Bayesian Neural Networks

Blair, Taylor Sorgman, Ava Bekaert, Conor

May 6, 2024

Contents

1 Machine Learning

Machine learning includes a variety of topics of which artifical intelligence (AI) and neural networks are a subset. Neural networks in particular involve an abstract imitation of the human brain using simulated neurons which is trained on data using particular algorithms.

The term supervised learning refers to a specific common method of machine learning that uses a labeled dataset to train a model to correctly predict the labels via some training algorithm. A classification problem in supervised learning involves predictions where all labels are grouped into a set of categories. The training process can be broken down into three major steps, which include a decision process, an error function, and an optimization process [**berkelyWhatIsML**]. The decision process is the set of steps the algorithm takes after receiving the data based on the goal of the model. The second step in the process is the error function, which is the method of measuring chosen to see if the algorithm gave a “good" or “bad" input. The two most common choices of the error function are a simple yes or no on whether a data point was classified correctly in the case of classification, or the difference in value between the predicted outcome and the actual observed outcome for continuous values. Finally, the third and last step is the part of the process that implements learning. This step, the updating or optimization process, requires the algorithm to review the past data and outputs of the error function in

order to better correct its decision-making process in the future.

Throughout this report, we will often use the terms machine learning, deep learning, and neural networks. It is important to note that although these are fundamentally related fields, deep learning is a subfield of neural networks that in particular focuses on more complicated neural networks, while neural networks are a subfield in machine learning.

We will be training a classifier on labeled images in a supervised learning process to predict what is depicted in the image from a range of possibilities such as dog, cat, and plane. These images come from the CIFAR-10 dataset, which will be discussed more in detail later in this paper.

1.1 Introduction to Neural Networks

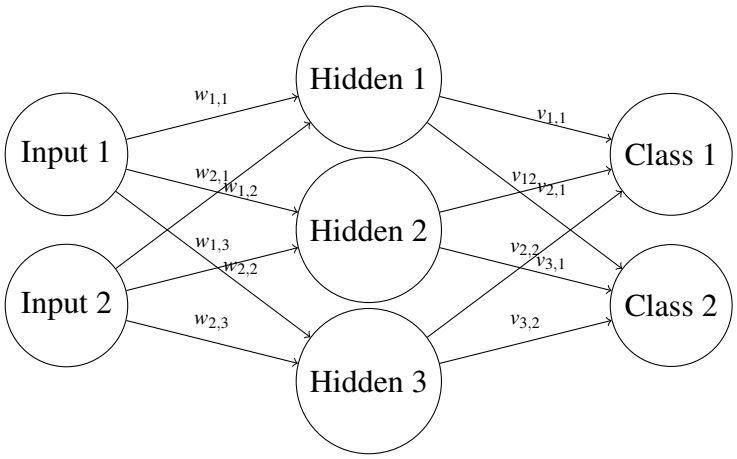


Figure 1: Example neural network

There are many effective introductions to neural networks to any level of detail available elsewhere, but we will present a summary of the core concepts here.

Many models for analyzing data involve making an assumption about the functional form of the outputs given the inputs or predictors. Neural networks are no exception - at the most basic conceptual level, a neural network is just a really complicated function that has the potential to represent many patterns through its complexity. Their name comes from the inspiration they take from the human brain, the idea that enough neurons with enough connections between them can encode everything needed for effective predictions. Indeed, the fundamental unit of a neural network is a neuron and its associated connections.

Neural networks are typically organized in a series of layers of many neurons, where each individual neuron of one layer is connected to every neuron of the previous layer. Data flows from an initial input layer, through hidden layers, then into the outputs as visualized in Figure [**fig:example-nn**]. The connections are assigned certain weights, which measure the influence of a neuron on the next layer. The next section will describe the structure of neurons in a linear layer, which is a typical type of layer.

