X_i}}(x_i)$:

$$\begin{aligned} &\mathbb{E}[E_{X_i}(x_i) + E_{X_i N_{X_i}}(x_i)] \\ &= -b_{X_i}^T x_i + \Psi_{X_i}(x_i), \end{aligned} \tag{5}$$

$$\mathbb{V}[E_{X_i}(x_i) + E_{X_i N_{X_i}}(x_i)] \quad (6)$$
$$= \sigma^2 \sum_{i=1}^{d_{X_i}} x_i^2, \quad \sigma \text{ known.}$$

This leads us to conclude that $E_{X_i}(x_i) + E_{X_i N_{X_i}}(x_i) \sim N(-x_i^T b_{X_i} + \Psi_{X_i}(x_i), \sigma^2 \sum_{i=1}^{d_{X_i}} x_i^2)$. See Equations (15) and (16), in Appendix A, for the complete derivations. Therefore, the assumptions initially proposed are equivalent to defining unary stochastic terms with the previous mean and variance expressions seen in Equations (5) and (6), and as illustrated in the right-most image of Figure 2. Regarding the update expression, it is presented below:

$$(x_i)_* = (\nabla \Psi^*_{X_i})(z_i), \quad (7)$$
$$\text{where } z_i = \sum_{f \in F(X_i)} \rho_i(W_f) x_j + (b_{X_i} + n_{X_i}).$$

For a comprehensive derivation, please consult Equation (17) in Appendix A. In that equation, the term ρ_i can be interpreted as either the identity or the transpose operator. Its exact interpretation depends on the relative positions of adjacent nodes to the node X_i. It's noteworthy to mention that $\Psi(h)$ serves as a regularizer, and its corresponding activation function is $\nabla \Psi^*(t)$. Table 1 lists some prevalent activation functions. Furthermore, Figure 2 provides an illustrative example of a FG, highlighting three non-stochastic terms and two stochastic counterparts.

Table 1: Regularizers $\Psi(h)$, with corresponding activation functions $\nabla \Psi^*(t)$, where $\phi(t) = t \log(t)$.

$\Psi(h)$	$(\nabla \Psi^*)(t)$
$\frac{1}{2}\|\|h\|\|^2$	t
$\frac{1}{2}\|\|h\|\|^2 + 1_{\mathbb{R}_+}(h)$	$\text{relu}(t)$
$\sum_j (\phi(h_j) + \phi(1 - h_j)) + 1_{[0,1]^d}(h)$	$\text{sigmoid}(t)$
$\sum_j (\phi(\frac{1+h_j}{2}) + \phi(\frac{1-h_j}{2})) + 1_{[-1,1]^d}(h)$	$\tanh(t)$
$-\mathcal{H}(h) + 1_\Delta(h)$	$\text{softmax}(t)$

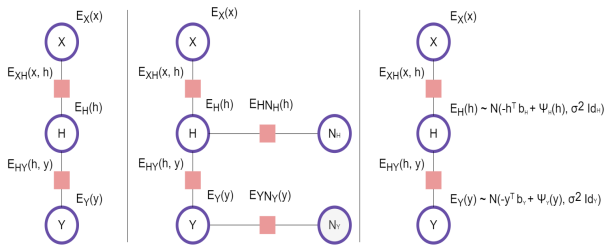

Figure 2: From left to right, FG for: network with one intermediate layer and no stochastic nodes, network with one intermediate layer and one stochastic node for each node in $V = X, H, Y$, network with redefined energy potentials

From the first to the second image, we see the incorporation of the stochastic terms in the network, and from the second to the third image we see two equivalent approaches to introducing stochasticity in the network. Note that in the third image, we are defining unary stochastic potentials. In the examples further ahead presented, we will consider the equations of the second image, *i.e.* the framework with deterministic unary and pairwise interaction energy terms, plus the stochastic energy terms. Another important remark is that the Equations (2) to (7) can be adjusted to suit the specific operations within the network. The next subsection delves into the energy expressions and updates of two scenarios explored in this work, the stochastic and undirected FFNN in sub-Section 2.2 and the stochastic and undirected CNN sub-Section 2.3.

