Harry Langford

Uncertainty Estimation for Spiking Neural Networks

Computer Science Tripos – Part II

Churchill College

8th May 2024

Declaration

I, Harry Langford of Churchill College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this dissertation I did not use text from AI-assisted platforms generating natural language answers to user queries, including but not limited to ChatGPT. I am content for my dissertation to be made available to the students and staff of the University.

Signed: **Harry Langford**

Date: **8th May 2024**

Proforma

Name:	**2350F**
Project Title:	**Uncertainty Estimation for Spiking Neural Networks**
Examination:	**Computer Science Tripos – Part II, 2024**
Word Count:	**12000**[1]
Line Count:	**12371**[2]
Project Originator:	**The candidate & Dr Damon Wischik**
Supervisor:	**Dr Damon Wischik**

Original Aims of the Project

This dissertation aims to investigate uncertainty estimation for spiking neural networks. I do this by implementing several methods of uncertainty estimation on spiking neural networks and comparing their performance. Specifically, I aim to invent and implement Bayesian spiking neural networks, use dropout as a Bayesian approximation for spiking neural networks and implement an efficient non-Bayesian uncertainty estimation method for spiking neural networks. The project then aims to evaluate whether the uncertainty estimates produced by these methods are meaningful or useful.

Work Completed

I successfully completed all core goals and numerous extensions. I implemented all the outlined methods and investigated behaviour in synthetic cases to eliminate methods which did not produce plausible uncertainty. I then trained all models on real-world neuromorphic data and through interventions was able to evaluate whether the uncertainty estimates were meaningful. I concluded that Bayesian uncertainty estimation is meaningful; weaker Bayesian approximations such as dropout produce meaningful uncertainty; and exploiting the timestep mechanism for non-Bayesian uncertainty estimation does not produce meaningful uncertainty.

Special Difficulties

None

[1]Word count computed by `texcount`

[2]Lines of code computed by `cloc`, and does not include comments or blank lines.

Contents

1 Introduction

This dissertation aims to unify the discipline of spiking neural networks (SNNs) with uncertainty estimation. SNNs are a type of neural network which are efficiently implementable in hardware, operate on continuous-time data and are amenable to asynchronous distributed training. I generate uncertainty estimates on SNNs and ask whether these uncertainty estimates are meaningful. After implementing selected uncertainty estimation methods on SNNs, I evaluate the performance of the resulting uncertainty-generating SNNs and determine the practical applicability of each method.

1.1 Motivation

The machine learning models ubiquitous in the modern world are compute-heavy, difficult to train and struggle to meet the high performance requirements of real-time applications. Spiking neural networks attempt to solve the first two of these problems by acting on asynchronous binarised input data, which allows for power-efficient hardware implementations and asynchronous distributed training. However, like conventional artificial neural networks (ANNs), they cannot meet the high performance requirements of real-time applications.

Networks can be made more robust to out-of-distribution data, adversarial examples and distribution shift by having them report model uncertainty. The naïve way of determining when a model is uncertain (and therefore likely to be wrong) is by inspecting the probabilities output by the model. This is known to fail in many cases since the probabilities output by networks often do not align with the true probability [Guo et al., 2017]. Models deployed in the real-world commonly face data unlike that on which they were trained due to distribution shift; in such cases model behaviour is tantamount to 'undefined behaviour'. Furthermore, it is easy to generate both in-distribution and out-of-distribution images where models output high probability predictions in an adversary-controlled manner [Goodfellow et al., 2015; Nguyen et al., 2014]. From this, we see the probabilities output by a model alone are insufficient to make it robust.

To make models robust, we need to know how confident a model is in its prediction. This can be achieved by generating uncertainty estimates. This project takes the first serious steps towards generating uncertainty estimates on SNNs. In this endeavour, I design and train uncertainty-generating SNNs for real-world data; and empirically evaluate whether the uncertainty estimates produced are meaningful.

1.2 Project goals

This project aims to discover how uncertainty estimation methods generalise to SNNs. This requires taking popular uncertainty estimation methods and implementing them on SNNs followed by a thorough evaluation to determine whether the methods are still effective. This entails two high-level goals, both of which have been achieved.

Goal 1: Generate uncertainty estimates on spiking neural networks

Generate uncertainty estimates on SNNs by: using dropout as a Bayesian approximation for SNNs; designing and implementing Bayesian SNNs; and using average-over-time SNNs (extension).

Goal 2: Evaluate the quality of uncertainty estimates produced

Thoroughly evaluate the quality of the uncertainty estimates produced. Literature in the evaluation of (or definition of) the quality of uncertainty estimates is sparse: I must therefore invent and implement sensible evaluations. This is through a combination of wide-ranging experiments and numeric metrics demonstrating similar behaviour to other uncertainty-generating models.

1.3 Achievements

The project has successfully extended the field of uncertainty estimation to encompass spiking neural networks. I have implemented all the methods outlined in the first project goal; and performed a thorough evaluation on real-world data to achieve the second project goal. I conclude that spiking neural networks are able to represent uncertainty; and that their uncertainty can be estimated by using Bayesian approximations.

In Chapter 3, I implement all the outlined methods of Bayesian SNNs, dropout as a Bayesian approximation and average-over-time SNN. I train these models on synthetic data and use them to produce plausible uncertainty estimates.

In Section 3.5, I investigate the behaviour of conventional uncertainty-generating models to establish which properties they have. Through this, I demonstrate in Section 3.6 that uncertainty-generating SNNs behave comparably to conventional artificial neural networks.

In Chapter 4, I demonstrate that the produced uncertainty estimates are meaningful by implementing them on downstream tasks and evaluating their utility.

I conclude that SNNs are able to generate meaningful uncertainty despite having only binary internal state. This conclusion is based on similar behaviour between uncertainty-generating SNNs and uncertainty-generating ANNs; and by demonstrating performance on a wide range of downstream tasks which require good uncertainty estimates.

2 Preparation

This section reviews material necessary to understand the rest of the dissertation. Section 2.1 introduces SNNs, Section 2.2 explains uncertainty and Section 2.3 reviews how to generate uncertainty estimates. I conclude with a breakdown of the project.

2.1 Spiking neural networks

Spiking neural networks are a type of neural network which are designed to operate on neuromorphic data (continuous time binary data) and to be efficiently implementable in analogue hardware. These networks are based on a unique activation layer known as a 'spike layer' which takes binary data as input, holds real state and outputs binary data at continuous times.

Definition 1: Neuromorphic data

Neuromorphic data is an event-based representation of *continuous temporal data*. For each dimension of the input, neuromorphic data has a collection of times at which an event happens. Intuitively, neuromorphic data takes fixed values at real times rather than real values at fixed times.

For data with d input dimensions over time period T, each input dimension d_i has a set of times $\{t_0, t_1, \ldots\} \subseteq [0, T]$ at which an event happens. Dimensions can have differing numbers of events.

SNN research is primarily carried out through software approximations which simulate the computation in discrete timesteps. These approximations do not directly operate on neuromorphic data but first discretise the set of possible times. I do the same.

Neuromorphic data X of dimensionality d is discretised by binning times. This creates a dataset containing f frames where each frame contains data with the domain $\mathbb{B}^d$. The i^{th} frame has value 1 at index j if and only if the input X_j spikes between time $\alpha \cdot i$ and $\alpha \cdot (i+1)$ for some constant α.

2.1.1 Formal specification

Spike layers consist of a collection of nodes. Each node takes a sequence $(x_0, x_1, \ldots)$ of values in $\mathbb{B}^d$ as input, combines it with intermediate state known as the 'membrane potential' and outputs a sequence $(s_0, s_1, \ldots)$. The output $s_t \in \mathbb{B}$ at timestep t is defined by:

$$s_t = \mathbf{1}_{\ell_t \geq 1}$$

where $\ell_t \in \mathbb{R}$ is an intermediate value of the membrane potential and $\mathbf{1}$ denotes the indicator function. ℓ_t is defined in terms of $m_t \in \mathbb{R}$, the membrane potential just before the timestep, by:

$$\ell_t = m_t + \lambda \cdot x_t$$
$$m_{t+1} = \beta \cdot (\ell_t - s_t)$$

where $\lambda \in \mathbb{R}^d$ is the weight vector of a single node and $\beta \in \mathbb{R}$ is a hyperparameter called the decay rate. Figure 2.1 visualises how the membrane potential changes with input.

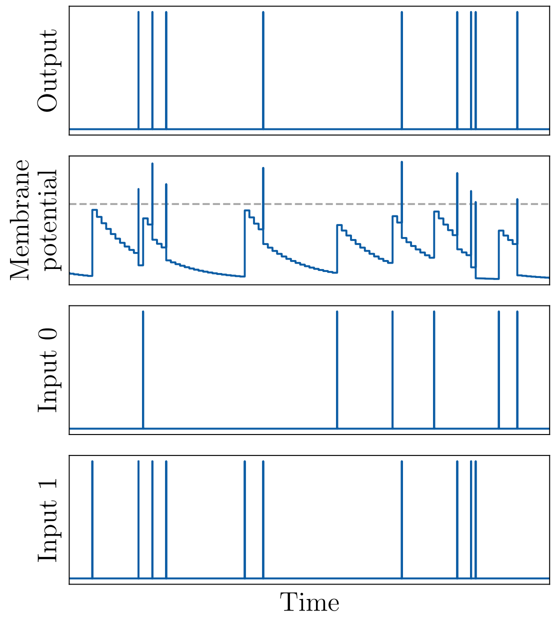


Figure 2.1: An illustration of the membrane potential for a single spiking node φ_θ against its two inputs and its output. Observe that the membrane potential decays exponentially; is increased by the dot product of the inputs with their respective weights; and if the potential is above 1, it immediately decreases by 1 and emits a spike.

$$\begin{tikzcd}
y_0 & & y_1 & & y_t \\
\varphi_{\theta^n} \arrow[u] \arrow[r, dash] & m_1^n \arrow[r] & \varphi_{\theta^n} \arrow[u] \arrow[r, dash] & \cdots \arrow[r] & \varphi_{\theta^n} \arrow[u] \\
\vdots \arrow[u] & & \vdots \arrow[u] & \iddots & \vdots \arrow[u] \\
\varphi_{\theta^1} \arrow[u, dash] \arrow[r, dash] & m_1^1 \arrow[r] & \varphi_{\theta^1} \arrow[u, dash] \arrow[r, dash] & \cdots \arrow[r] & \varphi_{\theta^1} \arrow[u, dash] \\
s_0^1 \arrow[u] & & s_1^1 \arrow[u] & & s_t^1 \arrow[u] \\
\varphi_{\theta^0} \arrow[u, dash] \arrow[r, dash] & m_1^0 \arrow[r] & \varphi_{\theta^0} \arrow[u, dash] \arrow[r, dash] & \cdots \arrow[r] & \varphi_{\theta^0} \arrow[u, dash] \\
x_0 \arrow[u] & & x_1 \arrow[u] & & x_t \arrow[u]
\end{tikzcd}$$

Figure 2.2: A diagram illustrating the computation of an SNN with $n+1$ spike layers and trainable parameters $(\theta^0, \theta^1, \ldots, \theta^n)$, each having domain $\mathbb{R}^{d\times d}$. The inputs to the network are $(x_0, x_1, \ldots, x_t)$, each having domain $\mathbb{R}^d$; and the outputs are $(y_0, y_1, \ldots, y_t)$, taking values from $\mathbb{B}^d$. Outputs are aggregated to form a prediction.

Layers consist of many nodes and can be fully connected or convolutional. I denote the function computed by a spike layer with k nodes and weights $\theta \in \mathbb{R}^{k\times d}$ by φ_θ. An SNN consists of a set of such layers stacked as in any conventional ANN; the computation of an SNN can be seen in Figure 2.2. At an implementation-level, an SNN can be thought of as a recurrent neural network which operates on binary input data and has a unique activation layer to ensure activations inside the network are binary.

An SNN can be used for classification between c classes by aggregating its outputs $(y_0, y_1, \ldots)$ where $y_i \in \mathbb{B}^d$ and interpreting the result as a probability. I use a probabilistic interpretation known as a 'rate coding', where for weights $w \in \mathbb{R}^{c\times d}$, the logits $l \in \mathbb{R}^c$ are given by $l = w\overline{y}$ and the predicted probability is the softmax of the logits. Loss is then the cross-entropy of the probability and the true class.

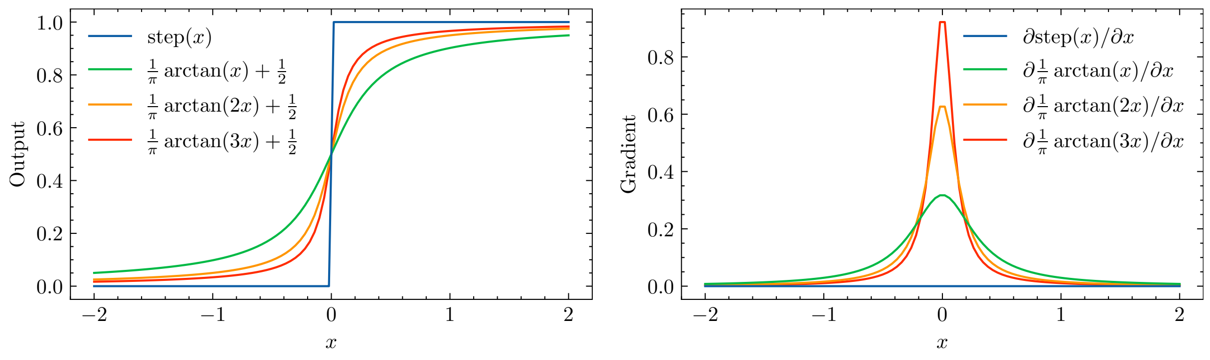
2.1.2 Training using surrogate gradients

Since $y_i = \mathbf{1}_{\ell_n \geq 1}$ is a step function, its derivative $\frac{\partial y_i}{\partial \theta} = \frac{\partial}{\partial \ell_i}(\mathbf{1}_{\ell_n \geq 1}) \cdot \frac{\partial \ell_i}{\partial \theta}$ is zero (or undefined) for all values of θ. This means we cannot directly optimise the loss if we use a rate coding. Specifically if the loss function $\mathcal{L}$ depends on the outputs of the network, we can use the chain rule to show that for any SNN, its gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ is zero:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}}{\partial y_i} \cdot \underbrace{\frac{\partial y_i}{\partial \ell_i}}_{=0} \cdot \frac{\partial \ell_i}{\partial \theta} = \sum_{i=0}^{t} \mathbf{0} = \mathbf{0}$$

To bypass this problem, the gradient $\frac{\partial s_i}{\partial \ell_i}$ is *replaced* with a surrogate gradient: the gradient of a differentiable function similar to the step function [Eshraghian et al., 2021; Zenke and

Vogels, 2021]. I use the derivative of arctan as a surrogate gradient for the step function throughout the project. Its behaviour is compared to the step function in Figure 2.3.



(a) Shifted arctan compared to the step function. **(b)** Gradients of arctan and the step function.

Figure 2.3: Comparison of arctan and its gradient to the step function. The gradient of arctan is used as a surrogate gradient for the step function when training SNNs to make $\frac{\partial \mathcal{L}}{\partial \theta}$ non-zero.

2.2 Uncertainty

To understand the project, it is essential to have a precise notion of what uncertainty is, how it can be computed, what it can be used for and how we know when we have it.

2.2.1 Defining uncertainty

We are interested in models which output two numbers: an answer and an uncertainty.

What is it? The uncertainty indicates how confident a model is in its answer. In this project, I focus on classification and therefore consider models whose answers are probabilities. For example, the output of an uncertainty-generating model in the binary classification case is of the form $p \pm y\%$, where y is the 'model uncertainty'.

How do we get it? I take a Bayesian approach to generating uncertainty. This means that I consider each model f as random and generate uncertainty on example x by producing a sample of *models* $F = \{f_1, f_2, \ldots\}$ from f and computing an ensemble of predictions $P_x = \{f_1(x), f_2(x), \ldots\}$. I report the mean of P_x as the model's overall prediction and the sample standard deviation of P_x as its uncertainty.

The pure Bayesian approach considers the parameters θ of the model f as random variables distributed according to Θ and generates F by sampling weights *i.e.* $f_i = f_{\theta_i}$ with $\theta_i \sim \Theta$. I consider *any* ensemble of models F to contain uncertainty, not just those generated by parameter randomness.

What can we do with it? Practitioners can use uncertainty estimates to determine when models are faced with particularly difficult examples, distribution shift or adversarial examples. This allows models to refuse to answer when they are likely to be wrong and achieve higher performance than they would normally be able to.

2.2.2 Measuring uncertainty

The most common evaluation method in machine learning is to log some well-known metrics and use those metrics to compare to some baseline or state-of-the-art. This is difficult for uncertainty estimation due to a lack of well-established metrics.

2.2.2.1 What makes an uncertainty estimate 'good'?

Before we discuss measuring the quality of uncertainty estimates, we must understand the abstract properties that good quality uncertainty estimates should have. After significant thought and literature review, I propose two desirable properties. The first relates to usability while the second relates to interpretability.

Can I use them? Uncertainty estimates are practically usable if they are meaningful and are correlated to the actual uncertainty of the model. Intuitively, this means that if a model outputs a higher uncertainty estimate on example a than example b, then it should genuinely be less certain about its answer for example a.

Definition 2: Internally meaningful

A model produces internally meaningful uncertainty estimates if we can determine which examples the model is confident on by comparing its uncertainty estimates to each other.

Can I understand them? We can ask that the uncertainty estimates we generate are interpretable. By defining interpretability as numerical interpretability, this can be lower-bounded in some situations.

Definition 3: Externally meaningful

A model produces externally meaningful uncertainty estimates if we can use them to construct explicit $\alpha\%$ confidence intervals for probability of the form $[p_1, p_2]$ where the 'ground truth' probability $p_{Y|X}(y \mid x)$ lies in an $\alpha\%$ confidence interval roughly $\alpha\%$ of the time: $P(p_{Y|X}(y \mid x) \in [p_1, p_2]) \approx \alpha$.

2.2.2.2 How can we evaluate uncertainty?

There exists a gap in uncertainty literature of how to empirically evaluate the quality of uncertainty estimates on probabilities in non-trivial or non-synthetic situations. I propose three high-level methods for empirical evaluation.

Perfect Information In the rare situation that the labels of a dataset are probabilities rather than classes, we can compare the confidence intervals generated by the uncertainty estimates to the 'truth'.

Method 1: Comparison to 'ground truth'

The 'quality' of uncertainty estimates are how close they are to some ground truth predictive distribution.

Embracing Empiricism I focus the evaluation on empiricism. Empiricism states that uncertainty estimates are good if they perform well on a task of interest. These tasks include differentiating between in-distribution and adversarial or out-of-distribution data.

Method 2: Performance on a downstream task

The 'quality' of uncertainty estimates are their performance on a downstream task of interest.

It behaves like uncertainty We often do not have perfect information or have a specific task of interest. In these cases we can show that uncertainty estimates are high quality by showing that they behave as we would expect high quality uncertainty estimates to. This idea shares motivation with duck typing: if uncertainty estimates look good and behave well then they are good.