1.2 Neural Network Neurons

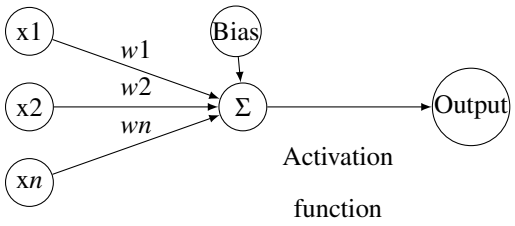


Figure 2: Neural network neuron

A neural network neuron in a linear layer is, in practice, a varying real scalar value. The value is produced from the values $X = (x_1, ..., x_n)$ of the neurons in the previous layer by taking the dot product of X with the weight parameters $W = (w_1, ..., w_n)$, plus an additive bias b. This weighted sum $X \cdot W + b$ is then passed through some activation function $\text{Act}(\cdot)$ which applies a nonlinear transformation to improve efficiency. As such, the action of a particular layer on the previous layer's neurons to produce the next value can be characterized as $X_{i+1} = \text{Act}(X_i \cdot W_i + b_i)$. Typical activation functions include the Rectified Linear Unit (ReLU) defined by $\text{ReLU}(x) = \max(0, x)$ and the sigmoid, $\sigma(x) = \frac{1}{1+e^{-x}}$.

This process can be generalized to an entire layer at once via matrix multiplication (which encodes the dot product), and this also improves efficiency. These layers are nested interatively to the depth of the network - for example, a network with three linear layers could take the form

$$\text{ReLU}(\text{ReLU}(\text{ReLU}(x \times W_1 + b_1) \times W_2 + b_2) \times W_3 + b_3).$$

A network architecture is generally defined by the depth (number of layers), the number of neurons/parameters in the layers, and the type of layer used.

1.3 Neural Network Training

To train a neural network requires a loss function L, such as the MSE or the cross-entropy loss (cross-entropy loss is essentially the log-likelihood loss from frequentist maximum likelihood estimation). An optimization process, typically

a variant of stochastic gradient descent, adjusts the neural network parameters β (the weights W and biases b) to minimize this loss on the provided data. Stochastic gradient descent updates the parameters at each step by the formula

$$\beta_{n+1} = \beta_n - \alpha(\nabla \ell(x_s, y_s, \beta)),$$

where α is the *learning rate*, which controls how quickly parameter adjustments are made. Minimization is implemented using a stochastic approximation of the gradient of the loss (hence the name stochastic gradient descent). The true loss $L(\beta)$ on the whole dataset x, y is approximated via an averaged estimate $\ell(x_s, y_s, \beta)$ on a subset (x_s, y_s) of the data. Gradients are calculated via *backpropagation*, which is beyond the scope of this discussion, but essentially consists of iterated application of the chain rule.

Each step of training is called an *epoch*. In each epoch, the data passes through the network, the gradients are approximated via backpropagation, and the parameters are updated with this approximation. These parameters, matrices of weights and biases, *are* the network once it is trained - they represent the "memory" of the network, whatever it has learned from the training data. Predictions are obtained as the outputs after passing new data through the same network architecture with these weights and biases specified. This is a "black box" model because there is no clear interpretation of the giant matrices of parameters - in the end, they're just floating-point numbers.

1.4 Convolutional Neural Networks

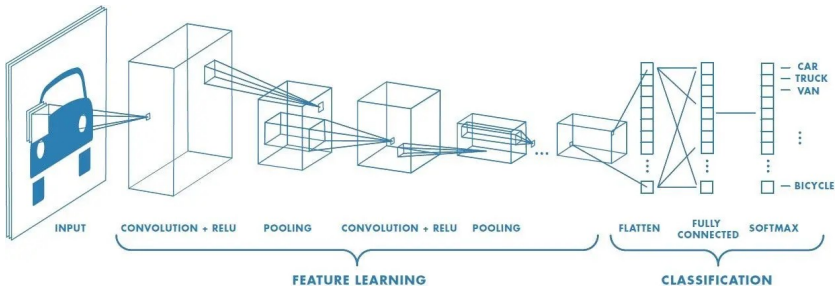


Figure 3: CNN pipeline [**eli5CNN**]

Convolutional neural networks (CNN) are a type of neural network that is better suited for image recognition. While this might sound like a separate model structure, CNNs are largely the same. The primary difference from a traditional neural network is the convolutional layer, as highlighted in Figure [**fig:cnn-pipeline**]. Instead of reading the entire image at once, a convolutional layer slides a small window over the image, shrinking that window even further by the application of a convolution operation. Once the window has slid over the entire image, a downsized image is formed that may encode useful information more densely. Because of this downsizing action, CNNs typically require fewer parameters relative to the traditional neural networks described previously, which improves training costs and the capabilities of the model.