2.2. Stochastic and Undirected Feed-Forward Neural Network

The first scenario we will explore is the stochastic and undirected FFNN, with one just one hidden layer. In this experience, the pixels of each input image x are treated as a single feature, and therefore $x \in \mathbb{R}^{784 \times 1}$. The corresponding FG is the second one (b), seen in Figure 1. Total energy function $E(x, h, y) = E$ is, in this case:

$$E = E_X(x) + E_H(h) + E_Y(y) + E_{XH}(x, h) + E_{HY}(h, y) + E_{HN_H}(h, n_H) + E_{YN_Y}(y, n_Y), \quad (8)$$

where

$$\begin{aligned} E_X(x) &= -\langle b_X, x \rangle + \Psi_X(x) = \Psi_X(x), \quad b_X = 0, \\ E_H(h) &= -\langle b_H, h \rangle + \Psi_H(h), \\ E_Y(y) &= -\langle b_Y, y \rangle + \Psi_Y(y), \\ E_{XH}(x, h) &= -\langle h, Wx \rangle, \\ E_{H_Y}(h, y) &= -\langle y, Vh \rangle, \\ E_{HN_H}(h, n_H) &= -\langle n_H, h \rangle, \quad n_H \sim N(0, \sigma^2 I_{d_H}), \\ E_{YN_Y}(y, n_Y) &= -\langle n_Y, y \rangle, \quad n_Y \sim N(0, \sigma^2 I_{d_Y}). \end{aligned} \quad (9)$$

In the two previous equations, the term σ is known. Using the expression from Equation (7), the update terms, denoted by x_*, h_* and y_* are given by:

$$\begin{aligned} x_* &= (\nabla \Psi^*_X)\left(W^T h\right), \\ h_* &= (\nabla \Psi^*_H)\left(Wx + V^T y + b_H + n_H\right), \\ y_* &= (\nabla \Psi^*_Y)\left(Vh + b_Y + n_y\right). \end{aligned} \quad (10)$$

In the previous equation, $x \in \mathbb{R}^{784 \times 1}$, $\{h, b_H, n_H\} \in \mathbb{R}^{128 \times 1}$, $\{y, b_Y, n_Y\} \in \mathbb{R}^{10 \times 1}$, $W \in \mathbb{R}^{128 \times 784}$, $V \in \mathbb{R}^{10 \times 128}$, $d_H = 128$ and $d_Y = 10$.

2.3. Stochastic and Undirected Convolutional Neural Network

The second scenario to assess is the stochastic and undirected CNN. The architecture of the CNN had already been implemented and made open-source in the UNN paper [13]. Therefore, we adapted this architecture to introduce stochasticity. In this experience, each input image x is treated as a grey-scale image, and therefore $x \in \mathbb{R}^{1\times28\times28}$. The corresponding FG is the third one (c), seen in Figure 1. The total energy function $E(x, h_1, h_2, y) = E$ is, in this case:

$$\begin{aligned} E &= E_X(x) + E_{H_1}(h_1) + E_{H_2}(h_2) + E_Y(y) \\ &+ E_{XH_1}(x, h_1) + E_{H_1H_2}(h_1, h_2) + E_{H_2Y}(h_2, y) \\ &+ E_{H_1N_{H_1}}(h_1, n_{H_1}) + E_{H_2N_{H_2}}(h_2, n_{H_2}) \\ &+ E_{YN_Y}(y, n_Y), \end{aligned} \tag{11}$$

where

$$\begin{aligned} &E_X(x) = -\langle x, b_X \otimes 1_{d_X}\rangle + \Psi_X(x) \overset{b_X=0}{=} \Psi_X(x), \\ &E_{H_1}(h_1) = -\langle h_1, b_{H_1} \otimes 1_{d_{H_1}}\rangle + \Psi_{H_1}(h_1), \\ &E_{H_2}(h_2) = -\langle h_2, b_{H_2} \otimes 1_{d_{H_2}}\rangle + \Psi_{H_2}(h_2), \\ &E_Y(y) = -\langle b_Y, y\rangle + \Psi_Y(y), \\ &E_{XH_1}(x, h_1) = -\langle h_1, C_1(x, W_1)\rangle, \\ &E_{H_1H_2}(h_1, h_2) = -\langle h_2, C_2(h_1, W_2)\rangle, \\ &E_{H_2Y}(h_2, y) = -\langle y, Vh_2\rangle, \\ &E_{H_1N_{H_1}}(h_1, n_{H_1}) = -\langle h_1, n_{H_1} \otimes 1_{d_{H_1}}\rangle, \\ &E_{H_2N_{H_2}}(h_2, n_{H_2}) = -\langle h_2, n_{H_2} \otimes 1_{d_{H_2}}\rangle, \\ &E_{YN_Y}(y, n_Y) = -\langle n_Y, y\rangle, \\ &\sigma \text{ known}, \quad n_{H_1} \sim N(0, \sigma^2 I_{d_{H_1}}), \\ &n_{H_2} \sim N(0, \sigma^2 I_{d_{H_2}}), \quad n_Y \sim N(0, \sigma^2 I_{d_Y}). \end{aligned} \tag{12}$$

