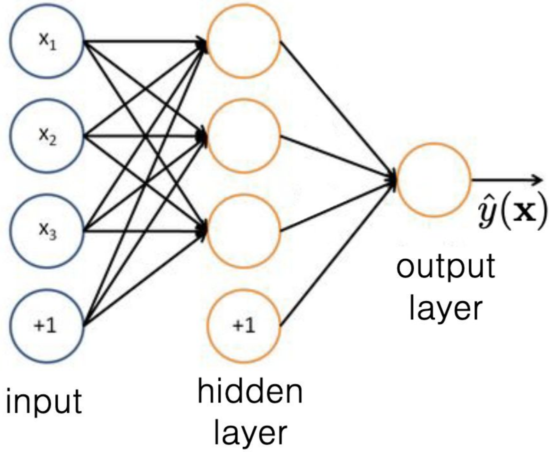
Weight Uncertainty in Neural Networks

Vinayak Bassi
Shreyas Bhat

Neural Networks 101

- An architecture that consist of input, hidden layer(s), and output layer.
  - applies a mathematical function to the input it receives and passes output to nodes in next layer.
  - similar to running several logistic regressions at the same time
- Adjusts weights and biases of connections between layers to minimize prediction loss
  - Backpropagation
- Given a set of training examples D = $(x_i , y_i)_i$

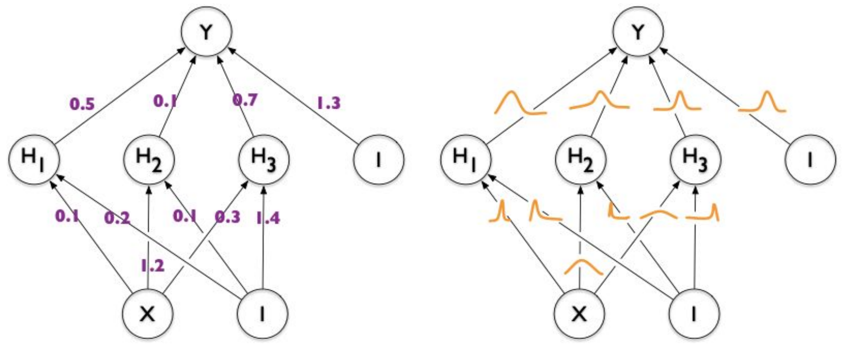


$$\begin{aligned}\mathbf{w}^{\text{MLE}} &= \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) \\ &= \arg\max_{\mathbf{w}} \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w}).\end{aligned}$$

Weights can be learnt by gradient descent (eg., Backpropagation)

Need for Regularization

- Plain feedforward neural networks are prone to overfitting
  - incapability of correctly assessing the uncertainty in the training data
  - over confident decisions about the correct class, prediction or action
- Motivation behind addition of uncertainty in weights:
  - regularisation via a compression cost on the weights
  - richer representations and predictions from cheap model averaging
  - exploration in simple reinforcement learning problems such as contextual bandits



Need for Regularization

- One way of avoiding the problem of overfitting is by introducing regularization
- Often achieved by imposing a prior on the weights of the network
- Weights are then found by maximum a posteriori estimation

$$\begin{aligned}\mathbf{w}^{\mathrm{MAP}} &= \arg\max_{\mathbf{w}} \log P(\mathbf{w}|\mathcal{D}) \\ &= \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) + \log P(\mathbf{w})\end{aligned}$$

- For example, imposing a Gaussian prior on the weights leads to L2 regularization
- This still, however, results in a point estimate of the weights
- Q: Can we efficiently learn a posterior distribution on the weights, given the data?