Method 3: Has properties good uncertainty estimates would have

We can demonstrate that uncertainty estimates are 'high quality' by showing that they behave like high quality uncertainty estimates. This behaviour includes higher uncertainty after distribution shift or on adversarial examples.

2.2.3 Metrics for uncertainty

With the above criteria in mind, I outline several quantitative metrics which can be used to evaluate uncertainty (directly or indirectly).

How wrong we expect to be I introduce a metric which measures how inaccurate, on average, a model's probabilities are. This alone does not measure uncertainty. I conjecture that by removing the most uncertain predictions, the probabilities will improve and the change in this metric is a proxy metric for quality of uncertainty.

Metric 1: Expected calibration error (ECE)

Given a model f and dataset $\{(x_0, y_0), (x_1, y_1), \ldots\}$ of size n, create m bins $B_i = \{x_j \mid \frac{i}{m} \leq \max f(x_j) < \frac{i+1}{m}\}$ where $i \in \mathbb{N}_{<m}$.

With b_i as the proportion of examples in bin i, c_i as the mean probability predicted by examples in bin i and p_i as the accuracy of bin i:

$$\begin{aligned} b_i &= \frac{|B_i|}{n} \\ c_i &= \mathbb{E}_{x_j \in B_i} \left(\max f(x_j)\right) \\ p_i &= \mathbb{E}_{x_j \in B_i} \left(\mathbf{1}_{\arg\max f(x_j) = y_j}\right) \end{aligned}$$

I define the expected calibration error:

$$\mathsf{ECE} = \sum_{i=0}^{n} b_i \cdot \|c_i - p_i\|$$

How right we usually are Consider a metric for a model which outputs confidence intervals for probability. The metric is the difference between the claimed coverages of the claimed confidence intervals and their real coverages. Since datasets are not usually labelled with probabilities, this is usually immeasurable.

Metric 2: Expected uncertainty interval calibration error (EUICE)

EUICE is the expected difference between an α confidence interval and its coverage, summed over $\alpha \in A \subseteq (0,1]$. For each α, let $\operatorname{confint}_\alpha(x) = [p_1, p_2]$ be an α confidence interval for x. Define $q(\alpha)$ in terms of $\operatorname{confint}_\alpha(x)$ the ground truth probability $p(x)$:

$$q(\alpha) = \frac{1}{n_{\text{test}}} \sum_{x_i \in \text{test}} \mathbf{1}_{p(x) \in \operatorname{confint}_\alpha(x)}$$

Then, define:

$$\mathsf{EUICE} = \frac{1}{|A|} \sum_{\alpha \in A} |q(\alpha) - \alpha|$$

for $A = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$

2.3 Uncertainty-generating models

This section reviews the uncertainty generation methods which I transfer to SNNs. Since no method is ‘perfect’, I review and implement all of them to establish which are most performant for SNNs.

2.3.1 Formal introduction to Bayesian models

The Bayesian school of thought teaches that we should consider the parameters θ of a model f_θ as random variables drawn from some distribution Θ. With the parameters of f_θ as random variables, a prediction for a given input x is a sample from a random variable $P_x = f_\Theta(x)$. We can create the sample $S = \{p_{x,1}, p_{x,2}, \ldots\}$ drawn from distribution P_x by running the network many times. The output from an uncertainty-generating Bayesian model is the mean and elementwise sample standard deviation of S.

Theorem 1: Law of Total Probability

$$p(A \mid B) = \int_C p(A \mid B, C) \cdot p(C \mid B) \, \mathrm{d}C$$

I use the law of total probability and definition of Monte Carlo integration to prove that if the model f_θ, parameterised by weights θ, is trained on dataset $\mathcal{D}$, then the sample mean is an unbiased estimator for the posterior predictive distribution of y at x given $\mathcal{D}$, $\Pr_{Y|\mathcal{D}}(y; x)$:

$$\begin{aligned}
\Pr_{Y|\mathcal{D}}(y; x) &= \int_\theta \Pr_{Y|\mathcal{D}}(y \mid \theta; x) \cdot \Pr_{\Theta|\mathcal{D}}(\theta \mid x) \, \mathrm{d}\theta && \text{LOTP} \\
&= \int_\theta \Pr_{Y}(y \mid \theta; x) \cdot \Pr_{\Theta|\mathcal{D}}(\theta) \, \mathrm{d}\theta && \theta \text{ does not depend on } x
\end{aligned}$$

All terms on the RHS of this equation are computable. Specifically, the term $\Pr_Y(y \mid \theta; x)$ is the y-component of $f_\theta(x)$ since y given θ is independent of $\mathcal{D}$; and we can fit the distribution $\Pr_{\Theta|\mathcal{D}}(\theta)$.

A Bayesian neural network (BNN) is a network whose trainable parameters are the parameters for the Θ-distribution $\Pr_\Theta(\theta)$. Training a BNN is equivalent to fitting the distribution of its parameters. Once parameters for the Θ-distribution have been fitted, the posterior predictive distribution can be approximated by Monte Carlo sampling. Using the definition of Monte Carlo integration, we have:

$$\begin{aligned}\int_\theta \Pr_Y(y \mid \theta; x) \cdot \Pr_\Theta(\theta \mid \mathcal{D}; x)\, \mathrm{d}\theta &\approx \frac{1}{n}\sum_{i=1}^{n} \Pr_Y(y \mid \theta_i; x) \\ &\approx \frac{1}{n}\sum_{i=1}^{n} f_{\theta_i}(x)_y \\ &\approx \overline{S_y}\end{aligned}$$

where $(\theta_1, \ldots, \theta_n)$ is a sample drawn from the distribution $\Pr_{\Theta|\mathcal{D}}(\theta)$, $f_\theta(x)_y$ is the y-component of the probability $f_\theta(x)$ and $S_y = \{f_{\theta_1}(x)_y, \ldots, f_{\theta_n}(x)_y\}$ is a sample generated by running the BNN many times. The uncertainty produced by a BNN for y is the elementwise sample standard deviation of S_y.

2.3.2 Algorithms for training Bayesian models

Training a BNN is equivalent to fitting the distribution $\Pr_{\Theta|\mathcal{D}}(\theta)$. With this distribution, we can approximate the posterior predictive distribution using Monte Carlo integration. The only missing piece of the procedure is how to train a BNN. I now review algorithms which are used to train BNNs.

The Old Markov Chain Monte Carlo algorithms are (usually) guaranteed to converge to the true posterior distribution $\Pr_{\Theta|\mathcal{D}}$ in the limit. They avoid making assumptions about the underlying distribution of θ by instead performing a random walk in a sample space of size exponential in the number of parameters Θ and using the empirical distribution.

Algorithm 1: Markov Chain Monte Carlo (MCMC)

Initialisation Randomly sample a set of initial points. These points are 'accepted'.

Iteration Propose a random set of 'neighbours' to accepted points. For each proposed point, evaluate its performance on the training dataset. Either accept or reject it based on its performance.

Sampling The fitted distribution can be sampled from by non-parametrically sampling from the set of accepted points.

The New The ultimate objective when training a BNN is for the sample mean of its outputs to approximate the posterior predictive distribution. Fitting the full Θ-dimensional joint posterior distribution is computationally intractable for nontrivial models. To bypass this problem, we define a surrogate predictive distribution known as a variational distribution which makes simplifying assumptions to make it trainable (such as weights being drawn from independent distributions). The training objective now becomes maximising the similarity between the variational distribution and the posterior predictive distribution. The information theoretic metric of the similarity of one distribution to another distributions is the Kullback–Leibler (KL) divergence. Therefore, we wish to minimise the KL divergence between the variational distribution and the posterior predictive distribution. This is, once again, incomputable. However, the evidence lower bound (ELBO) *is computable* and is a constant offset from the negation of the KL divergence for a fixed model and dataset.

Thus, we train the variational distribution by maximising the ELBO by backpropagation and gradient descent.

Algorithm 2: Stochastic Variational Inference (SVI) [Hoffman et al., 2013]

Initialisation Given a network f_θ with weights $\theta \in \mathbb{R}^d$, define trainable parameters $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$ of a variational distribution q_θ which acts as surrogate to the posterior predictive distribution p.

Iteration Compute $\mathsf{ELBO} = \mathbb{E}_{y \sim q_\theta(\cdot|x)} \left(\ln \frac{p(x,y)}{q_\theta(z|x)} \right) = \ln p(x) - D_{KL}(q_\theta \| p)$ and maximise by stochastic gradient-based coordinate ascent.

Sampling The fitted distribution is $\Theta \sim \mathcal{N}(\mu, \operatorname{diag}(\sigma^2))$

The Newer Explicitly parameterising the distribution $\Pr_\Theta(\theta)$ in the weights of a neural network allows us to fit the distribution by backpropagation. This exploits gradient-based methods to efficiently explore the parameter space. However, we must force a chosen distribution (such as Gaussian) onto θ. Furthermore, for large neural networks we must assume that each weight is independent of (most) other weights to reduce complexity.

Algorithm 3: Bayes by Backprop (BBB) [Blundell et al., 2015]

Initialisation Given a network f_θ with weights $\theta \in \mathbb{R}^d$, define trainable parameters $\mu \in \mathbb{R}^d$ and $\rho \in \mathbb{R}^d$.

Iteration Generate weights θ by first sampling $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then setting $\theta = \mu + \varepsilon \cdot \ln(1 + e^\rho)$. Then for training data $(\mathbf{x}, \mathbf{y})$, compute $\nabla_{\mu,\rho} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$, backpropagate and train using gradient-based optimisers *i.e.* SGD or Adam [Kingma and Ba, 2015].

Sampling The fitted distribution is $\Theta \sim \mathcal{N}\left(\mu, \operatorname{diag}\left(\ln\left(1 + e^\rho\right)\right)^2\right)$.

2.3.3 Generating ensembles with dropout

Regularisation layers are layers which improve training behaviour. Dropout is a simple regularisation layer which randomly zeroes nodes inside a neural network and thereby encourages the network not to depend on any one feature too much [Srivastava et al., 2014]. This empirically reduces overfitting.

BNNs require twice as many parameters as non-Bayesian neural networks and have a higher compute requirement due to repeatedly sampling weights. Gal and Ghahramani noticed that many common neural network regularisation layers (including dropout) are equivalent to approximate Bayesian inference; enabling efficient Bayesian approximations.

I focus on dropout. Applying dropout before each weight layer is equivalent to training a BNN which has weights drawn from a joint Bernoulli distribution. I focus my investigation on dropout since it easily transfers onto SNNs and would be easily implementable in hardware. Networks using dropout as a Bayesian approximation are trained using conventional machine learning training algorithms.

2.3.4 Average-over-time SNN

SNNs operate on neuromorphic data containing timesteps. The outputs of an SNN at each timestep are aggregated to form a prediction for the whole input sequence. Average-over-time SNNs (AoT-SNNs) attempt to exploit the timestep mechanism to produce uncertainty in an unsound 'non-Bayesian', but 'near-Bayesian' manner.

What does it do? AoT-SNNs produce uncertainty estimates in a computationally efficient, but *mathematically unsound* way. They augment an SNN with dropout (resampled every timestep) and consider a weighted sum of the outputs at each timestep to be samples *i.e.* if a network outputs $y_i \in \mathbb{B}^d$ at timestep i then for some weights $w \in \mathbb{R}^{c\times d}$, the prediction at timestep i will be $p_i = \text{softmax}(wy_i) \in \mathbb{R}^c$. Thus, in one pass through data with $t+1$ timesteps, an AoT-SNN produces a sample $S = \{p_0, \ldots p_t\}$. It reports the mean of S as its prediction with the sample standard deviation of S as its uncertainty.

Why is this *not* Bayesian? Although an AoT-SNN f is random and contains dropout, it does not produce uncertainty on x by generating a sample of models $F = \{f_1, f_2, \ldots\}$ and running them on x. This means that it does not conform to the definition of Bayesian uncertainty as laid out in Section 2.2.1. Intuitively, spike layers pass state from one timestep to the next. This means that the predictions after timestep t are dependent on the prediction at timestep t and so cannot be viewed as being drawn from independent sampled models. The correct interpretation of AoT-SNN is as a random network which produces t probabilities when presented with an input of dimensionality $\mathbb{B}^{t\times d}$.

Why is AoT-SNN worth investigating? AoT-SNN is close to Bayesian in that not propagating removing membrane potentials would make it Bayesian. In light of this, Sun et al. attempt to provide empirical evidence that despite the lack of a Bayesian interpretation, AoT-SNN does produce meaningful uncertainty and the dependent samples do not make a practical difference. Their evaluation focuses on an empirical evaluation of the accuracy and calibration error on a randomly sampled version of the MNIST [LeCun and Cortes, 2010] dataset. These evaluation metrics are taken over the average of the predictions. This means that AoT-SNNs have *never been evaluated either from an uncertainty-centric viewpoint or on 'real' neuromorphic data.*

2.4 Requirements analysis

The project sets out to generate meaningful uncertainty estimates for spiking neural networks. This segments into two goals: generating uncertainty estimates and demonstrating that they are meaningful, as discussed in Section 1.2. Figure 2.4 contains a breakdown of the components of the project. The first project goal will be achieved by generating uncertainty on synthetic data. The second goal is achieved by the experiments on real-world data at the bottom of the figure.

2.5 Tools and techniques

2.5.1 Software engineering tools

During the first few weeks of the project, I actively searched for tools which could be useful. Many of the identified tools proved superfluous so were not used. The thinned down set of tools formed a highly optimised and very efficient development environment.

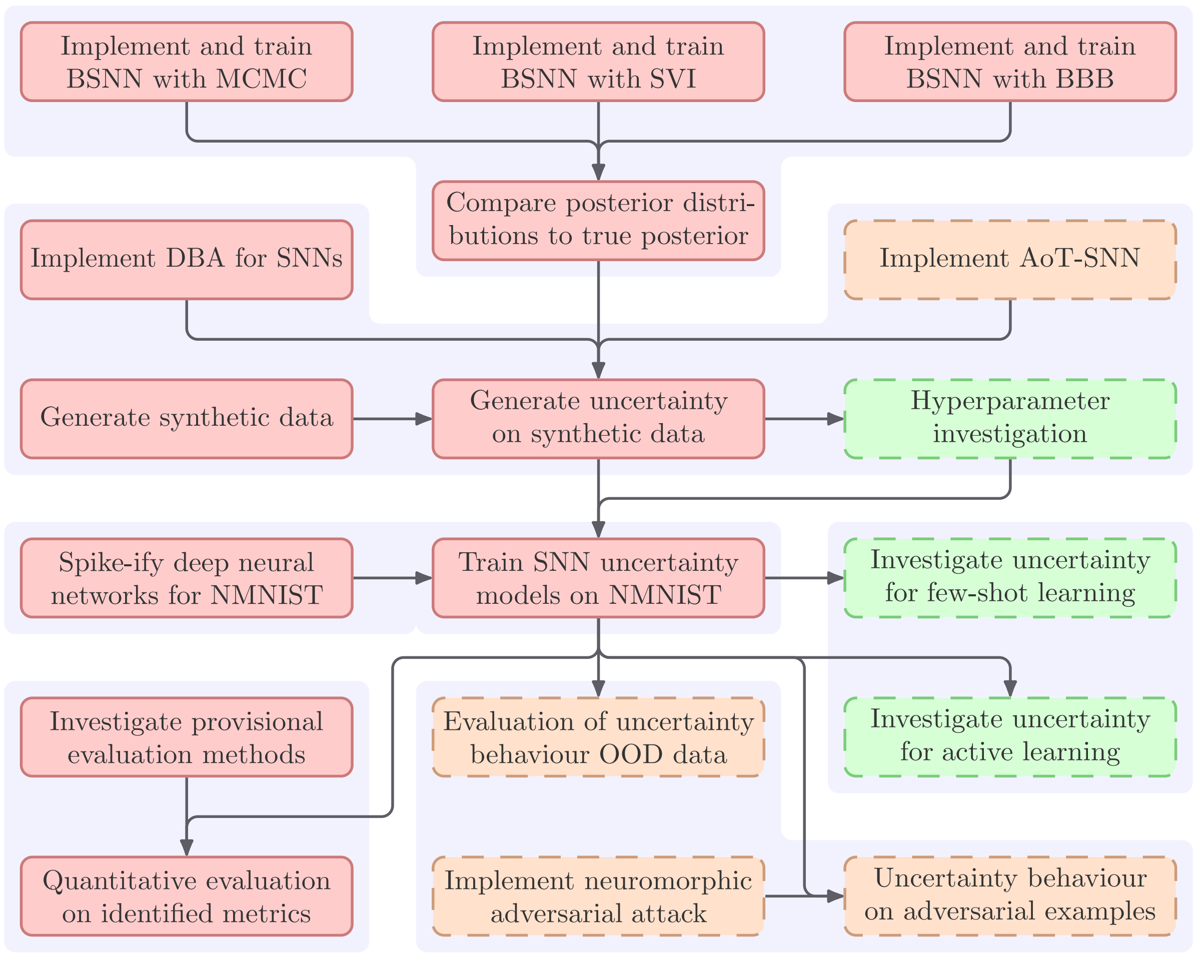

Implement and train BSNN with MCMC
Implement and train BSNN with SVI
Implement and train BSNN with BBB
Compare posterior distributions to true posterior
Implement DBA for SNNs
Implement AoT-SNN
Generate synthetic data
Generate uncertainty on synthetic data
Hyperparameter investigation
Spike-ify deep neural networks for NMNIST
Train SNN uncertainty models on NMNIST
Investigate uncertainty for few-shot learning
Investigate provisional evaluation methods
Evaluation of uncertainty behaviour OOD data
Investigate uncertainty for active learning
Quantitative evaluation on identified metrics
Implement neuromorphic adversarial attack
Uncertainty behaviour on adversarial examples

Figure 2.4: Dependency graph of the components of the project. If there is an arrow from component X to component Y, then component Y requires the completion of component X. Red indicates high priority, orange indicates medium priority and green indicates low priority. Dashed borders indicate extensions. Components which are closely related are in the same blue group. All components, except active learning are implemented. I did not implement active learning to due to sufficient results and space constraints.

Languages and Libraries Due to the large ML ecosystem in Python and my prior familiarity with PyTorch, the project was wholly implemented in Python [Van Rossum and Drake, 2009; Paszke et al., 2019]. Specifically, I was motivated by the Pyro library for deep universal probabilistic programming and the snnTorch library which implements basic components for SNNs [Bingham et al., 2019; Eshraghian et al., 2021]. I used Pytorch Lightning to abstract away unnecessary implementation detail and 'decouple the science from engineering'; I used Optuna in the early stages of the project to find and build intuition for suitable SNN hyperparameters since the literature was sparse on this; I also used Tonic to get neuromorphic datasets; and finally used TorchMetrics for common metrics *i.e.* expected calibration error [Falcon and The PyTorch Lightning team, 2019; Akiba et al., 2019; Lenz et al., 2021; Nicki Skafte Detlefsen et al., 2022].

Logging and reproducibility I logged results, hyperparameters and seeds to Weights and Biases [Biewald, 2020]; and enabled automatic code saving. This allowed me to reproduce all my experiments.