Using the expression from Equation (7) the update terms, denoted by x_*, h_{1*}, h_{2*} and y_* are given by:

$$\begin{aligned} x_* &= \left(\nabla\Psi^*_X\right)\left(C_1^T(x, W_1)\right), \\ h_{1_*} &= \left(\nabla\Psi^*_{H_1}\right)\left(C_1(x, W_1) + C_2^T(h_2, W_2)\right. \\ &\quad \left.+(b_{H_1} + n_{H_1}) \otimes 1_{d_{H_1}}\right), \\ h_{2_*} &= \left(\nabla\Psi^*_{H_2}\right)\left(C_2(h_1, W_2) + \sigma_y(V)y\right. \\ &\quad \left.+(b_{H_2} + n_{H_2}) \otimes 1_{d_{H_2}}\right), \\ y_* &= \left(\nabla\Psi^*_Y\right)\left(Vh_2 + (b_Y + n_Y)\right). \end{aligned} \tag{13}$$

In this experience, each input image x is treated as a grey-scale image, and therefore $x \in \mathbb{R}^{1\times28\times28}$. We decided to keep the same shapes of the weights and biases terms as in the convolutional UNN experiment done in [13], therefore we have: $W_1 \in \mathbb{R}^{32\times1\times6\times6}$, $W_2 \in \mathbb{R}^{64\times32\times4\times4}$, $V \in \mathbb{R}^{10\times64\times5\times5}$, $b_{H_1} \in \mathbb{R}^{32}$, $b_{H_2} \in \mathbb{R}^{64}$, $b_Y \in \mathbb{R}^{10}$. In consequence, $h_1 \in \mathbb{R}^{32\times12\times12}$ and $h_2 \in \mathbb{R}^{64\times5\times5}$. The σ_y operator rolls the axis of V corresponding to y to the last position, such that $\sigma_y(V) \in \mathbb{R}^{64\times5\times5\times10}$.

3. Experiments, Results & discussion

We present in this section the results of the proposed model in the experiments we made. Our rationale here was to tune specific SUNN-type models in order to find configurations that performed well in classification and prototype generation, at the same time, providing training data from the MNIST dataset. In total we did for fine-tuning processes: two SUNN models with FFNN-type operations (one with no noise and the other with noise) and two SUNN models with CNN-type operations (one with no noise and the other with noise). The versions without noise allow us to recover the UNN model which is also a reference model for comparison with the SUNN. The loss function of the SUNN models, $L(\mathbf{x}, \mathbf{y}, \hat{\mathbf{x}}, \hat{\mathbf{y}})$, is a combination of the *Multi-Class Cross Entropy* loss (L_{MCCE}) and the *Binary Cross Entropy* loss (L_{BCE}), given by:

$$L(\mathbf{x}, \mathbf{y}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) = L_{MCCE}(\mathbf{y}, \hat{\mathbf{y}}) + \alpha L_{BCE}(\mathbf{x}, \hat{\mathbf{x}}) \tag{14}$$

where L_{MCCE} measures the dissimilarity between the predicted probabilities $\hat{\mathbf{y}}$ and the true labels $\mathbf{y}$ for each class of each sample, and L_{BCE} measures the dissimilarity between the input $\mathbf{x}$ and the reconstructed one, $\hat{\mathbf{x}}$. The weight of the *Binary Cross Entropy* loss in the global function is determined by the factor α. During training we optimize this weighting factor. For each one of these four experiments, we first ran 50 different combinations of parameters using the ranges of values provided in the table. Additional runs could not be made due to computational limitations, and also because optimal configurations had already been achieved. The entire grid of hyperparameters to tune is given below in Table 2.