2 Bayesian Neural Networks

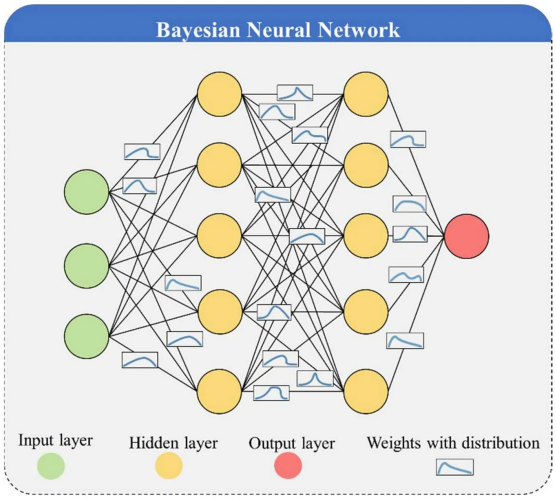


Figure 4: Example BNN [**FleszarBNN**]

Bayesian neural networks (BNN) are similar in principle to typical neural networks, but represent uncertainty in the predictions of a neural network by assigning probability distributions to the parameter (weight and bias) values and updating these distributions. This follows Bayesian principles, where rather than producing a single estimated value as in typical neural networks, the "evidence" provided by the data updates a distribution of possible true values.

Uncertainty can be built into these models through prior probability distributions. The average estimate of the parameters is computed through multiple different models during the training process, which can regularize the network. However, it can be very difficult to calculate the model posterior of a BNN. Approximations of the posterior of the model are most often used, but those are often still

very computationally intensive. Different methods of approximating the model's posterior is a current research topic within the field.

The two main types of uncertainty within a BNN are aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty is a measure of the uncertainty in the observations, which is uncertainty that is inherent in any form of data. This type of uncertainty is modeled by placing a prior distribution over the output of the BNN model. In contrast, epistemic uncertainty is a measure of the uncertainty caused by the model and its predictions. It is modeled by a prior distribution over the model's weights, and an analysis of how much the weights of the model change when the data given to the model changes [**shridhar2019comprehensive**].

But what is so great about calculating uncertainty in our model? One of the major issues with neural networks as they are is they are highly prone to overfitting the data. Although they are great at predictions on the training data they are given, they often struggle with predictions on the testing data they are given due to the issue of overfitting. Many methods have been developed in order to regularize neural networks to work to prevent this from occurring, such as weight control measures. The Bayesian framework, and particularly the choice of prior distributions, builds this into the model, which helps prevent overfitting.

Recall that a (trained) neural network is really just determined by its parameters - the weights and biases. This gives rise to the conceptual interpretation that a Bayesian neural network is an infinite ensemble of possible neural networks which can be sampled to obtain any number of individual networks.

Constructing a Bayesian neural network involves selecting a model architecture

and choosing the priors, which are also known as the "stochastic model" in this context. The definition of Bayesian neural networks provides flexibility, since the primary difference is that neuron weights get distributions instead of individual values. As a result, most model architectures that work for a typical neural network also work for a Bayesian version of that network.

2.1 Bayesian Neural Network Neuron

A visualization of a Bayesian neuron appears in Figure 5. It is in fact quite similar to Figure 2, with the essential difference that the individual values from before have been replaced with distributions. Within a neuron, the distributions from input neurons are weighted by the weight distributions and combined together along with the bias distribution. After passing through the activation function, the neuron output is produced.