Quality of Life I used Typer with Rich tracebacks to provide an easy-to-use and trivial-to-make command line interface with readable error messages [Ramírez, 2019; McGugan and Burns, 2019].

2.5.2 Best practices

Code Style and Documentation I maintained a focus on readable, concise and understandable code. I initially made documentation conforming to the sphinx documentation style and provided comments for complicated sections of code. Due to the research nature of the project, complicated docstrings quickly became out-of-date as functions were modified. Accordingly, I stopped writing sphinx docstrings in favour of inline type hints and descriptions at use-site. This was sufficient for me to return to and quickly understand code months later. I used the yapf [Wendling et al., 2015] formatter was used throughout the project to enforce a consistent and readable coding style.

Environments Experiments were run on several GPU servers which I `ssh`'d into. Each had a dedicated Anaconda [ana, 2020] environment with Python 3.11. All environments were set up identically.

Backups I used Git to move code from my local device to GPU servers. So, code was necessarily backed up to Git regularly in small sections. This workflow meant that code in the remote repository was *always* up-to-date and that the commits stored in WandB always matched the code which was run. My personal laptop had files backed up to Git, regular full backups on a second laptop and OneDrive and backed up system files with timeshift [George, 2017].

Dissertation The dissertation was developed locally with daily Git backups. Figures were made through tikz or matplotlib with SciencePlots [Garrett, 2021].

2.6 Licences

I plan to open source my codebase. I list the licences of all third-party libraries used during the project in Table 2.1. Apache, BSD, MIT and PSF are MIT-compatible licences. Since I only use tqdm through imports and the MPL licence is weak copyleft, I am permitted to use the MIT licence provided I inform users that I use tqdm.

Licence	Library

Apache Licence	`pyro`
	`pytorch_lightning`
	`torchmetrics`

BSD Licence	`mpmath`
	`numpy`
	`sklearn`
	`scipy`
	`torch`
	`torchvision`

MIT Licence	`optuna` `rich`

Licence	Library

MIT Licence	`scienceplots`
	`snntorch`
	`tabulate`
	`tonic`
	`typer`
	`wandb`
	`pytorch-cifar`

MPL 2.0 Licence	`tqdm`[a]
PSF Licence	`matplotlib`

[a] `tqdm` is MIT licenced, but contributions by the project maintainer are MPL licenced

Table 2.1: Table showing the licences of third-party libraries used during the project.

2.6.1 Starting point

I was familiar both with Python and PyTorch, but had no experience with SNNs or BNNs. My only familiarity with uncertainty estimation was from the IB Data Science course. There are *separate* open source libraries for both SNNs and BNNs; however there is no code using them in conjunction: nor did I find evidence that anyone *has ever used them in conjunction.*

3 Implementation

The heart of the project is the problem of generating meaningful uncertainty estimates for spiking neural networks. I start by outlining the implementation in Section 3.1 and with a repository overview in Section 3.2. The implementation begins in Section 3.3 by investigating how to train Bayesian SNNs and generating uncertainty estimates in a simple case (Section 3.4). In Section 3.5, I investigate methods which can infer that uncertainty estimates are not meaningful; and in Section 3.6 I prepare for the evaluation by implementing my uncertainty-generating methods on large models, training on a large dataset and using the methods from Section 3.5 to filter models which definitely do not produce meaningful uncertainty.

3.1 Implementation overview

This section clarifies the implementation strategy. I take the project's high-level goal and decompose it into necessary yet achievable sub-goals. I connect these sub-goals back to the project by relating them to the section where they are achieved.

Achieving the project's high-level goal of generating meaningful uncertainty estimates on SNNs requires generating uncertainty estimates and establishing that these uncertainties are meaningful. This aligns with the project goals outlined in Section 1.2.

3.1.1 Generating uncertainty estimates

Bayesian SNNs are a novel and unique architecture which I define in Section 3.3.1. Since they are novel, they have never been trained before. Training Bayesian SNNs with gradient-based BNN training algorithms such as SVI or BBB would mean using a surrogate gradient to fit a surrogate distribution. This double-approximation may be imprecise and make gradient-based training algorithms ineffective. I investigate which training algorithms are able to train Bayesian SNNs in Section 3.3.2. I perform this investigation in a two-node setting where the Bayesian posterior weight and posterior predictive distributions are computable by computational Bayes. This means there is a ground truth to compare against.

I implement all the uncertainty-generating methods on SNNs and investigate their behaviour on a synthetic dataset in Section 3.4. Then, in Section 3.4.1, I formally define how to use dropout as a Bayesian SNN approximation and define an AoT-SNN. With this knowledge, I can implement spiking multilayer perceptrons which employ these methods and train them on a simple synthetic dataset in Section 3.4.2. I then achieve the first project goal by using these models to generate uncertainty estimates. In preparation for training larger models, the section concludes with a sensitivity analysis in Section 3.4.3 where I investigate the relative hyperparameter importances.

3.1.2 Demonstrating meaning

I require a method to provisionally evaluate the quality of uncertainty estimates during my implementation. However, there is a gap in the literature on empirical methods for evaluating the quality of uncertainty estimates. In Section 3.5, I investigate how to provisionally evaluate uncertainty estimates by working in a perfect-information setting and comparing the conclusions drawn through different methods to a ground-truth. Through this, I find several key properties common to models producing meaningful uncertainty estimates. In Section 3.6, I use these properties to quickly discount models which definitely do not produce meaningful uncertainty estimates.

To determine that my methods are able to produce meaningful uncertainty, I must apply them to large models and investigate their behaviour on a large dataset. In Section 3.6, I do the first part of this: applying my methods to large models, training them on many values of the sensitive hyperparameters identified in Section 3.4.3 and using the provisional comparison methods from Section 3.5 to determine which models are most likely to produce meaningful uncertainty. I achieve the second project goal in Chapter 4 by evaluating the uncertainty estimates these networks produce on downstream tasks.

3.2 Repository overview

A directory tree showing the main components of the repository is given in Figure 3.1.

Directory	**Description**	**LOC**
`snn_uncertainty/`	Main codebase	**12371**
`exploration/`	Exploratory code	**3537**
`experiments/`	Experiments and training	**6122**
`bsnn_training/`	Training Bayesian SNNs (§3.3.2)	**660**
`mwe/`	Minimal working examples (§3.4)	**388**
`metrics/`	Investigation of metrics (§3.5)	**1391**
`cifar10/`	Uncertainty on CIFAR10 (§3.5.2)	**770**
`synthetic/`	Metrics on synthetic data	**621**
`nmnist/`	NMNIST experiments (§3.6)	**3197**
`train/`	NMNIST training code (§3.6)	**909**
`metrics/`	Provisional evaluation code (§3.6)	**299**
`ood/`	OOD experiments (§4.1)	**1005**
`adv/`	Adversarial experiments (§4.2)	**782**
`kshot/`	k-shot learning (§4.3)	**396**
`utils/`	Utilities	**924**
`datautils/`	Dataloaders for neuromorphic data	**602**
`layers/`	Custom dropout and Bayesian layers	**164**
`models/`	Models definitions	**1788**

Figure 3.1: A directory tree for the main components of the repository, with the filenames on the left and descriptions and lines of code in each directory on the right.

I used 146 lines of open source (MIT Licence) model architecture definitions (ResNet18, MobileNetV2 and VGG16) from the github repository `pytorch-cifar`, corresponding to roughly 1% of the source code in my project. I wrote all code pertaining to spiking neural networks, Bayesian neural networks or uncertainty estimation; and all code in the repository other than those 146 lines of architecture definitions.

Since the project grew larger than expected, exploration and many early experiments did not make it into the dissertation. These investigated properties of uncertainty-generating SNNs in synthetic situations and were superseded by experiments on NMNIST in Chapter 4.

3.3 Bayesian spiking neural networks are trainable

In this section, I transfer the most mathematically well-founded and well-known method of extracting uncertainty estimates from neural networks onto SNNs. This involves making the first implementation of a Bayesian SNN. Due to this novelty, there is no empirical evidence that standard BNN training algorithms will remain effective at training them. I therefore define minimal Bayesian SNNs and empirically verify that the algorithms discussed in Section 2.3.2 are able to train them.

3.3.1 Mathematically formalising Bayesian SNNs

A Bayesian SNN (BSNN) is a neural network composed of Bayesian spike layers, which are spike layers (Section 2.1.1) where the weights are not constants, but random variables sampled from a trainable distribution at the first timestep. A visualisation of Bayesian spike layers can be seen in Figure 3.2, which shows an n-depth spiking multilayer perceptron acting on an input with 3 timesteps. Notice that weights are shared between timesteps.

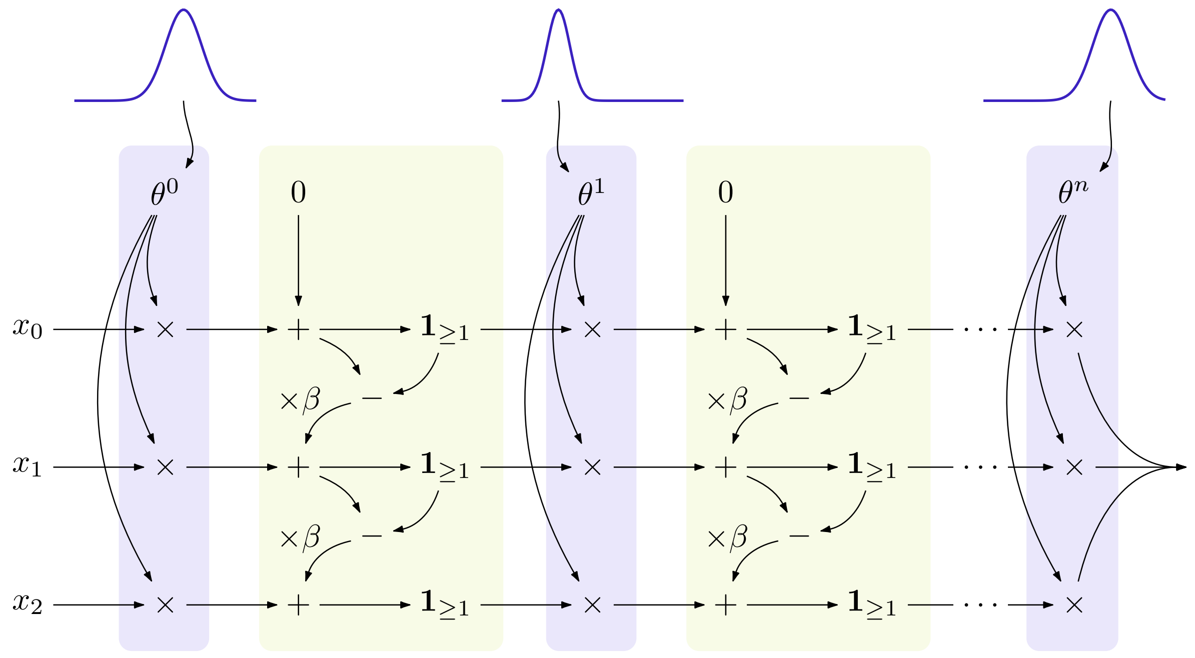


Figure 3.2: Visualisation of a BSNN acting on an input with 3 timesteps. Time flows downwards and the forward pass moves to the right. Bayesian layers are shaded blue, while spiking layers are shaded green. Weights are sampled once per layer and then shared across all timesteps. The outputs from each timestep are aggregated to form a prediction.

3.3.2 Modern training algorithms are effective at training BSNNs

Machine learning practitioners choose training algorithms based almost exclusively on their previous performance on similar problems. A BSNN is a unique architecture with significant differences to conventional BNNs. This means that the empirical evidence of an algorithm being effective at training conventional BNNs does not guarantee that this algorithm will also be effective at training BSNNs. Gradient-based algorithms applied to BSNNs would use a surrogate gradient to fit a surrogate distribution. The surrogate gradient and variational distribution approximations may not be orthogonal and may interfere with the validity of each other.

The approach In this section, I investigate three training algorithms and determine which, if any, are able to train BSNNs. I consider MCMC, SVI and BBB. I start by performing a preliminary investigation in a two-node setting. This setting is carefully designed such that the posterior distributions $\Pr_{\Theta|\mathcal{D}}$ and $\Pr_{Y|\mathcal{D}}(y;x)$ can be easily computed. I then implement a Bayesian spiking multilayer perceptron and establish how well the candidate algorithms perform in this scenario. At this stage, I use real data (*i.e.* $\in \mathbb{R}^d$) (rather than neuromorphic data) to obtain more interpretable results.

Dataset 1: Binary Classification Dataset

Let the dataset $\mathcal{D}$ contain 100 examples, where each example $((x_{i,0}, x_{i,1}, \ldots), y_i)$ consists of 50 datapoints $x_{i,j} \in \mathbb{R}$ and a class $y_i \in \mathbb{B}$. Examples are generated according to the following procedure:

$$y_i \sim \text{Bernoulli}(0.5)$$
$$x_{i,j} \sim \mathcal{N}(\lambda_i, \sigma^2)$$

where λ_i is the mean of x_i and is normally distributed with standard deviation 0.2 and class-dependent mean μ_{y_i} for $(\mu_0, \mu_1) = (-0.3, 0.4)$:

$$\lambda_i \sim \mathcal{N}(\mu_{y_i}, 0.2^2)$$

I use a two-node BSNN with two weights w_0 and w_1, which outputs two spike trains $s_0, s_1 \in \mathbb{B}^{50}$. I use the simple probabilistic interpretation that the predicted probability of class 1 is given by $\frac{\|s_1\|}{\|s_0\|+\|s_1\|}$. This network is trained using MCMC (with the No-U-Turn-Sampler [Hoffman and Gelman, 2014]), SVI and BBB. All algorithms are initialised with weights w_0, w_1 distributed according to the unit normal, $\mathcal{N}(0, 1)$.

The posterior distribution of the weights after training this network is given in Figure 3.3. Notice that all algorithms converged to similar approximations of the weights, however MCMC converged to a much looser approximation. In Figure 3.4, I visualise the posterior predictive distribution of the models after being trained with each training algorithm. All models converged to suitable approximations of the Bayesian posterior predictive distribution, but SVI and BBB converged to better approximations than MCMC.

These results suggest that SVI and BBB are not disrupted by the surrogate gradient approximation present in BSNNs. However, MCMC is struggling to train a two-node BSNN. Since MCMC does not exploit gradient information, it scales poorly. For these reasons, I conclude that MCMC is unlikely to be effective in a high-dimensional space and disregard it as a possible BSNN training algorithm.

3.4 All methods generate plausible uncertainty

Now that I have shown that it is possible to train BSNNs, I investigate whether SNNs actually have uncertainty through a minimal working example. The binary activations inside SNNs could conceivably discretise uncertainty or negate any stochasticity in the network, rendering attempts to extract uncertainty futile. This concern must be remedied before testing on non-synthetic data, where issues may be down to any number of variables. I therefore devise an easily visualisable dataset which contains regions with differing levels of probability and uncertainty.

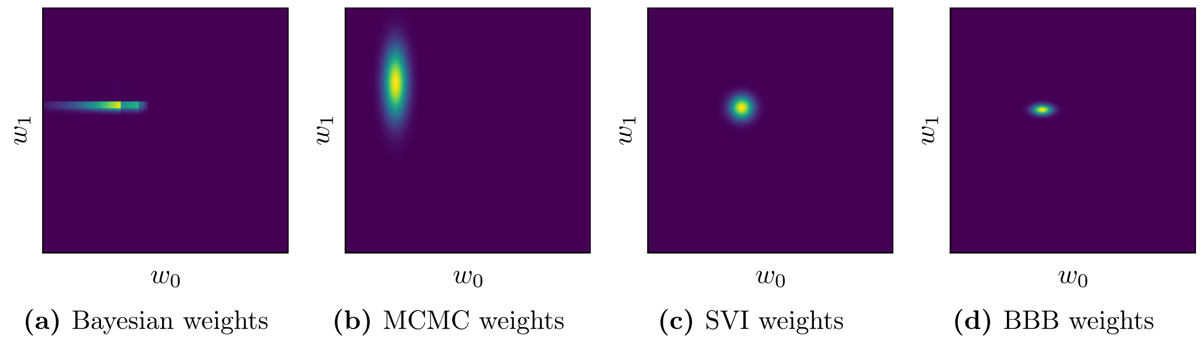


Figure 3.3: A comparison of the posterior distribution of the weights learnt via different training algorithms to the Bayesian posterior distribution. Observe that SVI and BBB converged to close approximations of the Bayesian posterior; while MCMC converged to a poor approximation.

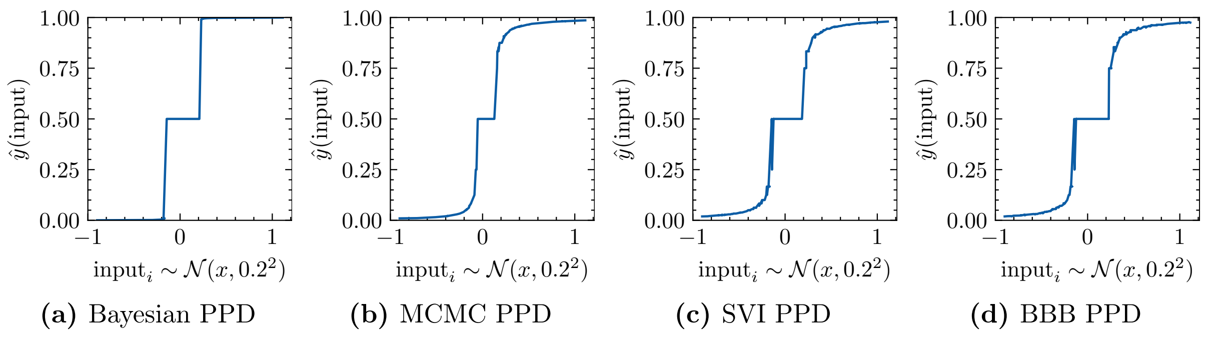


Figure 3.4: Comparison of the Bayesian predictive posterior distribution (PPD) to the PPDs from models trained with MCMC, SVI and BBB. All algorithms learnt sensible approximations. However, the PPD learnt by MCMC was furthest from the Bayesian PPD.