Table 2: List of hyperparameters for the optimization of UNN and SUNN models

Hyperparameter	Values/Range
dropout	0.1 – 0.5
learning_rate	0-0005 – 0.1
batch_size	128, 256, 512
unn_iter	1, 3, 5, 10
unn_y_init	uniform, rand, zero
backward_loss_coef	0.01 – 1
activation	relu, tanh
noise_sd	0 – 1

The parameter *unn_iter* denotes the number of repetitions for the energy minimization process

during the forward or backward pass. We evaluated this parameter with values 1, 3, 5, and 10. Exemplifying, for the forward task, the sets are defined as $x_\pi = \{H, Y\}$ for the FFNN scenario and $x_\pi = \{H1, H2, Y, H2\}$ for the CNN scenario. When this parameter is set to 2, each element within the x_π set undergoes an additional update iteration, in the same order they appear in the set.

The parameter *unn_y_init* pertains to the initialization of Y. We explore three configurations as discussed in [13]: uniform, random, and zero initialization. The cited study highlighted promising results of random initialization, an idea we kept in mind for this study.

The *backward_loss_coef* parameter deals with penalization when generating a prototype based on a label; here we consider a range from 0.01 to 1. As for the activation function of all layers except the output one (referenced in Table 2 as *activation_function*), we decided to test both $relu$ and $tanh$ since they are commonly used in practice. The value of the standard deviation of the noise, *noise_sd* was set to range from 0 to 1, where the minimum value of 0 allows us to recover the UNN model. Given the multitude of hyperparameters and their potential values, determining optimal combinations can be intricate.

We employed *Weights and Biases* to randomly compute 50 hyperparameter combinations, uniformly sampling from our predefined ranges. These runs aimed to identify the best settings for our problem. As SUNN operates bidirectionally, we selected the best configurations based on the validation set's loss function for both forward and backward tasks and accuracy for the forward task. We also incorporated an early stopping mechanism: if the lowest loss was recorded over 10 epochs prior for both loss functions, training ceased. This strategy conserves computational resources by halting non-optimal runs. The remaining hyperparameters are common to regular NN modes, and here we used reasonable ranges usually employed in practice.

The four final configurations of the models are given below in Table 3. The models **M1**, **M2**, **M3** and **M4** are, respectively, the undirected FFNN, the stochastic and undirected FFNN, the undirected CNN, and the stochastic and undirected CNN.

Table 3: Final configurations of the UNN and SUNN models

Hyp.	M1	M2	M3	M4
dropout	0.2933	0.3714	0.2323	0.2541
learning_rate	0.0005	0.0028	0.0006	0.0016
batch_size	128	512	256	256
unn_iter	5	3	3	1
unn_y_init	uniform	uniform	rand	zero
backward_loss_coef	0.9377	0.7148	0.6845	0.7882
activation	relu	tanh	tanh	relu
noise_sd	0	0.2382	0	0.1994

These configurations were some of the best performers, against their competitive runs, and therefore they were selected for testing purposes where classification and generation capacity are assessed. For the forward task, we compare these ones against benchmark models in terms of accuracy and uncertainty quantification. As for the backward task, we visually inspect their generative capacity and analyze both the quality of the prototypes and the diversity of the multiple generated images. Regarding the forward task, quantifying certain metrics can be challenging to understand directly. Hence, we adopt the approach from [7].

We conduct two experiments: the misclassification experiment, which detects incorrect predictions using an uncertainty measure, and the OOD detection experiment, which identifies if a sample is OOD (out of domain). The first focuses on the model's error tendencies, while the second evaluates its ability to spot OOD samples. A related experiment is discussed in [12]. Performance is gauged using the AUROC (area under receiver operating characteristic) and AUPR (area under precision-recall) metrics (refer to [14]). These metrics offer a safety layer during testing, enabling the model to highlight potential errors or unfamiliar data. By setting thresholds with AUROC and AUPR, we can balance true and false positives, ensuring the model's output matches its confidence level. This boosts model reliability and offers insights for practical applications. For critical tasks like medical diagnosis or autonomous driving, flagged predictions can be set aside for human review. This cautious approach ensures informed decisions based on the model's outputs.