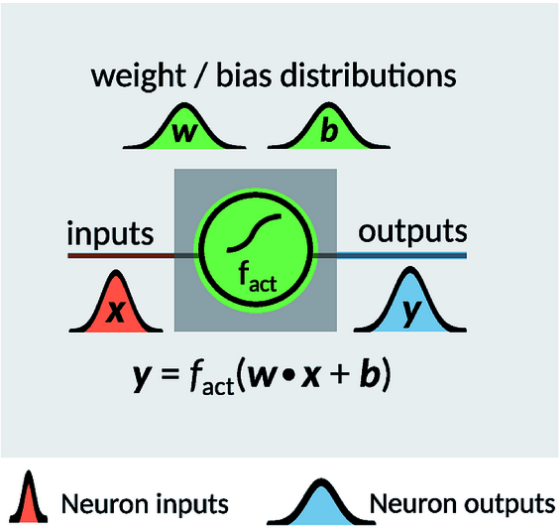


Figure 5: Example BNN Neuron [**hase2019machine**]

2.1.1 Priors

The Bayesian paradigm requires a prior on the parameters of a neural network to facilitate training. However, making choice of a prior on all the parameters of a network, following anything other than a simple concept, is difficult, especially given the limited understanding of how priors at one layer affects the next in general. For the purposes of this work, only one prior-selection scheme will be considered, since considering many methods would require introducing many further new concepts.

A choice of prior that is both commonly used and intuitive is an independent normal distribution on each weight. In particular, a $\mathcal{N}(0, \sigma^2)$ distribution with zero mean and some standard deviation σ for each particular weight typically works. This can be thought of as a reasonable choice of vague prior. Since further interpretation of how these priors influence the model becomes even more complicated for other priors, selecting a more informative prior would be tricky. For an example of this complexity, Vladimirova et al. contend that for a network with these priors whose activation functions satisfying a certain "extended envelope property", the priors on the k^{th} hidden layer become sub-Weibull-distributed with parameter $k/2$ conditional on the data passing through the network. Such interpretation is valuable, but beyond the scope of this paper.

However, it turns out there is suggestive parallel between a prior in a Bayesian neural network and *regularization* in a typical neural network. *Regularization* is an additional term in the loss that penalizes large weights. Intuitively, this helps resist overfitting by preventing a single neuron from overly influencing the model,

since such a neuron may be overemphasizing a fluke of the training set rather than a general pattern. The regularization term typically takes the form $\alpha||w||_p$ where $||\cdot||_p$ is some choice of metric on R^n, typically $p = 1$ (Manhattan/taxicab distance) or $p = 2$ (Euclidean distance), and α is the regularization hyperparameter that controls how strongly large weights are penalized. In other words, α is the *regularization rate*.

As in Jospin et al., consider network parameters θ with a loss function L assumed to be the negative log likelihood loss, where D_y represents the true labels for the training data D_x. Suppose the network does not use regularization. Maximum likelihood estimation aims to find the choice of θ which minimizes the loss on the given data:

$$\hat{\theta} = \arg\min_{\theta} \left\{ L_{D_x, D_y}(\theta) \right\}.$$

From the Bayesian perspective, since the negative log likelihood loss is used, this means that

$$\hat{\theta} = \arg\min_{\theta} \left\{ -\log P(D_y|D_x, \theta) \right\} = \arg\max_{\theta} \left\{ P(D_y|D_x, \theta) \right\}.$$

This formula indicates to find the choice of θ that maximizes the probability of the true labels D_y for the data given the observed D_x values for parameters θ. Now, suppose a prior $p(\theta)$ is introduced:

$$\hat{\theta} = \arg\max_{\theta} P(D_y|D_x, \theta) p(\theta).$$

When converted back to the frequentist paradigm under log-likelihood loss, the multiplicative factor for the prior becomes an additive regularization term:

$$\hat{\theta} = \arg\min_{\theta} L_{D_x,D_y}(\theta) + reg(\theta), \quad reg(\theta) = -\log(p(\theta)).$$

Jospin et al. state that normal prior selected above is "equivalent to a weighted ℓ_2 regularization (with weights $1/\sigma$) when training a point estimate network." In this sense, the Bayesian model which utilizes priors specified in advance helps prevent overfitting simply by including priors in the first place.