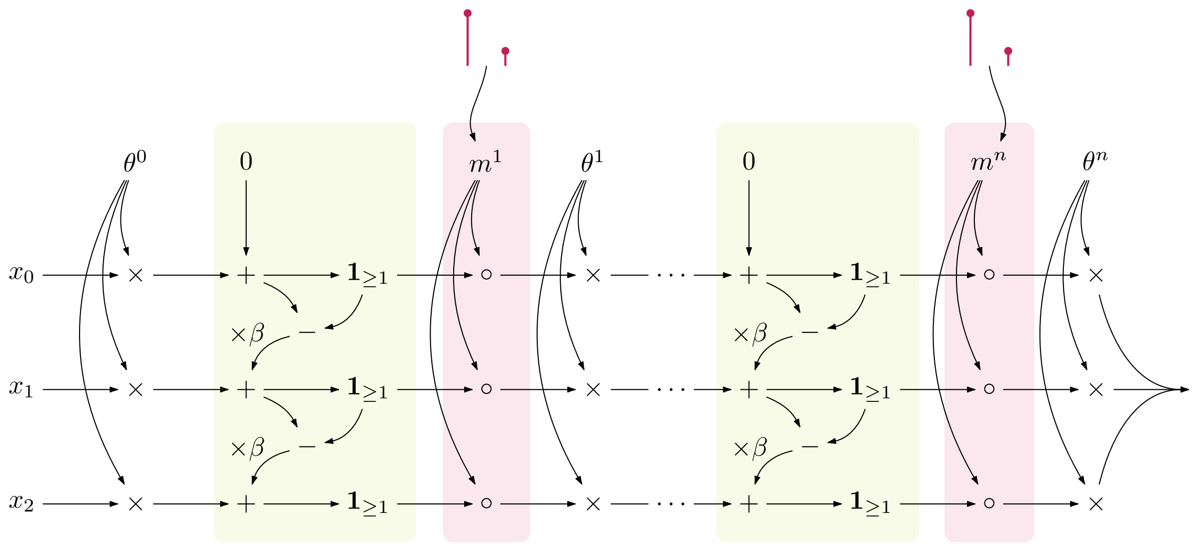
3.4.1 Mathematically formalising DBA for SNNs and AoT-SNNs

The rest of this chapter uses all four uncertainty-generating models. Accordingly, I show how an SNN can use dropout as a Bayesian approximation (DBA) in Figure 3.5a. Dropout layers are applied before every weight layer (except the input).

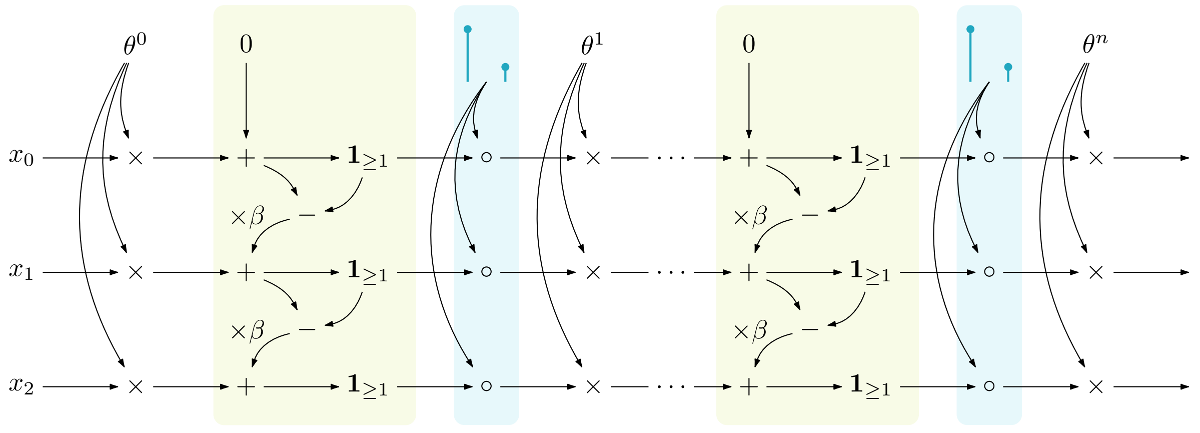
Figure 3.5b visualises the computation of an AoT-SNN. Notice that samples generated for each input $(x_0, x_1, \ldots)$ are not independent. This means that AoT-SNN takes the mean of a dependent, non-identically distributed set of samples.

3.4.2 Spiking multilayer perceptrons produce sensible uncertainty

By generating uncertainty estimates on a synthetic dataset, I achieve the first project goal in this section. I use spiking multilayer perceptrons and a synthetic binary classification dataset. The dataset is designed such that different regions should have all combinations of low/high and low/high predictive probability.



(a) Diagram of an SNN using dropout as a Bayesian approximation. Spiking layers are green, while layers required for DBA are red. Each mask $m^i \in \mathbb{B}^d$ is drawn from a multidimensional Bernoulli distribution once, elementwise multiplied with the output of the i^{th} layer and re-used for all timesteps. Uncertainty estimates are generated by running the whole network many times and considering the sample standard deviation of its outputs.



(b) Diagram of an average-over-time SNN. Spiking layers are green, while layers specific to average-over-time SNN are cyan. Dropout is re-taken each timestep and elementwise multiplied with the output of the i^{th} layer at that timestep. The outputs at different timesteps are not i.i.d. and do not have a Bayesian interpretation (see Section 2.3.4). We take their sample standard deviation as uncertainty.

Figure 3.5: A comparison of DBA for SNNs and AoT-SNN.

Dataset 2: 2D overlapping dataset

Let the dataset $\mathcal{D} = \{((x_{0;0}, x_{0;1}, \ldots), y_0), ((x_{1;0}, x_{1;1}, \ldots), y_1), \ldots\}$ where $y_i \in \mathbb{B}$ is generated according to the following distribution:

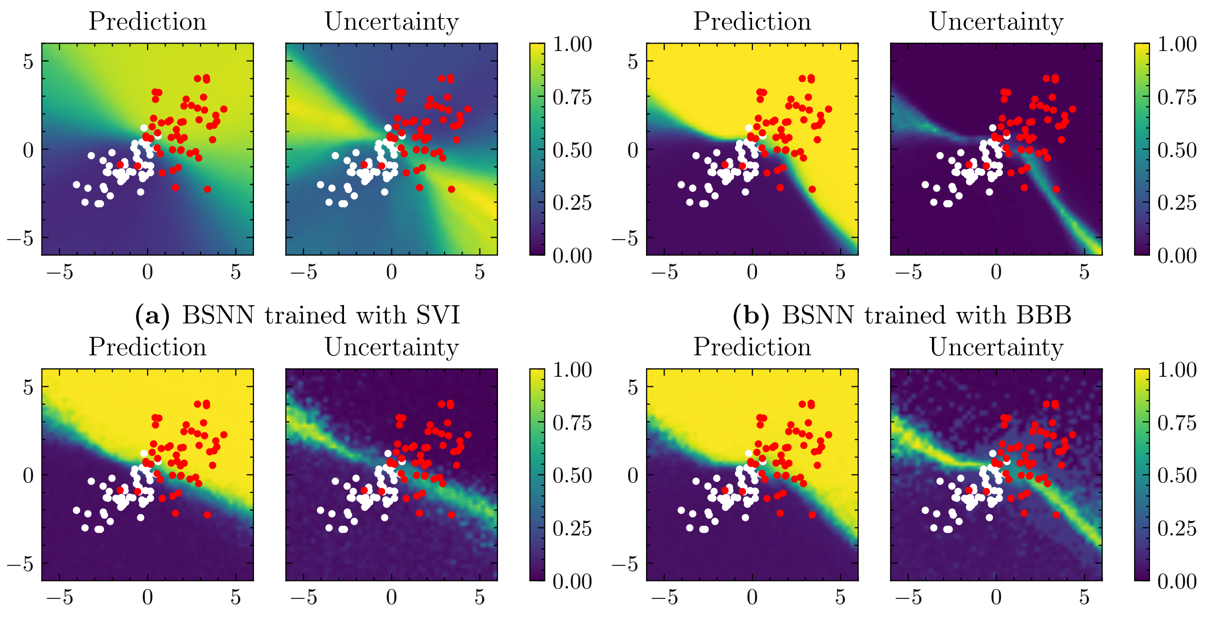
$$y_i \sim \text{Bernoulli}(0.5)$$

and $x_{i,j}$ is generated according to:

$$x_{i,j} \sim \mathcal{N}\left(\mu_{y_i}, \Sigma_{y_i}\right)$$

where the means μ_0, μ_1 and the covariances Σ_0, Σ_1 are:

$$\mu_0 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \qquad \mu_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \qquad \Sigma_0 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \qquad \Sigma_0 = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 2 \end{bmatrix}$$



(a) BSNN trained with SVI **(b)** BSNN trained with BBB **(c)** DBA **(d)** AoT-SNN

Figure 3.6: Visualisation of the predictive probabilities and uncertainties for uncertainty-generating spiking multilayer perceptrons using different methodologies on a 2D dataset of size 50. Models have 3 hidden layers of width 32 and were trained for 50 epochs with hyperparameters finetuned for cross-entropy loss. All models converge to plausible solutions. All models also converged to solutions with high probability, low-uncertainty regions in which there was no data, potentially allowing adversarial examples.

Achieving the first project goal We see in Figure 3.6 that all models were able to converge to plausible solutions. By generating uncertainty estimates on SNNs, I have now successfully achieved the first project goal.

3.4.3 BSNNs are very sensitive to hyperparameters

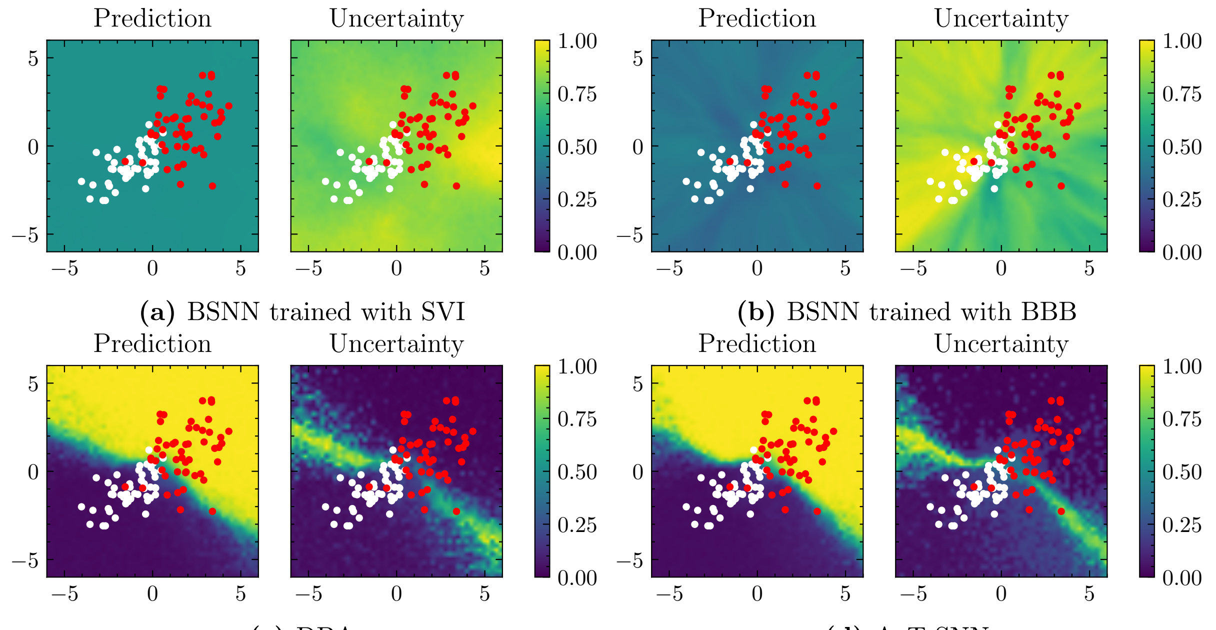
To build intuition for the interaction between SNNs and uncertainty-generating networks, I run a sensitivity analysis to determine which hyperparameters have the largest effect on performance. Through this, I pre-emptively solve hyperparameter problems before scaling up to larger datasets where a thorough exploration would be computationally infeasible.

For each model, I used Optuna [Akiba et al., 2019] to run a hyperparameter by training 250 models with the objective of minimising the cross-entropy loss on the 2D binary classification dataset. I investigated the dropout probability p, the initial standard deviation of the BSNN weights σ_0, initialisation strategies for BSNN means μ, the decay rate β (Section 2.1.1), the learning rate and the number of timesteps. Hyperparameters were discretised to reduce the search space. I show the relative hyperparameter importances in Table 3.1.

		Relative Importances					
Network	Impact on Loss	p	σ_0	μ	β	*learning_rate*	*num_steps*
BSNN + SVI	High	-	0.839	0.048	0.042	0.056	0.015
BSNN + BBB	High	-	0.996	0.000	0.003	0.001	0.000
DBA	Low	0.837	-	-	0.012	0.109	0.042
AoT-SNN	Low	0.195	-	-	0.018	0.778	0.010

Table 3.1: Table shows the relative importances of hyperparameters on cross-entropy. SVI requires a very high initial standard deviation σ_0; and BBB requires a very low initial standard deviation σ_0 to achieve good performance. DBA and AoT-SNN are insensitive to differing dropout probabilities p.

I find that BSNN initialisation of σ has a large impact on convergence. All other networks are relatively insensitive to all other hyperparameters. With this in mind, when I scale to larger datasets, I choose sensible arbitrary values for unimportant hyperparameters and explore options for the important hyperparameters.



(a) BSNN trained with SVI **(b)** BSNN trained with BBB **(c)** DBA **(d)** AoT-SNN

Figure 3.7: Visualisation of the predictions and uncertainties for uncertainty-generating spiking multilayer perceptrons arbitrary hyperparameters. Notice that both SVI and BBB struggled to train the BSNN.

Figure 3.7 visualises the sensitivity of methods to their hyperparameters. It shows the posterior predictive distributions learnt by each model if the most important hyperparameters are replaced with intuitive, non-finetuned values. AoT-SNN and DBA converged to sensible uncertainties for all hyperparameter configurations.

I draw two conclusions from this sensitivity analysis: BSNNs are highly sensitive to σ_0; and all models are insensitive to SNN-specific hyperparameters (decay rate β and *num_steps*). With this knowledge, I can pre-empt hyperparameter issues and focus later investigation into hyperparameters which are most likely to be problematic.

I am now able to choose which BSNN training algorithm to use for the rest of the project. In Figure 3.6, we saw that SVI converged to a solution with low predictive probabilities even in areas of high density. This indicates that SVI was struggling to train networks even in low-dimensional spaces. I believe that an algorithm struggling in a thousand-dimensional space will be ineffective in a million-dimensional space. Since SVI suffers from the same hyperparameter sensitivity as BBB, it seems preferable to train BSNNs through BBB. For these reasons, I train subsequent BSNNs through BBB.

3.5 Deriving uncertainty estimation properties on ANNs

With the first project goal achieved, the remainder of this chapter lays the groundwork for achieving the second project goal of evaluating the quality of the uncertainty estimates produced by BSNNs, DBA on SNNs and AoT-SNN. In this section, I hypothesise properties that uncertainty-generating models should have. These properties should be model-agnostic. We know how to produce meaningful uncertainty estimates on ANNs, so I investigate whether ANNs which produce meaningful uncertainty have these properties. I then propose quantities which measure these properties and allow me to provisionally determine whether models are likely to be meaningful. I later use these to quickly filter out models which are unlikely to produce meaningful uncertainty estimates.

3.5.1 A hypothesis on the behaviour of uncertainty estimates

If a model outputs a probability and an uncertainty, then we want the probability to be better for examples with lower uncertainty. ECE is a widely accepted proxy metric used to measure how good the probabilities output by a model are (see Section 2.2.3).

The hypothesis I hypothesise that if a model generates internally meaningful uncertainty estimates, then the ECE of low uncertainty examples will be lower than the ECE of examples with high uncertainty.

We can visualise the interaction between ECE and uncertainty by plotting a curve of ECE against coverage based on uncertainty estimates. For a dataset of size n, I order the datapoints by increasing uncertainty and plot the ECE of the first $c \cdot n$ datapoints as a function of $c \in (0, 1]$. This curve is computable on datasets whose labels are the ground truth classes: the common case.

I propose two quantitative measures to apply to this curve which I hypothesise will be meaningful estimators for the quality of uncertainty. The first is ECE ratios: for some coverage c, I hypothesise that the ratio ECE@c/ECE@1 will measure how much better the probabilities below coverage c are than typical probabilities. Secondly, I hypothesise that the area under calibration curve (AUCC) will be meaningful: equivalent to a sum of how 'good' probabilities are weighted inversely proportionally to their uncertainty. AUCC heavily penalises low quality probabilities at low c. Finally, note that if the ECE vs coverage curve often has a negative gradient, then the uncertainty estimates produced are not internally meaningful.

3.5.2 The core hypothesis holds on ANNs for CIFAR10

To empirically verify the hypothesis that models which produce meaningful uncertainty estimates have low ECE for examples with low uncertainty, I test many models on a real dataset and compare the true quality of their uncertainty estimates. Furthermore, I visualise their ECE vs calibration curves and establish whether the gradient is positive at all points.

Verifying this hypothesis requires having models which produce meaningful uncertainty estimates. I avoid bootstrapping by working in a perfect-information situation and using the CIFAR10-H probabilities to compute EUICE. EUICE is a mathematically sound aggregate measure of how externally meaningful uncertainty estimates are.

Dataset 3: CIFAR10-H

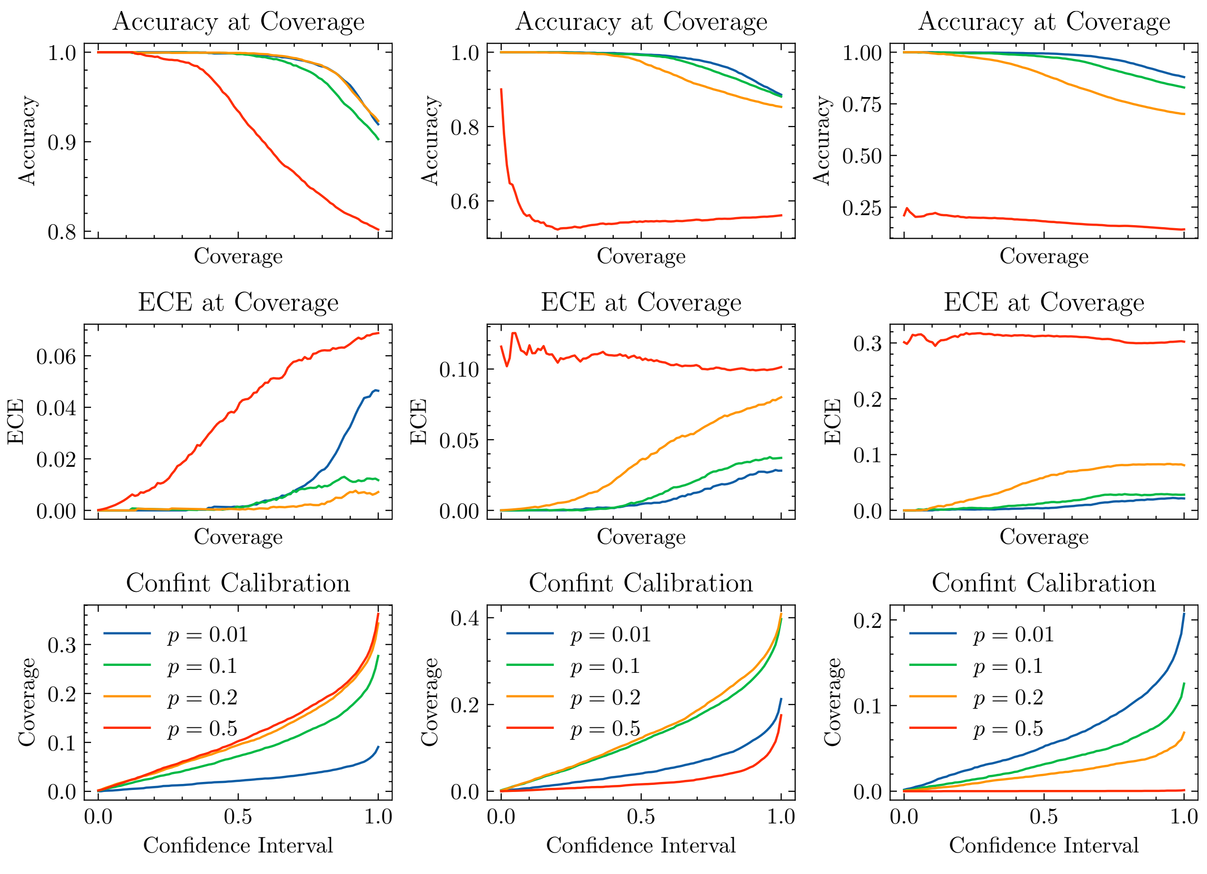
The CIFAR10-H [Peterson et al., 2019] labels for the CIFAR10 [Krizhevsky and Hinton, 2009] test dataset are human predictive probabilities generated by crowd-sourcing 500, 000 human classifications for all 10, 000 test images. These probabilities approximate the human posterior predictive distribution: the distribution from which the original CIFAR10 labels were sampled.