In summary, these metrics offer a robust framework for understanding and utilizing a model's uncertainty estimates, crucial for deploying systems in sensitive areas. We selected three distinct uncertainty metrics that can be evaluated using this methodology: entropy of the predicted probability distribution, mutual information (MI) and Jensen–Shannon divergence (JSD). For every model, each sample in the test set is passed 50 times, which allows for obtaining a predicted probability distribution through the average of these 50 predicted probability vectors.

Given that both **M1** and **M2** are FFNN-type models, they are analyzed together. We also evaluate two Bayesian neural network (BNN) models: **B1**, trained using Monte Carlo (MC) dropout, and **B2**, trained through variational inference. Both BNNs employ FFNN-type operations and maintain optimized architecture size and hyperparameters. CNN-type models are assessed collectively and in comparison to BNN models with CNN-type operations, namely **B3** and **B4**.

Starting with the FFNN-type models, Table 4 presents the results in the misclassification experiment.

Table 4: Results of FFNN-type models in misclassification experiment.

	AUROC (%)			AUPR (%)			Accuracy %
	Entropy	MI	JSD	Entropy	MI	JSD	
M1	**96.15**	-	-	38.14	-	-	**97.92**
B1	94.71	91.69	89.51	**43.25**	28.92	21.50	96.29
B2	94.66	94.52	90.65	40.44	37.13	17.04	96.42
M2	95.84	95.99	95.82	38.62	36.84	31.51	97.69

From the previous table, we see that the SUNN model (**M2)** is competitive against the UNN model (**M1**), especially in terms of accuracy, with both models surpassing the performance of the two benchmark BNN models, **B1** and **B2**. A notable observation is the marked difference between the AUROC and AUPR metrics across various uncertainty measures. This variation can likely be traced back to the significant class imbalance, which stems from the minimal misclassifications observed in all models. Turning our attention to the AUROC metric, it's clear that entropy is crucial in pinpointing positive class instances. In this scenario, entropy stands out as the more telling metric for all models, registering AUROC scores close to 95%. Interestingly, for the SUNN model (**M2**), the other metrics - MI and JSD - appear to be on par with entropy in terms of informativeness. In essence, both the UNN and SUNN models outshine the two benchmark BNN models when evaluated using the entropy metric. Additionally, the SUNN model also scores higher in the MI and JSD metrics relative to the benchmark models. However, when considering the AUPR metrics, the BNN model trained with MC dropout (**B1**) seems to have a slight edge over its counterparts, especially when assessed using the entropy metric. It achieves an AUPR score of 43.25%, edging out the SUNN model (**M2**), which posts a marginally lower score of 38.62%. Moving on to the next experiment, the results are summarized below in Table 5.

Table 5: Results of FFNN-type models in OOD detection experiment.

	AUROC (%)			AUPR (%)		
	Entropy	MI	JSD	Entropy	MI	JSD
M1	**78.23**	-	-	**77.23**	-	-
B1	77.99	76.90	74.32	76.59	75.34	71.96
B2	74.63	75.60	72.07	73.87	75.84	67.57
M2	76.27	76.53	76.37	74.99	75.61	74.85

It is noteworthy that there is a minimal discrepancy in the values recorded for AUROC and AUPR. This observation is attributed to the experiment's perfectly balanced scenario, featuring an equal number of ID (in domain) and OOD samples. Here we note that the BNN benchmark trained under MC dropout **B1** slightly outperforms both the UNN (**M1**) and the SUNN (**M2**) in maximizing the AUROC and the AUPR, especially when considering the *entropy* value. However, the difference is not too significant since the values for all models and uncertainty measures centre around 75%. Both for the AUROC and the AUPR, we see *entropy* and MI are overall the best metrics for all models when it comes to detecting outliers. For the SUNN (**M2**), the MI is slightly the better metric. Moving on to the models with CNN-type operations, the results are seen below in Table 6.

Table 6: Results of CNN-type models in misclassification experiment.