2.2 Bayesian Convolutional Neural Networks

As discussed previously, Bayesian neural networks work with most functional models that work for typical neural networks. Convolutional neural networks are no exception - by simply replacing typical CNNs with Bayesian counterparts containing Bayesian neurons and the appropriate operations, a Bayesian convolutional neural network (BCNN) is created.

3 Simulation

We use an implementation of BCNNs from Github based on work from [**shridhar2019comprehensive**] [**shridhar2018uncertainty**]. It allowed for direct substitution of Bayesian layers for deterministic neural network layers such that the same training procedure could be applied to both types of networks. The normal priors previously discussed are

built into these custom layers in the package.

3.1 CIFAR-10

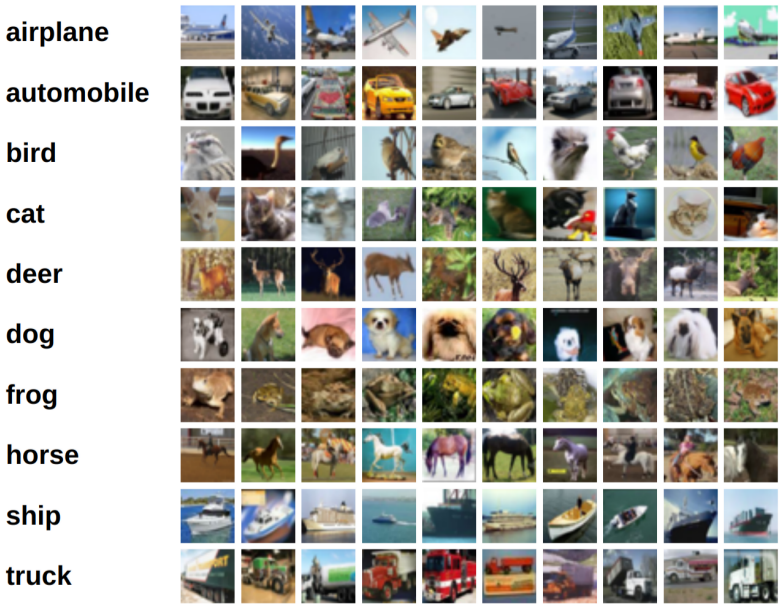


Figure 6: Example CIFAR-10 images [**cifar10**]

The CIFAR-10 (Canadian Institute For Advanced Research 10) dataset is a machine learning benchmarking set. It contains $60,000$ 32×32 RGB pictures of airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks [**cifar10**]. The ten classes within the dataset are mutually exclusive, meaning there is no image containing multiple of the objects mentioned above. The number of images is split evenly between each class, with $6,000$ images per each type of object. It is a very popular data set (in fact the most common) to use in machine learning and training neural networks, due to the extensive size, as well as the decent number of

categories. The purpose of the images within the dataset being of a lower resolution is to allow researchers to see more quickly (and with less computation effort) how well an algorithm works on classification with the dataset. Convolutional neural networks are often seen to be the “best" at classifying CIFAR-10.

3.2 Hyperparameters

The hyperparameters of a neural network, such as epochs, learning rate, regularization rate, are set before training to control the process and ensure the network achieves the best possible performance for its architecture. Epochs and learning rate were defined in Subsection 1.3 while the regularization rate and its connection to Bayesian network priors was introduced in 2.1.1. The choice of optimizer is not relevant to and beyond the scope of this work - Adamw as selected here simply provides a marginal performance improvement.