To verify that models which produce meaningful uncertainty estimates have lower ECE for examples with low uncertainty, I train a twelve ANNs on the CIFAR10 dataset and generate uncertainty estimates from them. I do this by using dropout as a Bayesian approximation with three different architectures, four amounts of dropout and either conventional dropout or spatial dropout [Tompson et al., 2015]. With these 12 models, I inspect the ECE against coverage curves to determine whether they have positive gradient and whether EUICE is related to the metrics I proposed.

Model	Dropout	Accuracy ↑	ECE ↓	EUICE ↓	AUAC ↑	AUCC ↓
ResNet18	0.01	91.6	0.0497	0.243	**98.9**	0.0093
	0.1	90.3	0.0132	0.213	98.3	0.0046
	0.2	92.3	**0.0060**	**0.175**	**98.9**	**0.0019**
	0.5	80.3	0.0699	0.325	92.1	0.0369
VGG16	0.01	88.5	0.0291	0.270	97.8	0.0093
	0.1	88.2	0.0388	0.224	97.3	0.0133
	0.2	85.3	0.0800	0.264	95.1	0.0361
	0.5	56.0	0.1010	0.480	55.6	0.1050
MobileNetV2	0.01	87.9	0.0214	0.443	97.5	0.0080
	0.1	82.9	0.0282	0.468	95.2	0.0145
	0.2	70.1	0.0809	0.484	87.4	0.0497
	0.5	14.2	0.3023	0.505	18.1	0.3085

Table 3.2: Table containing the performance of different models on various metrics. Metrics correlated with EUICE are likely to be good proxy metrics. AUAC is the area under the accuracy curve.

I find that EUICE is uncorrelated to the area-under the ECE curve – or the ECE at any particular point. This result is similar to Galil et al., who observed that the relation between ECE and other metrics (in this case EUICE) is architecture-dependent and does not generalise well. I do not have enough models or architectures to draw such a conclusion, but my experiments support it: there seems to be a positive correlation between EUICE and AUCC for ResNet18; with EUICE and AUCC having weaker relationships for VGG16 and MobileNetV2 [He et al., 2016; Simonyan and Zisserman, 2015; Sandler et al., 2018].



(a) ResNet18 **(b)** VGG16 **(c)** MobileNetV2

Figure 3.8: Graphs showing the selective accuracy and expected calibration error at different coverages for three ANN architectures generating uncertainty using DBA. The third graph shows the confidence interval calibration, which we want to be linear. All performant models had positive ECE against coverage curves. This validates my hypothesis that if a model produces internally meaningful uncertainty then its ECE is low for examples with low uncertainty.

Observe in Figure 3.8 that performant uncertainty-generating models had positive gradients at all points on their ECE against coverage curves. I can use this observation to easily recognise many models which do not produce meaningful uncertainties. Of particular interest is the note that the ECE of all models which produced ‘good’ uncertainty estimates is almost 0 below 0.5 coverage. Thus, I take the ratio ECE@0.5/ECE@1 as an approximate measure of how internally meaningful the uncertainty estimates are.

3.6 Uncertainty estimates from SNNs have properties of meaningful uncertainty

In order to establish that BBB, DBA and AoT-SNN generate meaningful uncertainty estimates, I must investigate their behaviour on a real dataset. Scaling up to a large dataset requires using larger models. To meet these requirements, I train spiking ResNets on the NMNIST dataset.

3.6.1 Introducing the NMNIST dataset

I require a dataset small enough for training many models to be computationally feasible, yet large enough to produce meaningful results. The NMNIST [Orchard et al., 2015] dataset is the smallest neuromorphic image dataset, and meets these requirements well. NMNIST is of size $50 \times 60000 \times 2 \times 34 \times 34$: making it 150 times larger than MNIST ($6000 \times 1 \times 28 \times 28$) and 37 times larger than CIFAR10 ($60000 \times 3 \times 32 \times 32$).

Dataset 4: NMNIST

The NMNIST dataset is neuromorphic data generated by taking a neuromorphic sensor and passing it images of the original MNIST dataset. The NMNIST dataset is a set of positions and timestamps of the form (c, x, y, t), where $(c, x, y, t) \in \mathsf{NMNIST}$, indicates that the neuromorphic sensor observed an event at time t, of type c at position (x, y).

To use the data in a software simulation, it is binned into T timesteps leading to a dataset of size $[T, 60000, 2, 34, 34]$. I visualise this in Figure 3.9.

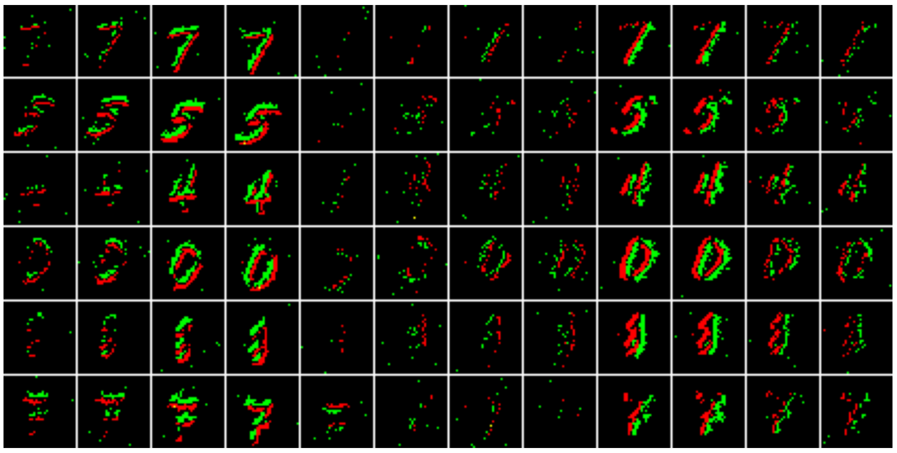


Figure 3.9: Figure visualising random examples from the NMNIST dataset at random timesteps. Different rows contain different examples while different columns contain different timesteps. Neuromorphic sensors can sense two types of events (ON and OFF) which are represented through channels. These channels are visualised as colours, but are not colours.

Like many sequence-to-sequence models, SNNs backpropagate through time. This means that gradients depend on previous timesteps and require an amount of memory linear in the number of timesteps. During the project, I was limited by GPU RAM. I alleviated, by limiting the number of timesteps T to 50.

3.6.2 Constructing uncertainty-generating Spiking ResNets

To work with a dataset 37 times larger than CIFAR10, I must implement the methods discussed on large networks. In Section 3.5 we found that the relationship between ECE and EUICE was model-dependent and seemed strong for ResNets. Based on this, I spike-ify and implement the uncertainty-generation methods on the ResNet18 architecture. Figure 3.10 provides an overview of this process.

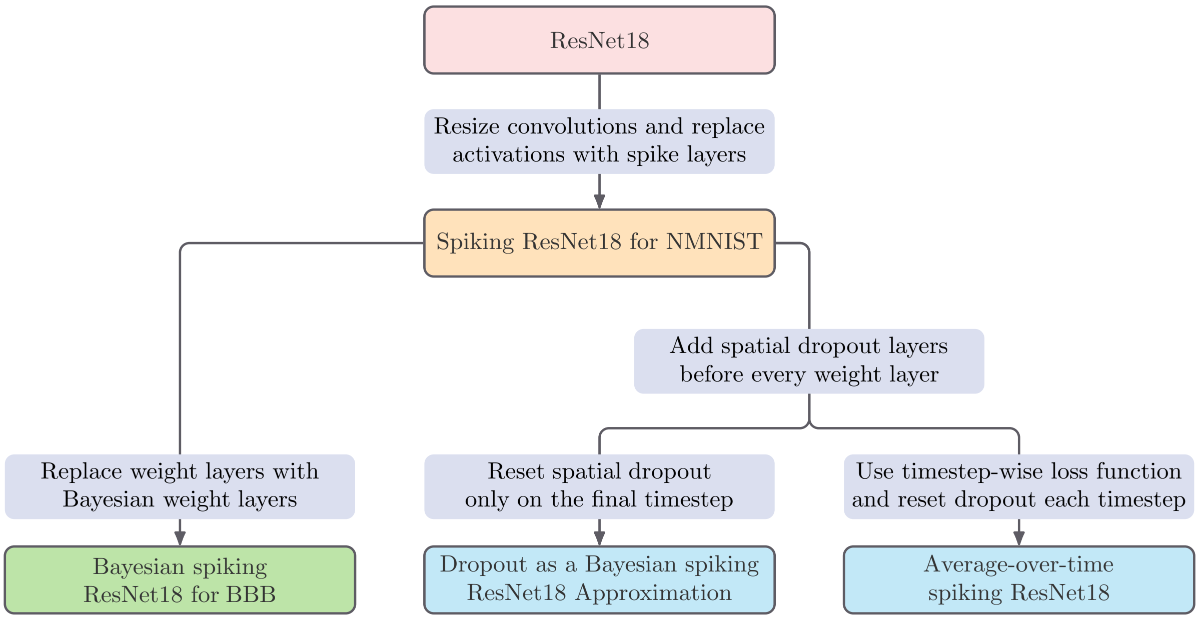

ResNet18
Resize convolutions and replace activations with spike layers
Spiking ResNet18 for NMNIST
Add spatial dropout layers before every weight layer
Replace weight layers with Bayesian weight layers
Reset spatial dropout only on the final timestep
Use timestep-wise loss function and reset dropout each timestep
Bayesian spiking ResNet18 for BBB
Dropout as a Bayesian spiking ResNet18 Approximation
Average-over-time spiking ResNet18

Figure 3.10: Diagram showing the modifications which had to be made to a vanilla ResNet18 to make the uncertainty-generating spiking ResNets used in the latter half of Chapter 3 and throughout Chapter 4.

From ResNet18 to spiking ResNet18 I convert a vanilla ResNet18 into a spiking ResNet18 by replacing all activations with spiking layers and iterating over the time dimension. This network now operates on data of size $T \times B \times 3 \times 32 \times 32$; but NMNIST has dimensionality $T \times B \times 2 \times 34 \times 34$. I therefore adjust the padding and number of input channels in the first convolution such that it operates on data of the correct size.

Bayesian spiking ResNet18 To convert from a spiking ResNet18 into a Bayesian Spiking ResNet18, I sample weights from a trainable variational distribution. The parameters of the network are now the parameters of the variational distribution. I use the method described by Blundell et al. [2015].

Dropout as a Bayesian spiking ResNet18 approximation To convert the spiking ResNet18 to use the method of DBA, I place spatial dropout layers before each weight layer and reset them only after all timesteps have been processed.

Average-over-time spiking ResNet18 I convert the spiking ResNet18 into an AoT-SNN by placing dropout layers before each weight layer, resetting them every timestep and considering the output at each timestep as a different prediction during training.

3.6.3 SNN uncertainty estimates have desirable properties

This section focuses on training the uncertainty-generating SNNs from Section 3.6.2 on the NMNIST dataset and demonstrating that they have the properties identified in Section 3.5.

In Section 3.4.3, I found that each model is sensitive to one hyperparameter: for BBB, the initial standard deviation σ_0; for DBA and AoT-SNN, the dropout probability p. As such, I choose to train models on several values of their sensitive hyperparameters. I use the metrics identified in Section 3.5 to decide which models to use in the evaluation

A baseline SNN converged within 2 epochs. I therefore give my networks 2 epochs to train since practitioners will not generate uncertainty estimates if doing so requires a much larger training compute budget. I found that models were bimodal and either converged quickly or would take many epochs to converge.

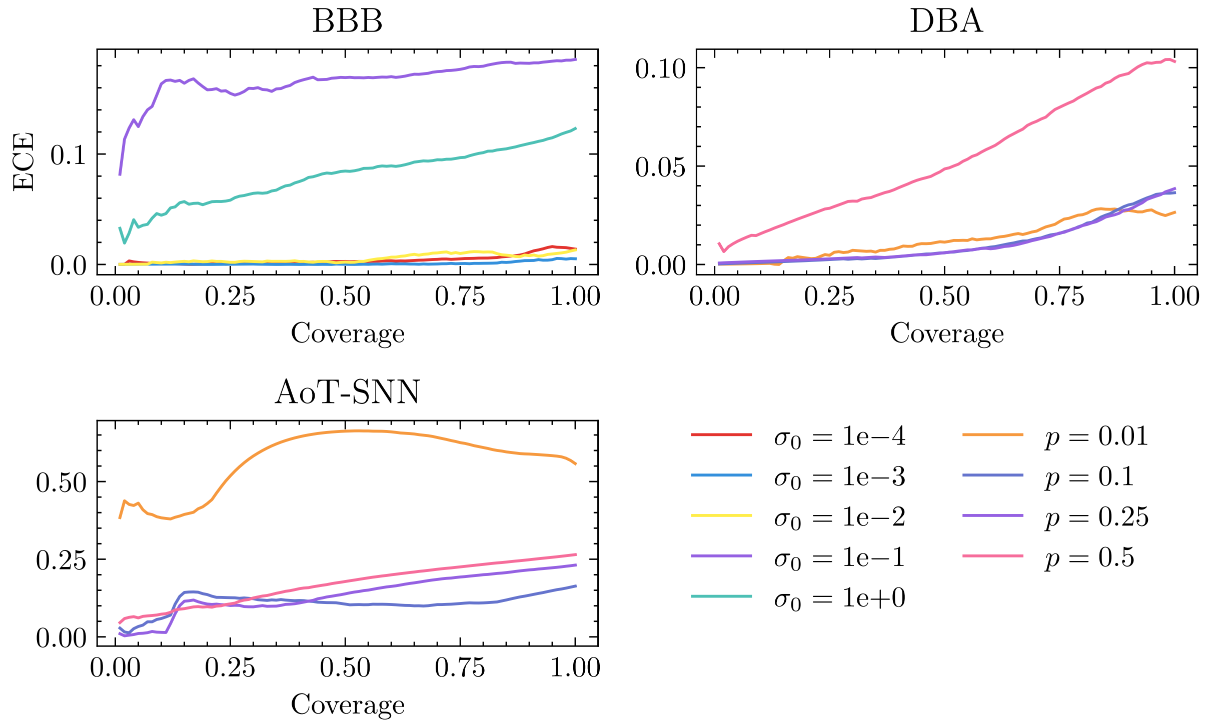


Figure 3.11: The ECE of models trained using each of the regimes with different dropout probabilities p and initial standard deviations σ. BSNNs trained with BBB are by far the best-calibrated models.

Observe in Figure 3.11 and Table 3.3 that for all models which converged (*i.e.* achieved performance comparable to the baseline), the gradient of the calibration curve was positive. This indicates that the ECE is related to the uncertainty. Of particular note are BSNNs trained with BBB which are significantly better calibrated than the other methods. AoT-SNN produced uncertainty estimates and even beat the baseline on accuracy, but had a greatly elevated ECE and has a weaker relation between uncertainty and ECE.

Summary In this section, I achieved the first project goal of generating uncertainty estimates on SNNs. I also achieved one of the extensions by implementing AoT-SNN. I have now laid the groundwork for rigorous evaluation to determine whether the uncertainty estimates produced are meaningful.

Method	Dropout p	Initial σ	Accuracy $\uparrow$	ECE $\downarrow$	AUCC $\downarrow$	ECE50/ECE $\downarrow$	ECE80/ECE $\downarrow$
Baseline	-	-	95.9	0.0405	-	-	-
DBA	0.01	-	86.1	**0.0265**	0.01291	0.429	0.956
	0.1	-	95.7	0.0365	0.01082	0.163	0.546
	0.25	-	**96.2**	0.0385	**0.01071**	**0.155**	**0.513**
	0.5	-	91.6	0.1031	0.05372	0.471	0.832
AoT-SNN	0.01	-	18.3	0.5578	0.57166	1.187	1.093
	0.1	-	77.0	**0.1633**	**0.10977**	0.645	**0.670**
	0.25	-	87.2	0.2310	0.13833	**0.600**	0.866
	0.5	-	**95.3**	0.2647	0.16974	0.673	0.881
BBB	-	1e−4	**95.7**	0.0139	0.00402	0.198	0.415
	-	1e−3	95.3	**0.0052**	**0.00092**	**0.030**	**0.204**
	-	1e−2	86.6	0.0131	0.00534	0.165	0.855
	-	1e−1	10.4	0.1856	0.16668	0.912	0.967
	-	1e+0	10.2	0.1231	0.07986	0.687	0.823

Table 3.3: Tables demonstrating the performance of models on each of the metrics identified for different hyperparameter values. BSNNs trained with BBB are the best-calibrated and have high correlation between miscalibration and uncertainty. DBA achieved high performance and had performed well on the proxy metrics. Models generating uncertainty through AoT-SNN only appeared to have a weak relation between uncertainty and ECE. Furthermore, they had higher calibration error than the baseline.

4 Evaluation

This section answers the second question of the dissertation: are the uncertainty estimates I produce meaningful? I conclusively establish that the uncertainty estimates generated by BBB and DBA are meaningful while those generated by AoT-SNN are not. In Section 2.2.2, I identified three methods of evaluating uncertainty. I investigated the first in Section 3.6, so this section focuses on the remaining two: performance on a downstream task and demonstrating properties that good uncertainty estimates have.