	AUROC (%)			AUPR (%)			Accuracy %
	Entropy	MI	JSD	Entropy	MI	JSD	
M3	**98.14**	-	-	36.79	-	-	98.87
B3	97.89	97.69	97.21	**38.36**	33.72	31.28	99.01
B4	97.76	97.74	97.69	35.05	29.00	26.62	**99.18**
M4	97.82	97.84	96.89	34.08	35.27	16.27	98.95

The first thing to notice is a substantial increase in the AUROC values, which consistently approach nearly 100%. This stands in stark contrast to the AUROC values recorded for FFNN-type models, as presented in Table 4. Across all models under consideration, it becomes evident that both entropy and MI exhibit a pronounced association with misclassification, consistently yielding values approaching approximately 98%. It is worth noting that the application of hyperparameter optimization has had a more substantial impact on BNNs (**B3** and **B4**), resulting in test set accuracies surpassing the 99% threshold. In comparison, the UNN (**M3**) and SUNN (**M4**) achieved test accuracies of nearly 99%, with the difference between them being relatively minor. Shifting our focus to the area under the AUPR values, one particular model stands out slightly in comparison to the SUNN (**M4**). Specifically, the BNN trained using MC dropout (**B3**) attains an AUPR of approximately 38% when employing entropy as the criterion. In contrast, the SUNN (**B4**) achieves an AUPR value of approximately 34% when considering MI. As for the OOD detection experiment using CNN-type models, the results are presented below in Table 7.

Table 7: Results of CNN-type models in OOD detection experiment.

	AUROC (%)			AUPR (%)		
	Entropy	MI	JSD	Entropy	MI	JSD
M3	86.13	-	-	85.24	-	-
B3	97.03	96.11	95.36	96.65	94.24	93.43
B4	**98.43**	97.15	97.10	**98.21**	93.80	93.62
M4	92.26	92.09	91.15	91.00	90.86	86.93

From the results of the previous case, there is a larger performance gap between the UNN (**M3**) and the SUNN (**M4**) models with the BNN ones (**B3** and **B4**), where the latter ones outperformed the first ones. The SUNN model (**M4**) was revealed

to be better for uncertainty quantification than the UNN model (**M3**), but still not as good as the remaining ones. Regarding the AUROC, the BNN trained under VI (**B4**) obtained a value around 98%, while the SUNN (**M4**) only obtained a value around 92%, when considering the entropy metric. As for the AUPR, the BNN trained under VI (**B4**) obtained a similar value of about 98%, while the SUNN (**M4**) only obtained a value of 91%, again considering the entropy metric. For both models, we see the CNN type models surpass the FFNN ones, and as stated above this way of assessing the models allows us to introduce a safety layer when the model is deployed in real life applications. Focusing on the SUNN model with CNN layers (**M4**), we decided to pick an appropriate metric and threshold for both the misclassification and OOD detection experiments. Figure 3 shows the confusion matrix obtained when considering MI with a 5×10^{-5} threshold, in the misclassification experiment for the SUNN model with CNN-type operations.

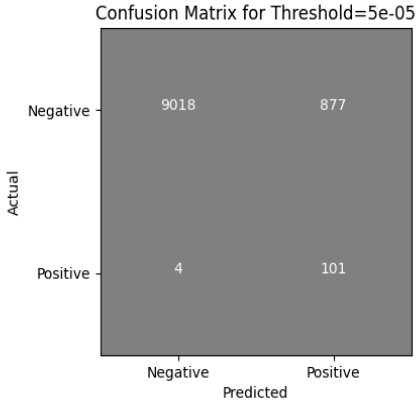

Figure 3: Confusion Matrix of the SUNN model with CNN-type operations, using MI with a 5×10^{-4} threshold in misclassification experiment.

We note that only 4 positive samples are classified as negative with this particular threshold. There is, however, a large number of negative samples being classified as positive. Nevertheless, this will not be problematic in real-life applications where it is not a problem to have many false alarms, which clearly is happening in this case. In sum, this model achieves, approximately, a TPR of 96%, a FPR of 9%, but a precision of 10%. We omitted to present the equivalent plot for the SUNN model with FFNN-type operations, although one could be obtained with higher precision than the one presented in the previous figure, but with lower TPR and higher FPR. Figure 4 shows the confusion matrix obtained when considering entropy with a 5×10^{-4} threshold, in the OOD detection experiment for the SUNN model with CNN-type operations.