Background

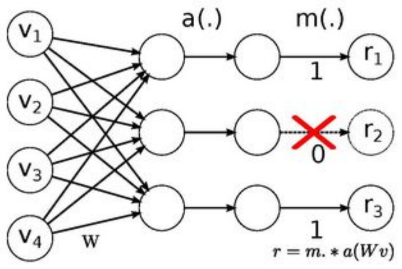
- Some regularization schemes have been used to prevent overfitting
  - Dropout
  - Early Stopping
  - Weight Decay
- Weights with greater uncertainty introduce more variability into the decisions made by the network, leading naturally to exploration
- In some research related to Deep Generative models: Variational inference has been applied to stochastic hidden units of an autoencoder.
  - However, in our case instead of applying variational inference on the hidden units we apply it on parameters (weights)

Background

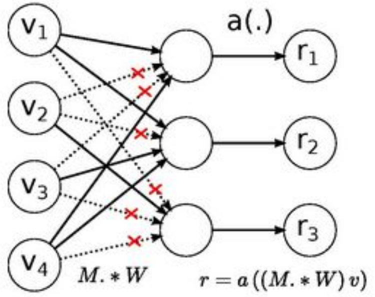
Some of the algorithms that have been reported in previous literatures that enable regularization of a neural network

1. SGD by Dropout:
   a. randomly sets some hidden units to zero during each training iteration
   b. the dropout is applied not only during training but also during the computation of the gradient
2. SGD by Drop-Connect:
   a. instead of dropping out individual units, it drops out individual weights in the connections between units

Note that the algorithm we are going to use (*Bayes by Backprop*) uses Bayesian inference on determining weights.



DropOut Network

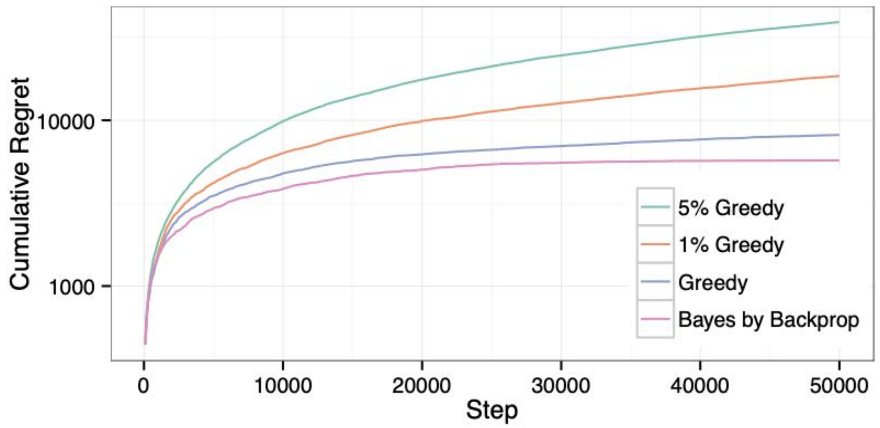


DropConnect Network

Need for Uncertainty in Weights

Contextual Bandit example:

- Problem: This is a problem of maximization of reward when an agent has to decide whether to eat or not eat given set of mushrooms.
- Rewards Schema:
  - +5 for edible mushroom
  - -35 for poisonous mushrooms (with p =0.5)
  - 0 for not choosing to eat



Bayes by Backprop quickly learns to eat edible mushrooms and avoid poisonous ones

Bayes by Backprop

- Bayesian inference for neural networks calculates the posterior distribution of the weights given the training data, P(w|D)
- Variational learning finds the parameters θ of a distribution on the weights q(w|θ) that minimises the KL divergence with the true Bayesian posterior on the weights
- Cost function:

$$P(\mathbf{w}|\mathcal{D}) \propto P(\mathcal{D}|\mathbf{w})P(\mathbf{w})$$

$$\begin{aligned}\theta^\star &= \arg\min_\theta \mathrm{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] \\ &= \arg\min_\theta \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})P(\mathcal{D}|\mathbf{w})} \mathbf{dw} \\ &= \arg\min_\theta \mathrm{KL}\left[q(\mathbf{w}|\theta) \mid\mid P(\mathbf{w})\right] - \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log P(\mathcal{D}|\mathbf{w})\right]\end{aligned}$$