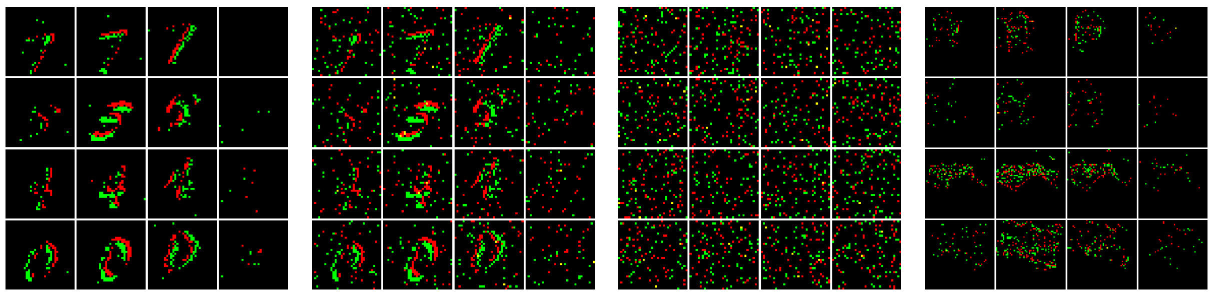
The downstream tasks of interest in this section are out-of-distribution detection, adversarial example detection and k-shot learning. These evaluate the robustness of the trained models. The properties which I investigate are whether the uncertainty on out-of-distribution or adversarial examples is higher than in-distribution; and whether uncertainty decreases as the amount of training data increases.

4.1 Uncertainty increases on out-of-distribution data

I now investigate the behaviour of uncertainty-generating SNNs on out-of-distribution (OOD) data. I explore how their uncertainty changes between in-distribution and OOD data, and investigate whether the uncertainty I produce can be used to differentiate between in-distribution and OOD examples.

4.1.1 Introducing the OOD datasets

There are two broad types of OOD detection: 'near OOD' and 'far OOD'. Near OOD data is drawn from a similar distribution to the original dataset, while far OOD is drawn from a completely different distribution. I consider one near OOD task and two far OOD tasks. The near OOD dataset is NMNIST with added salt-and-pepper noise (random bit-flips) parameterised by probability p. The first far OOD dataset is random noise parameterised by bit-flip probability q. The second far OOD dataset is a different neuromorphic dataset with non-overlapping classes. They are visualised in Figure 4.1.



(a) NMNIST images **(b)** Noisy images, $p = 0.02$ **(c)** Random images, $q = 0.05$ **(d)** NCaltech images

Figure 4.1: Examples from the original dataset compared to the OOD datasets. Visualisation explained in Figure 3.9.

Why Near OOD? Near OOD datasets have similar characteristics to in-distribution data. My near OOD datasets are made my adding noise to NMNIST and contain semantic information. This is the more difficult case of OOD detection since models can sometimes recognise features and may produce meaningful answers.

Dataset 1: Noisy Dataset

Generate noisy data D' according to the following process:

$$D'_{t,c,y,z} = D_{t,c,y,z} \otimes \text{Bernoulli}(p)$$

Where D are data in the original test set, $\otimes$ denotes exclusive or and $p \in [0, 1]$ is the bit flip probability.

Why Far OOD? Far OOD data is dissimilar to the data a model was trained on. These datasets have little or no semantic information, making it impossible for models to produce meaningful predictions on them. My first far OOD dataset is pure random data.

Dataset 2: Random Dataset

Generate random data D'' according to the following process:

$$D''_{t,c,y,x} = \text{Bernoulli}(q)$$

Where $q \in [0, 1]$ is the bit-flip probability.

Why NCaltech101? By adding a dataset with semantically meaningful information and complex patterns which was drawn from a different distribution, we can see how the uncertainty estimates behave in a completely different distribution without any noise. The limitation is that we are unable to observe how the uncertainty behaves in-between NMNIST and NCaltech101 [Orchard et al., 2015].

Dataset 3: NCaltech101

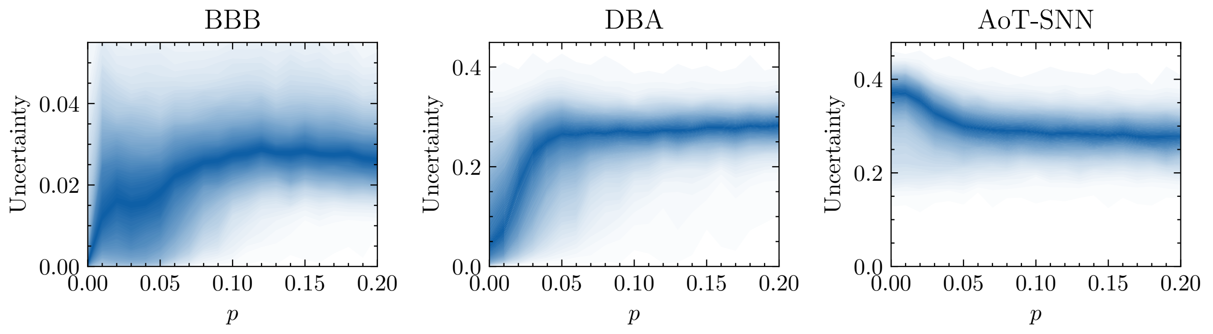
This is a neuromorphic dataset generated by sensing the images in the Caltech101 [Fei-Fei et al., 2007] dataset with a neuromorphic sensor. This dataset contains 101 classes.

4.1.2 BBB and DBA are uncertain on noisy data

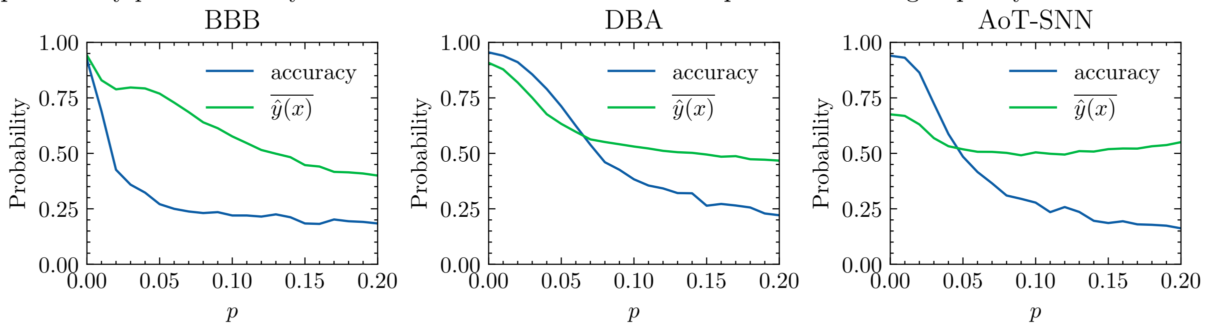
The noisy NMNIST datasets contain semantic information about the original classes. As the bit-flip probability p increases, the dataset becomes corrupted and the amount of information about the original class decreases. I therefore expect the uncertainty to increase as p increases. Good uncertainty estimates would have low uncertainty for low p and increase as p increases until some threshold at which the uncertainty saturates. At this threshold, the model is unable to extract any information from its input.

For each uncertainty-generating model and noisy dataset with bit-flip probability $p \in \{0, 0.1, \ldots, 0.2\}$, I produce uncertainty estimates on 1000 examples. I plot the distribution of their uncertainty in Figure 4.2.

Notice in Figure 4.2 that the uncertainty of Bayesian methods increases as p increases. Importantly, their median uncertainty is roughly proportional to the difference between mean predictive probability, $\overline{\hat{y}(x)}$, and classification accuracy. This indicates that the BBB and DBA models are producing meaningful uncertainty estimates.



(a) Distribution of uncertainty estimates on images in the noisy datasets as a function of the bit flip probability p. The density of uncertainties on each model is represented through opacity.



(b) The average predictive probability, $\overline{\hat{y}(x)}$ and accuracy on images in the noisy family of datasets as a function of the bit flip probability p.

Figure 4.2: Mean predictive probability and uncertainty on the noisy dataset as a function of bit-flip probability p. We want to find a positive correlation between uncertainty and p. I observe this for BBB and DBA; and that their median uncertainty appears to be positively related with the difference between mean predictive probability and accuracy. However, median uncertainty for AoT-SNN appears to be unrelated to both p and the difference between mean predictive probability and accuracy.

Of interest is the numerical difference between the uncertainty of BBB and DBA/AoT-SNN in Figure 4.2. This numerical difference is due to the low initial standard deviation σ_0 for BBB chosen in Section 3.6. This was chosen since low σ_0 led to models which were more performant according to the metrics identified in Section 3.5. Pure Bayesians require uncertainty to be numerically interpretable *i.e.* externally meaningful. However, this project focuses on generating internally meaningful uncertainty estimates and so I do not consider this numerical difference problematic.

Notice how the uncertainty estimated by BBB varies. Initially, the median uncertainty was 6.89e−4 and d_p, the difference between mean predictive probability and accuracy, was 1%. However, at $p = 0.01$, d_p increased to 14% and the median uncertainty instantly increased by a factor of 17 to 1.18e−2. Median uncertainty continued to increase and remained roughly proportional to d_p. This is in-line with the expected behaviour of good uncertainty estimates outlined above and is therefore a strong indication that the uncertainty estimates produced by BSNNs trained through BBB are internally meaningful.

I observe similar results with DBA: the uncertainty increases until a saturation point, which aligns with the desired behaviour. The model accuracy remains fairly high. While median uncertainty seems to be positively correlated with the difference between mean predictive probability and accuracy, this relation is weak. I therefore consider this experiment to only give weak evidence that DBA is producing meaningful uncertainty estimates.

Finally, I observe no obvious relationship between the uncertainty produced by AoT-SNN and the difference between mean predictive probability and accuracy. This indicates that AoT-SNN is not producing meaningful uncertainty estimates.

4.1.3 Can we use uncertainty to detect noisy data?

For uncertainty estimates to be practically useful, we must be able to use them for some downstream task of interest. This section investigates whether a practitioner could use my uncertainty estimates to differentiate between in-domain and noisy (OOD) data.

I investigate two subtasks which practitioners are likely to be interested in: maximising classification accuracy and maximising accuracy of OOD identification subject to a 90% coverage constraint (*i.e.* 90% of in-domain data must be classified as in-domain).

I consider each model and bit-flip probability $p \in \{0, 0.01, \dots, 0.2\}$ separately. For each model and p, I compute the uncertainty on both in-domain data and noisy data. I investigate the first task by recording the accuracy of a classifier which classifies the least uncertain half of datapoints as in-domain and the most uncertain half of datapoints as OOD (Figure 4.3). To address the second task, I record the proportion of OOD data which is correctly labelled as OOD by a classifier which labels examples as OOD if they have uncertainty greater than 90% of in-domain examples (Figure 4.4).

Figures 4.3 and 4.4 shows the accuracies of these classifiers as a function of the bit-flip probability p. I find that both BBB and DBA are performant on these tasks, indicating they may be of practical use and suggesting that they generate meaningful uncertainty estimates. The figures also show that the uncertainty from AoT-SNN cannot be used to differentiate between in-distribution and noisy data.

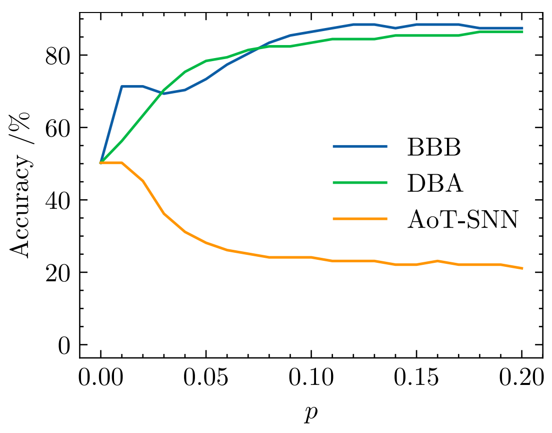


Figure 4.3: Accuracy when using uncertainty to differentiate between in-domain and noisy data. This assumes uncertainty is higher on noisy data. I observe that BBB and DBA are performant while AoT-SNN is not.

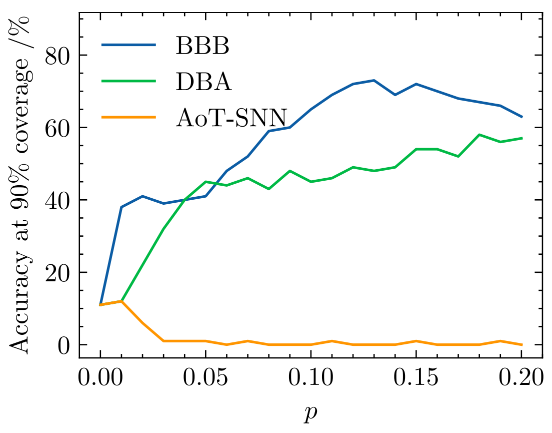
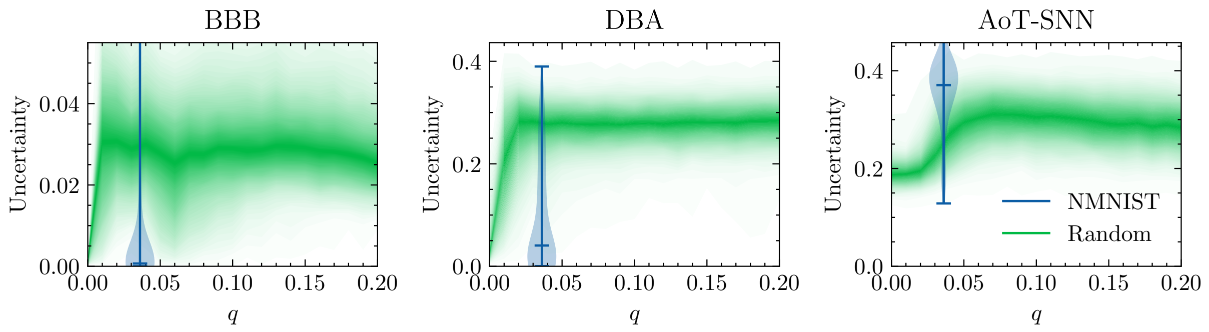


Figure 4.4: Accuracy on the OOD identification task using uncertainty to differentiate, and subject to classifying 90% of in-domain data as in-domain. Again, I find that BBB and DBA are performant, while AoT-SNN is not.

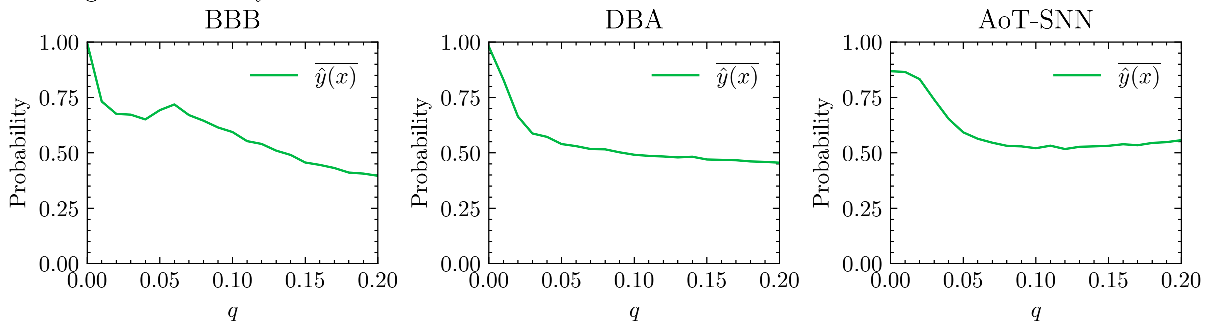
4.1.4 All models are uncertain on random data

In Section 4.1.2, I demonstrated that uncertainty on BBB and DBA was higher when the semantics of NMNIST was disrupted by noise. In addition to disrupting the semantics, adding noise affected a much simpler property: it increased the number of spikes in the input. This section investigates whether this, and not the disrupted semantics, was the cause of the increased uncertainty. I do this by investigating the uncertainty on a family of purely random datasets. These datasets contain no semantic information and have characteristics unlike the NMNIST dataset. We therefore desire the uncertainty to be constant with respect to q for these datasets, with low predictive probability for all values of q.

To test whether my models have these properties, I generate random datasets with bit flip probabilities $q \in \{0, 0.1, \dots, 0.2\}$ and calculate the distribution of uncertainties for each model. I visualise these distributions in Figure 4.5. I also compare the uncertainty on these random datasets to that of NMNIST, in which 3.6% of input features spike.



(a) Distribution of uncertainty for images in the family of random datasets as a function of their bit-flip probability q. The distribution of uncertainties for the NMNIST dataset is shown through a violin plot. The uncertainty of BBB and DBA on the random dataset is higher than on NMNIST. For AoT-SNN, the uncertainty on NMNIST is higher than for random data. This suggests that BBB and DBA produce meaningful uncertainty while AoT-SNN does not.



(b) The mean predictive probability $\overline{\hat{y}(x)}$ on the datasets in the family of random datasets.

Figure 4.5: The behaviour of uncertainty and predictive probability on the family of random datasets. The visualisation is explained in Figure 4.2.

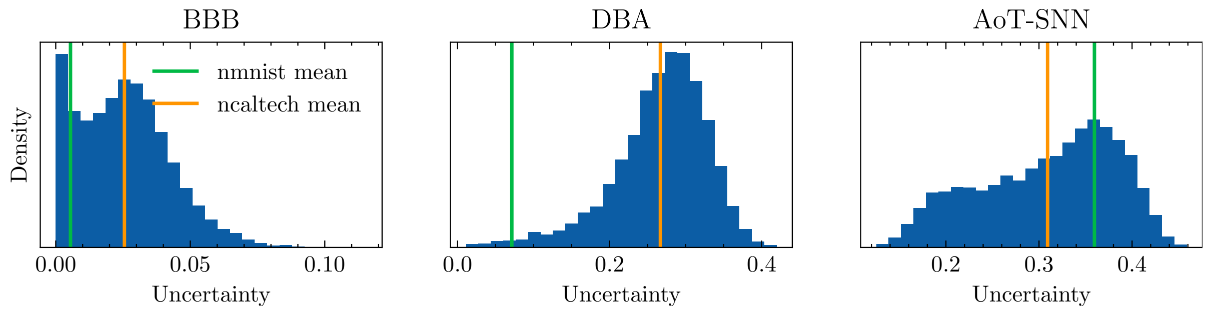
Notice by comparing Figure 4.5 and Figure 4.2, that noisy data with low bit-flip probability has a much lower uncertainty than random data with the same proportion of spikes. This suggests that the cause of the increasing uncertainty in the noisy dataset is not the bit-flip probability, but the increasing disruption to the semantic information.

4.1.5 BBB and DBA are uncertain on NCaltech

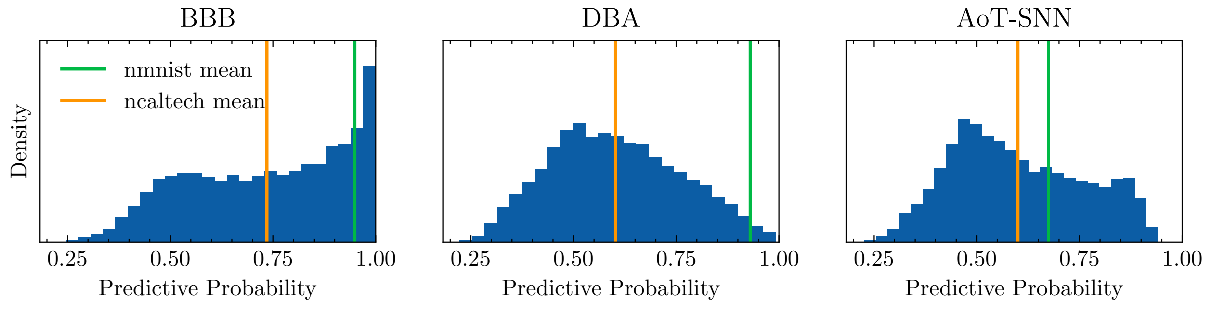
In Sections 4.1.2 and 4.1.4, I investigated uncertainty on synthetic datasets to establish how disrupting information affects uncertainty. I now ask: how do the uncertainty estimates behave if information is present in the data, but the model cannot understand it? To answer this, I evaluate the uncertainty on a different neuromorphic dataset: NCaltech.