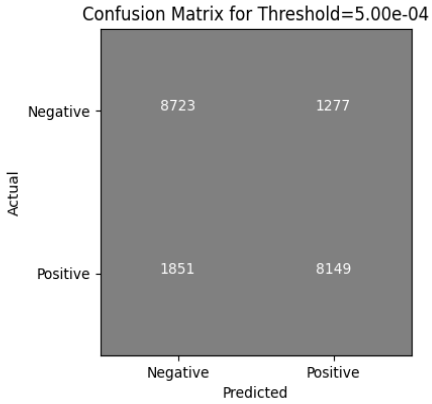

Figure 4: Confusion Matrix of the SUNN model with CNN-type operations, using entropy with a 5×10^{-4} threshold in OOD detection experiment.

This model achieves, approximately, a TPR of 81%, a FPR of 13% and a precision of 86%. These results outperform the results of the SUNN with FFNN-type operations using any threshold and metric and show that the presented model is adequate for the OOD detection experiment. One could, however, obtain better results using one of the benchmarks models, since they performed a bit better than the SUNN model. Nonetheless, the SUNN model has the ability to generate prototypes (as we will see next) besides providing these predictions we are assessing. In terms of backward predictions, we start with the prototypes generated by the UNN with CNN-type operations, which can be seen below in Figure 5.

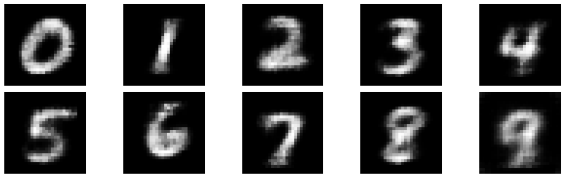

Figure 5: Digits generated by the UNN model with CNN-type operations. From left to right, top to bottom, all the digits from 0 to 9.

The UNN with CNN operations produces prototypes that accurately match their classes, reminiscent of those in [13]. However, the model is unable to generate diverse images. The SUNN prototypes are shown in Figure 6.

First of all, it is immediately evident that the digits still reflect their respective class. There is, as well, some diversity among the digits of the same class, which is more clear in classes 1 and 3. Starting with class 1, there is some diversity in the shape of the digit, more noticeable when we look at the digits from the first and last column. While the first digit has a standard or regular thickness throughout, the last one is characterized by pronounced or increased thickness at both the top and bottom

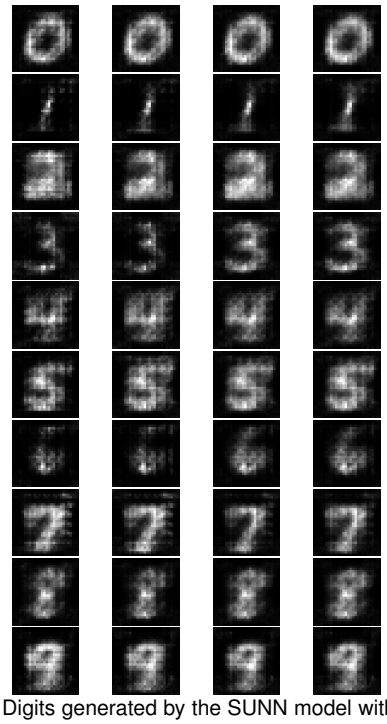

Figure 6: Digits generated by the SUNN model with CNN-type operations. From top row to bottom row: classes 0 to 9.

parts, making it appear bolder or more robust in these regions compared to the first digit. Regarding class 3. the top part of the digit of the first column seems less elongated and the bottom part is more elongated when compared to the other digits of the same class. While the other two classes also display some diversity, it predominantly manifests in pixel intensity rather than digit shape.

4. Conclusions and Future Work

In this research, we introduced the SUNN model, a framework with inherent stochastic and undirected characteristics. The SUNN excels in uncertainty estimation and prototype generation, particularly in identifying misclassifications and OOD samples when integrated with feedforward neural network (FFNN) operations. However, its performance lags behind benchmarks when combined with convolutional neural network (CNN) operations. In prototype generation, the SUNN version with CNN-type operations, is able to produce diverse and accurate digit representations. Looking forward, this work lays the foundation for further exploration of the SUNN model. Potential research directions include adapting SUNN to natural language processing tasks like dependency parsing. In fact, the integration of confidence metrics within parse trees becomes pivotal, as these metrics serve as a means to gauge whether the model has effectively captured the nuances of different linguistic characteristics and syntactic structures within a given language. The same is valid translation tasks, for example, where the model has the ability to translate between two languages and also to output some uncertainty measures regarding its predictions. In addition, there is room for future research in reimagining the architecture of the SUNN model. In our study, we introduced several constraints, including the dependency of stochastic nodes on the defined architecture and the use of normal Gaussian noise. However, relaxing some of these constraints and developing a theoretical framework to understand various architectural configurations and their consequences presents an intriguing avenue for further exploration. We also propose examining the implications of varying the number of energy minimization steps in the model. Intuitively, more energy updates could lead to better representations, especially in image generation tasks. This unexplored territory offers potential for enhancing SUNN's capabilities.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi. Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3784–3797, 2018.