Complexity Cost: Prior dependent part

$$\mathcal{F}(\mathcal{D},\theta) = \mathrm{KL}\left[q(\mathbf{w}|\theta) \mid\mid P(\mathbf{w})\right] - \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log P(\mathcal{D}|\mathbf{w})\right].$$

Likelihood Cost: Data dependent part

Bayes by Backprop

Exactly minimising the mentioned cost function naively is computationally prohibitive. Therefore we use unbiased Monte Carlo sampling to approximate a cost function as:

$$\mathcal{F}(\mathcal{D},\theta) \approx \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)}|\theta) - \log P(\mathbf{w}^{(i)}) - \log P(\mathcal{D}|\mathbf{w}^{(i)})$$

here $w^{(i)}$ denotes the ith Monte Carlo sample drawn from the variational posterior $q(w^{(i)}|\theta)$.

Notable Points:

1. The closed form of the complexity cost (or entropic part) is not used in this case.
   a. Enables us to try more combinations of prior and variational posteriors.

2. Every term of this approximate cost depends upon the particular weights drawn from the variational posterior

3. We used a proposition that stated: Under certain conditions, the derivative of an expectation can be expressed as the expectation of a derivative

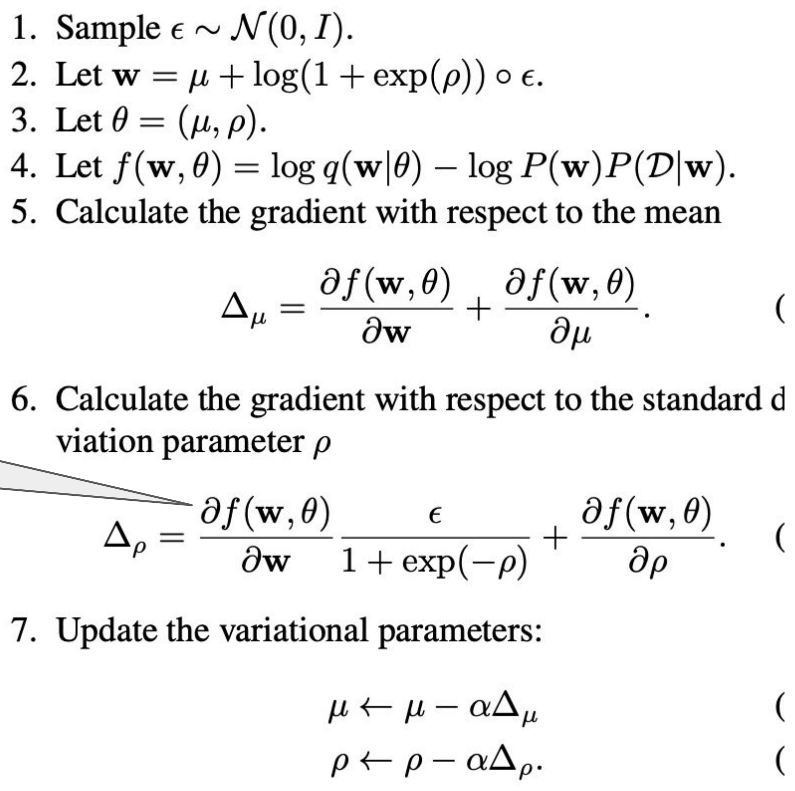
$$\frac{\partial}{\partial\theta}\mathbb{E}_{q(\mathbf{w}|\theta)}[f(\mathbf{w},\theta)] = \mathbb{E}_{q(\epsilon)}\left[\frac{\partial f(\mathbf{w},\theta)}{\partial\mathbf{w}}\frac{\partial\mathbf{w}}{\partial\theta} + \frac{\partial f(\mathbf{w},\theta)}{\partial\theta}\right]$$

Bayes by Backprop

- Variational posterior is a diagonal Gaussian distribution
- We parameterize the standard deviation pointwise as σ = log(1 + exp(ρ)) and so σ is always non-negative.
- Variational posterior parameters are θ = (μ, ρ)

Gradient term for mean and S.D are shared
(Similar to Backpropagation)