NCaltech shares no classes with NMNIST and has different low-level features (Figure 4.1). This means that models trained on NMNIST should be unable to extract or represent information about images in the NCaltech dataset. I therefore expect the uncertainty for models trained on NMNIST to be very high with a very low predictive probability.

I compute the distribution of uncertainties and predictive probabilities of each model on NCaltech and show them in Figure 4.6. The predictive probabilities on the NCaltech dataset are all far higher than the ideal equiprobable. However, I also notice that they are far lower than the predictive probabilities for NMNIST. For BBB and DBA, the uncertainties for NCaltech are far higher than the uncertainties for NMNIST and have roughly the same distribution as for a pure random dataset with high bit-flip probability. The uncertainty for NCaltech on AoT-SNN has decreased: contrary to the desirable behaviour.



(a) Distribution of uncertainties generated by models on the NCaltech dataset. Observe that the uncertainty of BBB and DBA greatly increases, while the uncertainty of AoT-SNN remains roughly constant.



(b) Distribution of probabilities predicted by the uncertainty-generating models on the NCaltech dataset. All models produced many high predictive probabilities, even the highly-calibrated BBB model.

Figure 4.6: The predictive probability and uncertainty of models on the NMNIST dataset (in-domain). Green lines are the mean of the model on NMNIST, while orange lines are the means of the models on NCaltech (out-of-domain). The NCaltech dataset is structured data but distinct from NMNIST so should be semantically meaningless to models trained on NMNIST.

4.2 Are the models uncertain on adversarial examples?

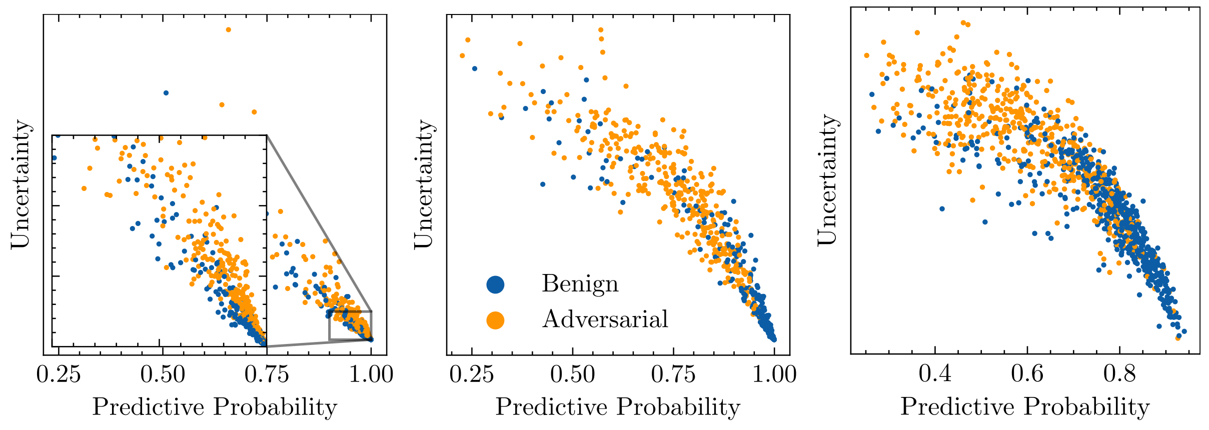
A desirable property of meaningful uncertainty is that the uncertainty on adversarial examples should be higher than on benign examples. If I demonstrate that my uncertainty estimates have this property, it will provide strong evidence that they are meaningful and may be practically useful. I am therefore interested in comparing the uncertainties of adversarial examples and benign examples.

I am particularly interested in comparing adversarial examples to benign examples with the same predictive probability. If the uncertainty on adversarial examples is higher than on benign examples with the same predictive probability, then I can claim that the cause of the increased uncertainty is their adversarial nature and not failed attacks.

4.2.1 BBB is uncertain on adversarial examples

I wish to compare the uncertainty for benign and adversarial examples with the same predictive probability. To do this, I use an adversarial attack for neuromorphic data (described in Section 4.2.2) to generate adversarial examples. I then compute the predictive probability and uncertainty on adversarial and benign examples.

A scatter plot of uncertainty against predictive probability would ideally show two lines. The first line would be the uncertainty of benign examples. The second line would show the uncertainty of adversarial examples and would be vertically above the first. Whilst overlap is acceptable, these lines should be visually distinguishable. The predictive probabilities and uncertainties of adversarial and benign examples are shown in Figure 4.7.



(a) BBB **(b)** DBA **(c)** AoT-SNN

Figure 4.7: A comparison of the predictive probability to the uncertainty for adversarial examples generated using Algorithm 1. Notice that the uncertainty for high predictive probability adversarial examples for the BBB model appears to be higher than the uncertainty for benign examples. The adversarial example generation algorithm was insufficient and unable to generate adversarial examples for AoT in this low-dimensional example.

By inspection of Figure 4.7a, BBB seems to have the desired behaviour: adversarial examples for BBB appear to have higher uncertainty than benign examples having the same predictive probability. However, neither DBA nor AoT-SNN appear to have noticeable differences in uncertainty between adversarial examples and benign examples. The predictive probabilities of adversarial examples on DBA and AoT-SNN are comparatively low for both models. This appears to be partially due to a failure of the adversarial example generation algorithm, so I am unable to make meaningful inference for DBA or AoT-SNN.

I investigate whether the uncertainty of adversarial examples on BBB is higher than that of benign examples by performing a hypothesis test. I observe that for predictive probabilities above 0.9, there is a roughly linear relationship between uncertainty and predictive probability and propose that uncertainty is distributed according to a linear model. With $U_i^{(d)}$ as the uncertainty of example i belonging to dataset $d \in \{\text{adv}, \text{benign}\}$, I propose the model:

$$U_i^{(d)} \sim \mathcal{N}\left(m^{(d)} \cdot (1 - P_i), (\sigma^{(d)} \cdot (1 - P_i))^2\right)$$

where P_i is the predictive probability of example i. I use this to test whether the gradient $m^{(\text{adv})}$ is the same as $m^{(\text{benign})}$. I non-parametrically resample a million datasets and find that none are as extreme as what I observed with BBB. By modelling the distribution of these datasets as normal, I find that the uncertainty for adversarial examples is statistically different to benign examples at $p = 10^{-16}$.

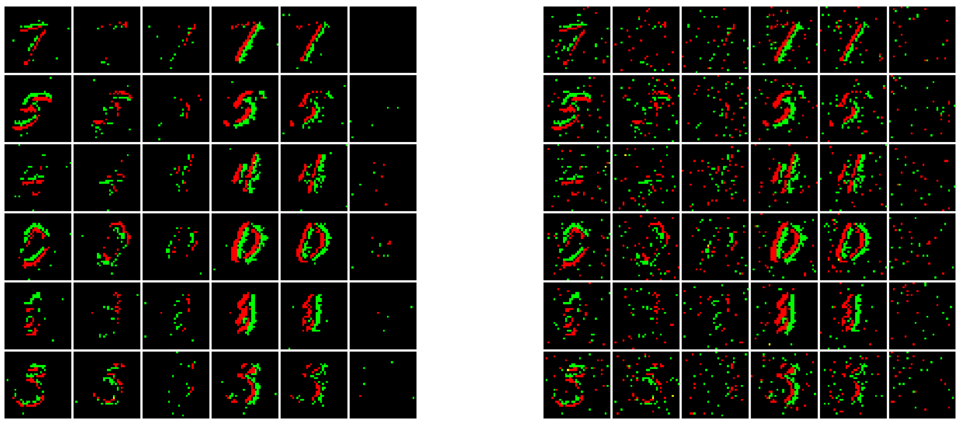
4.2.2 Details of the adversarial dataset

To generate the adversarial examples analysed in Section 4.2.1, I replicated a paper which implemented an attack on neuromorphic data. Neuromorphic data is a type of discrete time series data. Since gradients are only locally valid, discrete data is often much harder to attack and requires custom attacks. Liang et al. propose attacking neuromorphic data by computing the gradients of the inputs with respect to the outputs and 'normalising' them to valid probabilities $p \in [0, 1]$. Bits in the input are then randomly flipped based on this probability p. Simplified pseudocode for the attack is given in Algorithm 1.

Algorithm 1: Simplified version of the adversarial attack by Liang et al.. They use an unspecified normalisation method: I find that normalising such that the sum of squared probabilities within timesteps and channels is 1 generates good adversarial examples.

```
for $i = 0, \dots, N$ do
    $\delta_s \leftarrow \nabla_x(f_\theta(x))$
    $p \leftarrow \text{normalise}(\delta_s)$                    /* Unspecified normalisation method */
    $\delta_{mask} \leftarrow \text{Bernoulli}(p)$
    $x \leftarrow \text{clamp}(x + \text{sign}(\delta_s \cdot \delta_{mask}), 0, 1)$
end
```

Generalising for nondeterministic models The attack by Liang et al. attacks a single neural network. However, I must target a probabilistic ensemble and so must generalise their algorithm. Generalising the algorithm requires setting $\delta_s \leftarrow \mathbb{E}_\theta\left(\nabla_x(f_\theta(x))\right)$. However, approximating this would require many Monte Carlo samples: making it computationally intractable. Instead, I propose using the approximation $\mathbb{E}_\theta\left(\nabla_x(f_\theta(x))\right) \approx \nabla_x(f_{\mathbb{E}(\theta)}(x))$. Using this approximation, I can explicitly construct a model whose weights are $\mathbb{E}(\theta)$ and calculate its gradients efficiently. I visualise adversarial examples generated through the attack in Figure 4.8.



(a) Benign images (b) Adversarial images

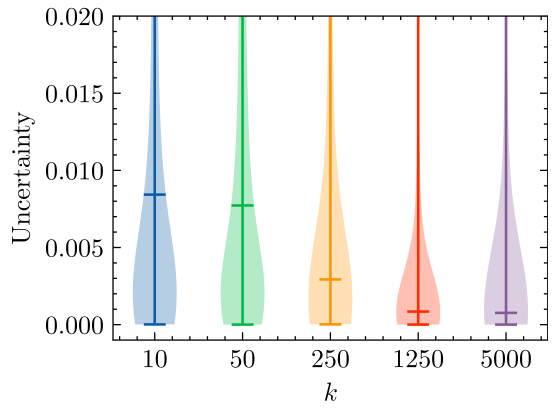
Figure 4.8: Comparison of benign examples from the NMNIST dataset and adversarial examples generated through Algorithm 1. Visualisation explained in Figure 3.9.

An attack for uncertainty This attack aims to generate examples models incorrectly output a high predictive probability on. It does not aim to generate examples with low uncertainty. It may be possible to construct attacks which generate examples with high predictive probability and low uncertainty.

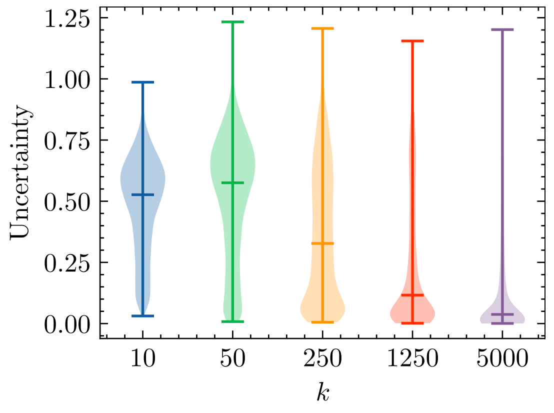
4.3 Models are uncertain in few-shot settings

We want model uncertainty to decrease as the number of examples the model has seen increases. In this section, I use few-shot learning to demonstrate that my uncertainty generation methods have this property. Specifically, I implement k-shot learning on the NMNIST dataset with $k \in \{10, 50, 250, 1250, 5000\}$ using the uncertainty-generating spiking ResNet18 architectures from Section 3.6.2. I use models trained in k-shot settings to generate uncertainty estimates on the NMNIST test dataset. If the uncertainty estimates I generate are meaningful, then models trained on more data should have lower uncertainty.

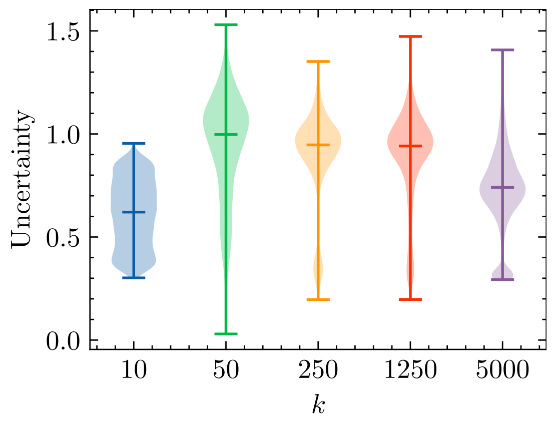
For each model trained on a dataset with k examples, I visualise its distribution through a violin plot in Figure 4.9. Notice that the uncertainty of BBB and DBA decreases as the dataset size increases, while the same does not hold for AoT-SNN.



(a) BBB



(b) DBA



(c) AoT-SNN

Figure 4.9: Violin plots showing the distribution of uncertainty estimates for each model. Notice that the median uncertainty generated by BBB and DBA decreases as the number of datapoints in the training dataset increases. I observe no relation between training dataset size k and AoT-SNN uncertainty.

Summary In this section, I achieved the second project goal of evaluating the quality of uncertainty estimates. This required novel evaluation methodology. I empirically found that Bayesian uncertainty estimation methods produce the most useful uncertainty.

5 Conclusions

5.1 Achievements

The project met both project goals, completed numerous extensions and went far beyond my initial expectations. As such, I deem the project successful. I implemented three types of uncertainty-generating network; two of which have never been done before. This project produced novel results for both the fields of uncertainty estimation and for SNNs.

Generating uncertainty The project began by inventing and implementing Bayesian SNNs and deriving how to train them. From here, I used dropout to approximate Bayesian inference on SNNs and implemented average-over-time SNNs. All three models were used to generate plausible uncertainty on a 2D dataset. This achieved the first project goal.

Demonstrating meaning The project then moved onto achieving the second project goal of demonstrating that the uncertainty estimates produced are meaningful. I did this by making and training deep convolutional spiking neural networks on the NMNIST dataset and comparing their uncertainty estimates on downstream tasks to the desired behaviour of uncertainty estimates. From this, I was able to conclude that BSNNs and DBA produce meaningful uncertainty while AoT-SNN does not.

Contributions to the field This project produced novel results for two fields: SNNs and uncertainty estimation. This project generates the first uncertainty estimates on SNNs, demonstrating that they are still able to represent uncertainty despite having discrete internal state. This project helps to fill a gap in the uncertainty estimation literature by proposing and implementing empirical methods for evaluating the quality of uncertainty produced by a model.

5.2 Lessons Learnt

Falling behind can be okay The early stages of this project marked the first time I have *ever* fallen significantly behind in anything important. Michaelmas courses and my Michaelmas module took significantly longer than expected, and the project encountered several inscrutable, undocumented bugs early on. I learnt that plans can change and falling behind is okay if there is a realistic plan to catch back up.

Formalism For the first few months of the project, I put a lot of effort into formalising every notion before attempting it. This meant months were spent on experiments which (while aiding intuition) did not make significant contributions to the dissertation. In a future project, I would aim to implement a minimum working example earlier to understand the end-product first.

Adaptivity I started off with very clear ideas of how certain implementations (*i.e.* BSNN) should work; and resisted change. An essential part of research is adapting implementations when necessary.

5.3 Future Work

During this project, I thought of many interesting research directions which fell far outside the scope of this project. I summarise the two most important of these below:

SNN-specific Bayesian models This project focused on augmenting SNNs with existing uncertainty estimation methods known to have Bayesian interpretations. Future work could instead investigate whether SNN-specific stochasticity has a Bayesian interpretation and could be used to generate uncertainty estimates. I believe that firing spikes with probability dependent on the membrane potential, or randomly zeroing the membrane potential may have a Bayesian interpretation and be able to produce meaningful uncertainty.

Efficient non-Bayesian uncertainty estimation One of the core results of this project is that AoT-SNNs do not produce meaningful uncertainty. I conjecture that there exist other non-Bayesian methods to produce meaningful uncertainty estimates both for ANNs and SNNs. Specifically, I hypothesise that Bayesian models using some weight/state sharing between samples could produce very weakly dependent samples in sub-quadratic time. I believe that for some types of weight/state sharing, these samples will entail meaningful uncertainty. Future work could implement and compare different methods of weight/state sharing and evaluate how meaningful the uncertainty produced by different methods is. Efficient uncertainty estimation could have implications for the adoption of uncertainty estimation far beyond the field of SNNs.

Bibliography

Anaconda software distribution, 2020. URL https://docs.anaconda.com/.

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701. URL https://doi.org/10.1145/3292500.3330701.

L. Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research (JMLR)*, 20(1):973–978, January 2019. ISSN 1532-4435. URL http://jmlr.org/papers/v20/18-403.html.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, page 1613–1622, 2015. URL http://proceedings.mlr.press/v37/blundell15.pdf.

J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, Bennamoun, D. S. Jeong, and W. Lu. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE*, 111:1016–1054, 2021. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10242251.

W. Falcon and The PyTorch Lightning team. Pytorch lightning, 2019. URL https://github.com/Lightning-AI/lightning.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 2007.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059. PMLR, 2016. URL https://proceedings.mlr.press/v48/gal16.html.

I. Galil, M. Dabbah, and R. El-Yaniv. What Can we Learn From the Selective Prediction and Uncertainty Estimation Performance of 523 Imagenet Classifiers? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=p66AzKi6Xim.

J. D. Garrett. garrettj403/scienceplots. 2021. doi: 10.5281/zenodo.4106649. URL http://doi.org/10.5281/zenodo.4106649.

T. George. timeshift, 2017. URL https://github.com/teejee2008/timeshift.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

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*, Proceedings of Machine Learning Research. PMLR, 2017. URL https://proceedings.mlr.press/v70/guo17a/guo17a.pdf.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Los Alamitos, CA, USA, June 2016. IEEE Computer Society. doi: 10.1109/CVPR.2016.90. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.90.

M. D. Hoffman and A. Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014. doi: 10.5555/2627435.2638586. URL https://www.jmlr.org/papers/volume15/hoffman14a/hoffman14a.pdf.

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013. doi: 10.5555/2567709.2502622. URL http://jmlr.org/papers/v14/hoffman13a.html.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, Toronto, Ontario, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf.

Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

G. Lenz, K. Chaney, S. B. Shrestha, O. Oubari, S. Picaud, and G. Zarrella. Tonic: event-based datasets and transformations., 2021. URL https://doi.org/10.5281/zenodo.5079802. Documentation available under https://tonic.readthedocs.io.

L. Liang, X. Hu, L. Deng, Y. Wu, G. Li, Y. Ding, P. Li, and Y. Xie. Exploring Adversarial Attack in Spiking Neural Networks With Spike-Compatible Gradient. *IEEE Transactions on Neural Networks and Learning Systems*, 34(5):2569–2583, 2023. doi: 10.1109/TNNLS.2021.3106961. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9527394.

W. McGugan and D. Burns. Rich, 2019. URL https://github.com/Textualize/rich/.

A. Nguyen, J. Yosinski, and J. Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, December 2014. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7298640.

Nicki Skafte Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. TorchMetrics - Measuring Reproducibility in Pytorch, 2022. URL https://github.com/Lightning-AI/torchmetrics.

G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Frontiers in Neuroscience*, 9, 2015. ISSN 1662-453X. doi: 10.3389/fnins.2015.00437. URL https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2015.00437.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

J. Peterson, R. Battleday, T. Griffiths, and O. Russakovsky. Human Uncertainty Makes Classification More Robust. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9616–9625, 2019. doi: 10.1109/ICCV.2019.00971. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9010969.

S. Ramírez. Typer, 2019. URL https://github.com/tiangolo/typer.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, Los Alamitos, CA, USA, June 2018. IEEE Computer Society. doi: 10.1109/CVPR.2018.00474. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00474.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

T. Sun, B. Yin, and S. Bohté. Efficient Uncertainty Estimation in Spiking Neural Networks via MC-dropout. In *Artificial Neural Networks and Machine Learning – ICANN 2023*, pages 393–406, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-44207-0. URL http://dx.doi.org/10.1007/978-3-031-44207-0_33.

J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient Object Localization using Convolutional Networks. In *CVPR*, pages 648–656. IEEE Computer Society, 2015. ISBN 978-1-4673-6964-0. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2015.html#TompsonGJLB15.

G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. 2009. ISBN 1441412697.

B. Wendling, K. Gottfried, and E. Bendersky. yapf, 2015. URL https://github.com/google/yapf.

F. Zenke and T. P. Vogels. The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks. *Neural Computation*, pages 899–925, March 2021. URL https://doi.org/10.1162/neco_a_01367.

A Project Proposal

A.1 Introduction

Uncertainty Estimation has been a significant area in machine learning research – and arguably the disciplines greatest failing. Substantial progress has been made – there are many methods which work well, but all have some limitations: training many models [Lakshminarayanan et al., 2017]; training an entirely different class of models [Neal, 1995]; or only working on networks which use dropout [Gal and Ghahramani, 2016]. As a result, uncertainty estimation in neural networks has not been widely adopted and is still very much an open problem with new methods emerging regularly. This leads to situations where models are not adopted in places which would sorely benefit from them; or are overly trusted in situations where they should not be. The lack of uncertainty in models has even spawned an entire sub-discipline: adversarial attacks [Szegedy et al., 2014; Goodfellow et al., 2015] – which itself led to many further interesting methods [Goodfellow et al., 2014].

Spiking Neural Networks(SNNs) [Maass, 1997] are a relatively new take on the mechanics underlying neural networks – their main appeal is their computational efficiency on specialist hardware [Roy et al., 2019]. Rather than a single pass through a set of matrix multiplications, SNNs have binary activations dramatically reducing the number of multiplications required. Data is passed through models multiple times – with layers storing state from previous passes. This is very efficient on specialist hardware. Furthermore, SNNs are approaching performance comparable to similar Artificial Neural Networks (ANNs) on some benchmarks [Li et al., 2021]. It is therefore surprising that there has never been a systematic investigation and evaluation of uncertainty estimation for SNNs. This project aims to rectify that.

A.2 Core

This project aims to implement uncertainty estimation for SNNs and evaluate to what extent existing methods known work on ANNs work in the context of SNNs. Namely, the project shall:

- Use Monte Carlo Dropout-based Uncertainty Estimation (MCDU) [Gal and Ghahramani, 2016] to get uncertainty estimates for SNNs
- Transfer the ideas behind Bayesian Neural Networks (BNNs) [Neal, 1995] from ANNs into the context of SNNs in order to create Bayesian Spiking Neural Networks (BSNNs)
- Evaluate how well these uncertainty estimation methods work in this new context

A.2.1 MCDU

Dropout [Srivastava et al., 2014] zeroes each activation of a layer with probability p during training, then uses all activations (rescaled by p) at inference time. This has been found to cause neural networks to train better and can be interpreted as a form of model averaging.

Gal and Ghahramani noticed that if dropout was considered as model averaging, then an inference with a subset of activations zeroed could be considered as the result of inference from a different model. They then showed that by using dropout at evaluation time, this led to a distribution of outputs which could be viewed as an uncertainty estimate. Thus, they obtained uncertainty estimates from networks without any additional compute overhead at training time and with comparatively low additional inference cost. This method was applied to the context of SNNs by Sun et al., who were able to get uncertainty estimates on SNNs with minimal additional overhead at training time. The project will start off by implementing MCDU for SNNs.

Success Criteria 1 (MCDU). The MCDU method has been employed on SNNs and used to get uncertainty estimates.

A.2.2 BSNN

Many of the earlier attempts to get uncertainty estimates from Neural Networks involved the use of BNNs [Neal, 1995]. In these networks, the parameters are modelled as a distribution. Over the decades of research into uncertainty estimation via BNNs, many different types of BNN have arisen. As part of the core of the project, I will investigate BNNs and aim to use their ideas to create a BSNNs and then use it to get uncertainty estimates. Due to the wide range of types of BNNs, there is a large range of possible routes (and corresponding difficulties) which this stage of the project could take.

Success Criteria 2 (BSNNs). The ideas behind BNNs have been transferred into the context of SNNs to create a BSNN and have been used to get uncertainty estimates.

A.2.3 Evaluate Quality

The purpose of uncertainty estimates is to accurately reflect the uncertainty of the model which we are evaluating – which itself should accurately reflect the uncertainty of the data. There has been significant research into this area and there exist many common metrics [Chung et al., 2021]. Most of these can be fairly easily transferred from the SNN literature onto SNNs. Many methods of evaluating uncertainty estimates are controversial [Sluijterman et al., 2023] meaning that wider reading is required to establish the best metrics to use for my specific case. In short: evaluating the quality of uncertainty estimates is a *well-researched and well-understood open problem* with lots of available resources and tools. This means that there are a wide range of evaluations which could be carried out; and opens the opportunity for meta-evaluation. *I consider this to be an advantage*; giving greater opportunity and justification for an interesting in-depth evaluation.

Part of my research phase will involve deciding on suitable evaluation metrics (such as negative log-likelihood, continuous ranked probability score, check score, interval score *etc.*) and datasets. Any datasets we evaluate on should both be small but be complex enough to have large and easily measurable uncertainty – there are many datasets which fit these criterion [Cohen et al., 2017; Krizhevsky and Hinton, 2009; mnmoustafa, 2017]. Investigation is required to determine whether there are common datasets designed specifically for uncertainty estimation. We can compare against the uncertainty estimation implementation from Sun et al., and compare against how the methods work for ANNs with similar performance.

At minimum,there should be evaluation on synthetic datasets where the true uncertainty is known, such as linear regression plus gaussian noise. I could either make such datasets myself or use open source datasets [Kabir et al., 2023]. Using models for which the “true uncertainty” is not known introduces complexities. These may be addressed by using conventional uncertainty estimation techniques from ANN literature – such as deep ensembles [Lakshminarayanan et al., 2017]. Note that the success criteria is met if the implementations have been transferred onto SNNs and evaluated; regardless of *how well* the methods works in this new context.

Success Criteria 3 (Quality). Suitable evaluation metrics have been found in the ANN literature and used to evaluate how good the uncertainty estimates from our implementations of MCDU and BSNNs are.

A.3 Extensions

This project lends itself to a wide range of extensions. I anticipate that after having completed the core of my project, I will have a better idea of which extensions will be the most worthy. Here is a set of extensions I could do if the core tasks are met:

A.3.1 Get uncertainty estimates on a wider range of SNNs and BNNs

There is a wide range of variants of both SNNs *and* BNNs. This leads to a vast set of variants of BSNNs: the set of variants of BSNNs is the cartesian product of the variants of SNNs and BNNs. It could be interesting to consider how different variants can affect the quality of uncertainty estimates and the general characteristics of these BSNNs.

A.3.2 Evaluate on larger datasets and models

Since the core of the project aims to *implement* uncertainty estimation, it has no real requirement for usage of large datasets or models. However, a rigorous evaluation would include evaluation on larger [Deng et al., 2009] or real-world tasks with models of practical size [Tan and Le, 2021; Howard et al., 2019; Liu et al., 2021]. As such, a possible extension would be to scale up the experiment size to real datasets. This introduces a number of complexities – and adds the potential for our methods to be compute bound.

A.3.3 Create a testbench designed for SNN uncertainty estimation

It is well-known that SNNs are better suited for neuromorphic datasets [Iyer et al., 2021] – data where the data is encoded in “spikes” and which allows them to use their innate temporal aspect. It has been proposed that SNNs should be evaluated on separate benchmarks in some cases [Deng et al., 2020] – it would be interesting to consider whether this is the case for uncertainty estimation – and if so which datasets should be used. A possible extension to this project would be to create a testbench for uncertainty estimation which specifically uses datasets designed for SNNs.

A.3.4 Distribute a library for SNN uncertainty estimation

A core part of the motivation for this paper is that there has been little research in, and are no open source implementations of SNN uncertainty estimation. A sensible extension would therefore be to distribute an open-source library with an emphasis on uncertainty estimation in SNNs. Such a library would be built on top of PyTorch [Paszke et al., 2019] and an open source SNN library [Pehle and Pedersen, 2021; Eshraghian et al., 2021]. This should make future research in this area easier, as well as adding further credibility to our results by making replication easier.

A.4 Starting Point

I have written no code pertaining to this project. I have prior experience with machine learning – but neither with SNNs or BNNs. In preparation for this project, I read some papers on SNNs and Uncertainty Estimation and worked through some tutorials on an open source SNN library. The only prior experience I have with uncertainty estimation comes from the Part IB Data Science Course – I expect this project to go far beyond that. There exist separate open source libraries for SNNs, for BNNs and for uncertainty estimation. However, to the best of my knowledge there is no open source code pertaining to uncertainty estimation for SNNs.

A.5 Work Plan

A.5.1 Michaelmas Term

- **16th October 2023 – 27th October 2023**
  - Literature review, including papers on SNNs, papers on uncertainty estimation and papers on BNNs
  - Investigate and decide on suitable tools to use throughout the project *e.g.* kanban boards, testing frameworks, linters, reference managers *etc.*
  - **Deliverable:** Document summarizing how SNNs and BNNs work and a low level summary of how we can extend them to uncertainty estimation
  - **Deliverable:** List of tools identified to be used throughout the project
- **28th October 2023 – 10th November 2023**
  - Review the literature and decide on which metrics are most suitable for evaluating the quality of uncertainty estimates
  - Decide on suitable benchmarks and datasets (and get local copies) for use to uncertainty estimation based on common practice in the uncertainty estimation literature
  - Create or obtain simple synthetic datasets with known uncertainty for quick provisional evaluation of the quality of uncertainty estimation
  - Make a simple testbench on which to evaluate the quality of uncertainty estimates
  - **Deliverable:** Testbench which evaluates the quality of uncertainty estimates on simple datasets
- **11th November 2023 – 24th November 2023**

  - Slack Time, start implementing MCDU if possible
  - **Commitment:** Category Theory Graded Exercise Sheet
  - **Deliverable:** Document outlining the approach to implementing a BSNNs
- **25th November 2023 – 8th December 2023**
  - Implement MCDU
  - **Deliverable:** Working implementation of MCDU with provisional results on synthetic test data

A.5.2 Michaelmas Vacation

- **9th December 2023 – 22nd December 2023**
  - Read papers on BNNs and determine how to implement BSNNs
  - Implement BSNNs
  - **Deliverable:** Document outlining the approach to implementing a BSNNs; working BSNNs with provisional results on synthetic test data
- **23rd December 2023 – 4th January 2024**
  - Slack time, finish off any outstanding work if needed
  - Break for Christmas

A.5.3 Lent Term

- **5th January 2024 – 19th January 2024**
  - Train networks on evaluation datasets
  - Obtain uncertainty estimates from evaluation datasets
  - **Commitment:** Category Theory Take-Home test
  - **Deliverable:** Trained networks on evaluation datasets with uncertainty estimates
- **20th January 2024 – 2nd February 2024**
  - Use evaluation metrics identified to evaluate the quality of the uncertainty estimates
  - **Deliverable:** Tables summarising the quality of the uncertainty estimates obtained on a range of different metrics
  - **Deliverable:** Completed Core
- **3rd February 2024 – 16th February 2024**
  - Slack Time, start working on extensions if possible
  - Review which extensions are most interesting and update the work-plan accordingly

  - **Deliverable:** Summary of extensions to attempt and updated work plan

- **17th February 2024 – 1st March 2024**
  - Work on extensions
  - **Deliverable:** Extension-specific – specified in the updated work plan
- **2nd March 2024 – 15th March 2024**
  - Work on extensions
  - Start writing the introduction and methodology chapters
  - **Deliverable:** Extension-specific – specified in the updated work plan

A.5.4 Lent Vacation

- **16th March 2024 – 29th March 2024**
  - Work on extensions
  - Finish the introduction and methodology chapter
  - **Commitment:** Revision
  - **Deliverable:** Introduction and Methodology completed
- **30th March 2024 – 12th April 2024**
  - Write the evaluation chapter
  - Finish the first draft of the dissertation
  - **Commitment:** Revision
  - **Deliverable:** First draft of dissertation completed

A.5.5 Easter Term

- **13th April 2024 – 26th April 2024**
  - Revision
  - Iterate on Dissertation
  - **Commitment:** Revision
  - **Deliverable:** second draft of Dissertation completed
- **27th April 2024 – 10th May 2024**
  - Final Correction and submit report
  - **Commitment:** Revision
  - **Deliverable:** Dissertation completed and submitted

A.6 Resource Declaration

I will use my personal laptop (Acer Swift 5 – i7-1165G7 2.80GHz, 8GB RAM) as my primary working device, with an older HP laptop as backup. I shall use GitHub and OneDrive to perform regular backups of my repository. *I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.*

I will need access to GPUs for training the models and performing uncertainty estimation. I have access to GPUs in the CL from when I interned there; I anticipate using these and have asked permission to do so. As a backup, I would rent GPUs (*i.e.* Google Colab) or pay for HPC (*i.e.* Wilkes 3).

Bibliography

Y. Chung, I. Char, H. Guo, J. Schneider, and W. Neiswanger. Uncertainty Toolbox: an Open-Source Library for Assessing, Visualizing, and Improving Uncertainty Quantification. *arXiv preprint arXiv:2109.10254*, 2021. URL https://arxiv.org/abs/2109.10254.

G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017. doi: 10.1109/IJCNN.2017.7966217. URL https://ieeexplore.ieee.org/document/7966217.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://ieeexplore.ieee.org/document/5206848.

L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, and Y. Xie. Rethinking the Performance Comparison between SNNS and ANNS. *Neural Networks*, 121(C): 294–307, January 2020. ISSN 0893-6080. doi: 10.1016/j.neunet.2019.09.005. URL https://doi.org/10.1016/j.neunet.2019.09.005.

J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, Bennamoun, D. S. Jeong, and W. Lu. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE*, 111:1016–1054, 2021. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10242251.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059. PMLR, 2016. URL https://proceedings.mlr.press/v48/gal16.html.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

A. Howard, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, Los Alamitos, CA, USA, November 2019. IEEE Computer Society. doi: 10.1109/ICCV.2019.00140. URL https://doi.ieeecomputersociety.org/10.1109/ICCV.2019.00140.

L. R. Iyer, Y. Chua, and H. Li. Is Neuromorphic MNIST Neuromorphic? Analyzing the

Discriminative Power of Neuromorphic Datasets in the Time Domain. *Frontiers in Neuroscience*, 15, March 2021. doi: 10.3389/fnins.2021.608567. URL https://doi.org/10.3389%2Ffnins.2021.608567.

H. M. D. Kabir, M. Abdar, A. Khosravi, D. Nahavandi, S. K. Mondal, S. Khanam, S. Mohamed, D. Srinivasan, S. Nahavandi, and P. N. Suganthan. Synthetic Datasets for Numeric Uncertainty Quantification Proposing Datasets for Future Researchers. *IEEE Systems, Man, and Cybernetics Magazine*, 9(2):39–48, 2023. doi: 10.1109/MSMC.2022.3218423. URL https://ieeexplore.ieee.org/document/10104146.

A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, Toronto, Ontario, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf.

Y. Li, S. Deng, X. Dong, R. Gong, and S. Gu. A Free Lunch From ANN: Towards Efficient, Accurate Spiking Neural Networks Calibration. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6316–6325. PMLR, 18–24 July 2021. URL https://proceedings.mlr.press/v139/li21d.html.

Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021. URL https://openaccess.thecvf.com/content/ICCV2021/papers/Liu_Swin_Transformer_Hierarchical_Vision_Transformer_Using_Shifted_Windows_ICCV_2021_paper.pdf.

W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(97)00011-7. URL https://www.sciencedirect.com/science/article/pii/S0893608097000117.

M. A. mnmoustafa. Tiny imagenet, 2017. URL https://kaggle.com/competitions/tiny-imagenet.

R. M. Neal. Bayesian learning for neural networks. 1995. URL https://api.semanticscholar.org/CorpusID:251471788.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

C. Pehle and J. E. Pedersen. Norse - A deep learning library for spiking neural networks, January 2021. URL https://doi.org/10.5281/zenodo.4422025. Documentation: https://norse.ai/docs/.

D. Roy, I. Chakraborty, and K. Roy. Scaling Deep Spiking Neural Networks with Binary Stochastic Activations. In *2019 IEEE International Conference on Cognitive Computing (ICCC)*, pages 50–58, 2019. doi: 10.1109/ICCC.2019.00020. URL https://ieeexplore.ieee.org/document/8816938.

L. Sluijterman, E. Cator, and T. Heskes. How to Evaluate Uncertainty Estimates in Machine Learning for Regression?, 2023. URL https://arxiv.org/abs/2106.03395.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

T. Sun, B. Yin, and S. Bohté. Efficient Uncertainty Estimation in Spiking Neural Networks via MC-dropout. In *Artificial Neural Networks and Machine Learning – ICANN 2023*, pages 393–406, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-44207-0. URL http://dx.doi.org/10.1007/978-3-031-44207-0_33.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR (Poster)*, 2014. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2014.html#SzegedyZSBEGF13.

M. Tan and Q. Le. EfficientNetV2: Smaller Models and Faster Training. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR, 18–24 July 2021. URL https://proceedings.mlr.press/v139/tan21a.html.