[3] L. Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29:141–142, 2012.

[4] T. Dozat and C. D. Manning. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations*, 2017.

[5] O. Duerr, B. Sick, and E. Murina. *Probabilistic Deep Learning: With Python, Keras and Tensorflow Probability*. MANNING PUBN, 2020.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[7] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, Toulon, France, 2017. International Conference on Learning Representations (ICLR).

[8] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[9] D. P. Kingma. *Variational Inference & Deep Learning: A New Synthesis*. Phd thesis, Faculty of Science (FNWI), Informatics Institute (IVI), University of Amsterdam, 2017.

[10] E. Kiperwasser and Y. Goldberg. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

[11] D. J. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[12] A. Malinin and M. Gales. Predictive uncertainty estimation via prior networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7047–7058, Red Hook, NY, USA, 2018. Curran Associates Inc.

[13] T. Mihaylova, V. Niculae, and A. F. T. Martins. Modeling structure with undirected neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

[14] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[15] G. Papandreou and A. L. Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. *2011 International Conference on Computer Vision*, pages 2257–2264, 2011.

[16] R. Simões. Sunns - msc thesis, 2023. Accessed: June 23, 2023. Available: https://github.com/ricardosimoes00/Thesis.

[17] Y. Xu and W. Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6:1758–1789, 2013.

A. Detailed Expressions

General mean and variance expressions:

$$\begin{aligned}\mathbb{E}[E_{X_i}(x_i) + E_{X_iN_{X_i}}(x_i)] &= \mathbb{E}[-\langle b_{X_i}, x_i\rangle + \Psi_{X_i}(x_i) - \langle n_{X_i}, x_i\rangle] \\ &= -\langle b_{X_i}, x_i\rangle + \Psi_{X_i}(x_i) \\ &= -b_{X_i}^T x_i + \Psi_{X_i}(x_i) \\ n_{X_i} &\sim N(0, \sigma^2 I_{d_{X_i}}), \sigma \text{ known.}\end{aligned} \tag{15}$$

$$\begin{aligned}\mathbb{V}[E_{X_i}(x_i) + E_{X_iN_{X_i}}(x_i)] &= \mathbb{V}\left[-\langle b_{X_i}, x_i\rangle + \Psi_{X_i}(x_i) - \langle n_{X_i}, x_i\rangle\right] \\ &= \mathbb{V}\left[-\langle n_{X_i}, x_i\rangle\right] = \sigma^2 \sum_{i=1}^{d_{X_i}} x_i^2, \\ n_{X_i} &\sim N(0, \sigma^2 I_{d_{X_i}}), \sigma \text{ known.}\end{aligned} \tag{16}$$

General update expression

$$\begin{aligned}(x_i)_* &= \arg\min_{x_i} E_{X_i}(x_i) + \sum_{f=\{X_i,X_j\}\in F(X_i)} E_f(x_f) + E_{X_iN_{X_i}}(x_i) \\ &= \arg\min_{x_i} \Psi_{X_i}(x_i) - b_{X_i}^T x_i - \sum_{f=\{X_i,X_j\}\in F(X_i)} \langle W_f, \bigotimes_{j\in f} x_j\rangle - n_{X_i}^T x_i \\ &= \arg\min_{x_i} \Psi_{X_i}(x_i) \underbrace{-(b_{X_i} + n_{X_i})^T x_i - \sum_{f=\{X_i,X_j\}\in F(X_i)} \langle W_f, \bigotimes_{j\in f} x_j\rangle}_{-z_i^T x_i} \\ &= (\nabla\Psi_{X_i}^*)(z_i), \text{ where } \quad z_i = \sum_{f=\{X_i,X_j\}\in F(X_i)} \rho_i(W_f)x_j + (b_{X_i} + n_{X_i}).\end{aligned} \tag{17}$$