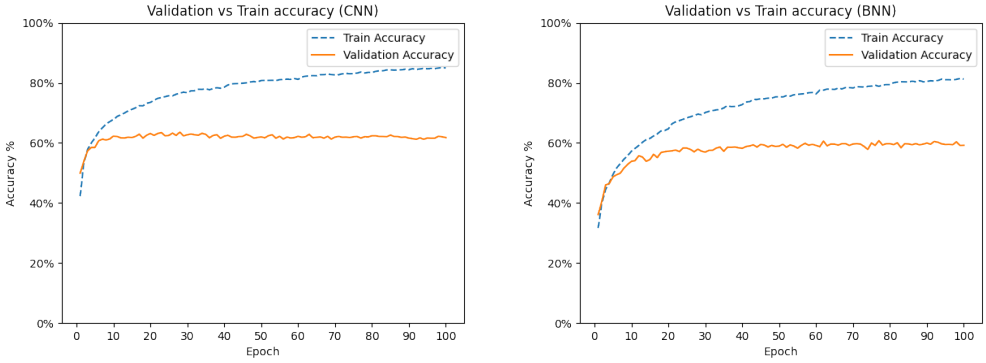
We choose to keep the number of layers and levels in the model the same to keep the models roughly equal, but tuned the hyperparameters separately. We did not adjust the priors on the BCNN, as those were set by the BCNN layer code from [**shridhar2018uncertainty**]. We found the following hyperparemeters created the best model:

Hyperparameter	CNN	BCNN
Epochs	100	100
Learning Rate	0.001	0.003
Regularization Rate	0.001	0.001
Optimizer	Adamw	Adamw

3.3 Results

Metric	CNN	BCNN
Train Accuracy	84.96%	81.27%
Validation Accuracy	61.76%	59.21%
Time to Train	16 min 11 sec	22 min 11 sec

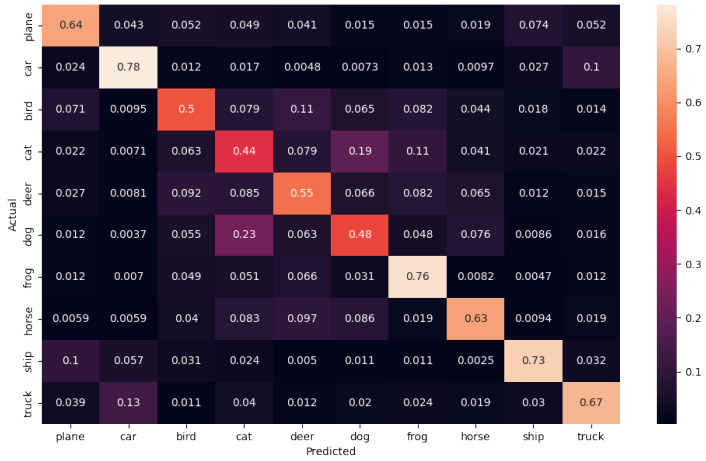
The results we achieved with training were about the same between the BCNN and the CNN. However, the bayesian model took longer to train.



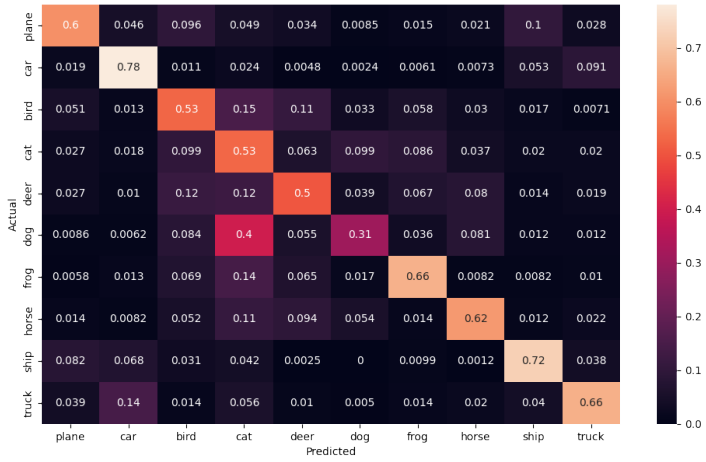
(a) CNN (b) BCNN

Figure 7: Train vs validation accuracy over time

Based on the divergent lines in figure 7, the model needs tuning. Specifically, the regularization rate needs to be raised. One interesting thing that appears in the figures, is that the BCNN took longer to diverge from the train accuracy compared to CNN.



(a) CNN



(b) BCNN

Figure 8: Confusion matrices

Based on the confusion matrix from two models in 8, the two models confuse similar sets of classes. Both models struggle to differentiate animals, but the BCNN

has poorer accuracy on the classes that are easily confused.