ALGORITHM



1. Sample $\epsilon \sim \mathcal{N}(0, I)$.
2. Let $\mathbf{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
3. Let $\theta = (\mu, \rho)$.
4. Let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w})P(\mathcal{D}|\mathbf{w})$.
5. Calculate the gradient with respect to the mean

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}. \qquad ($$

6. Calculate the gradient with respect to the standard d
   viation parameter ρ

$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}. \qquad ($$

7. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu \qquad ($$

$$\rho \leftarrow \rho - \alpha \Delta_\rho. \qquad ($$

Scale Mixtures for Prior [P(w)]

- A simple scale mixture prior combined with a diagonal Gaussian posterior is used.

$$P(\mathbf{w}) = \prod_j \pi \mathcal{N}(\mathbf{w}_j|0, \sigma_1^2) + (1-\pi)\mathcal{N}(\mathbf{w}_j|0, \sigma_2^2),$$

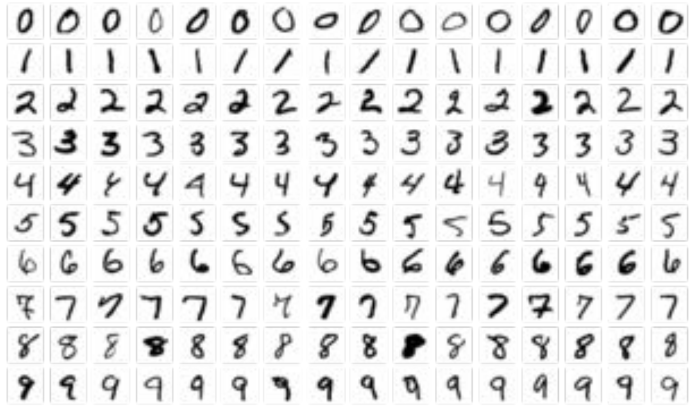
  - Here σ_1, σ_2, and π are hyperparameters for our fixed-form prior.
  - Each density is zero mean, but differing variances.
  - w_j is the jth weight of the network
  - $N(x|\mu, \sigma^2)$ is the Gaussian density evaluated at x with mean μ and variance σ^2
  - σ_1^2 and σ_2^2 are the variances of the mixture components

Q. Why don’t we optimize the fixed-form parameters or try to tune it during training?

MNIST Database

- A large dataset of images of handwritten digits
  - 60000 training images
  - 10000 testing images
- Each datapoint is a 28x28 pixel grayscale image
- Commonly used dataset for training image classifiers

- We randomly chose 50000 images from the training dataset as a training set and set the remaining 10000 as a validation set



Neural Network Architecture

- Fully connected network architecture
- Input layer with 784 units
- Two hidden layers with 1200 units each
- Output layer with 10 classes
- ReLU activation is being used after each layer

- Results in 2.4 million trainable weights in a deterministic network
- Or, 4.8 million trainable parameters

Hyperparameters

- The hyperparameters of this network are:
  - Number of samples, learning rate, minibatch size, σ_1, σ_2, and π
- The paper does not report the hyperparameters for which they have reported their results
- They only mention the set of hyperparameters for which they trained their neural network

$$\text{Number of samples} \in \{1, 2, 5, 10\}$$

$$\text{Learning rate} \in \{10^{-3}, 10^{-4}, 10^{-5}\}$$

$$\text{Minibatch size} = 128$$

$$-\log \sigma_1 \in \{0, 1, 2\}$$

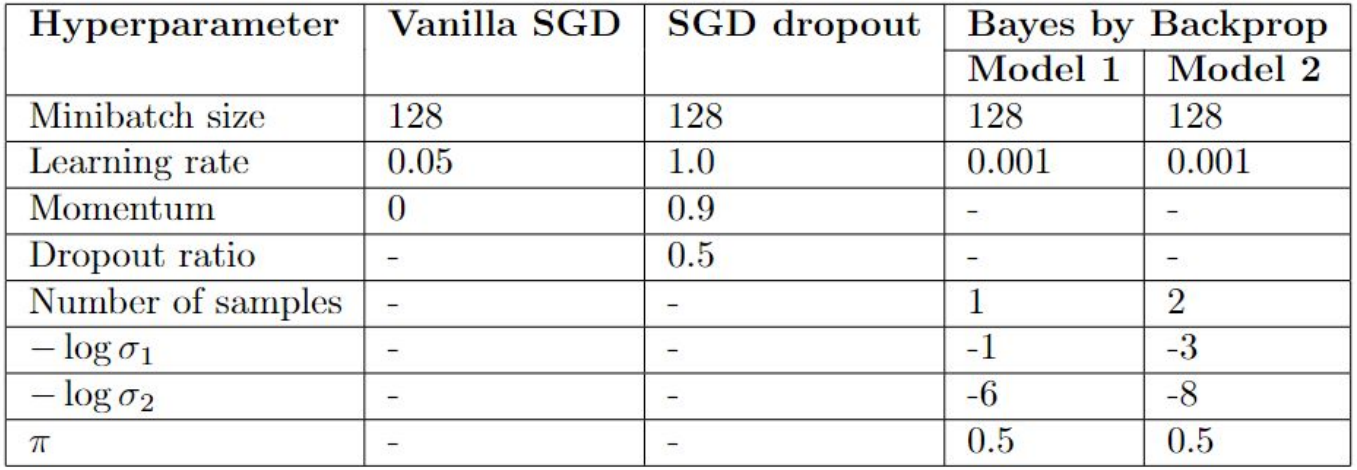
$$-\log \sigma_2 \in \{6, 7, 8\}$$

$$\pi \in \left\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\right\}$$

- This results in 324 possible combinations of hyperparameters that the authors may have used
- Due to limitations of time and resources, we could not train with all of the hyperparameters
- We tried a few combinations and report two of our best results

Results

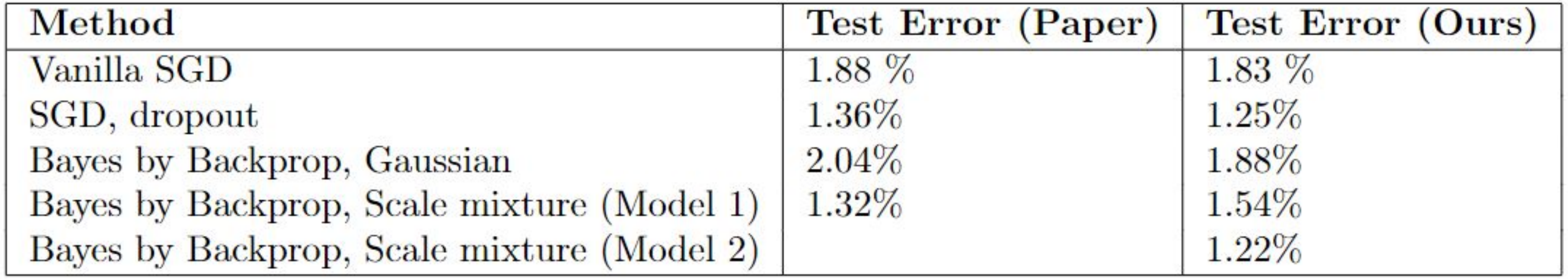
- We also replicated results from Simard et al. 2003 (vanilla SGD) and Hinton et al., 2012 (SGD with dropout) for comparison with the results from this paper
- These were easier to replicate as the hyperparameters were mentioned in their respective papers
- The table below provides the hyperparameters used for all our models



Hyperparameter	**Vanilla SGD**	**SGD dropout**	**Bayes by Backprop**	
			Model 1	**Model 2**
Minibatch size	128	128	128	128
Learning rate	0.05	1.0	0.001	0.001
Momentum	0	0.9	-	-
Dropout ratio	-	0.5	-	-
Number of samples	-	-	1	2
$-\log \sigma_1$	-	-	-1	-3
$-\log \sigma_2$	-	-	-6	-8
π	-	-	0.5	0.5

Results

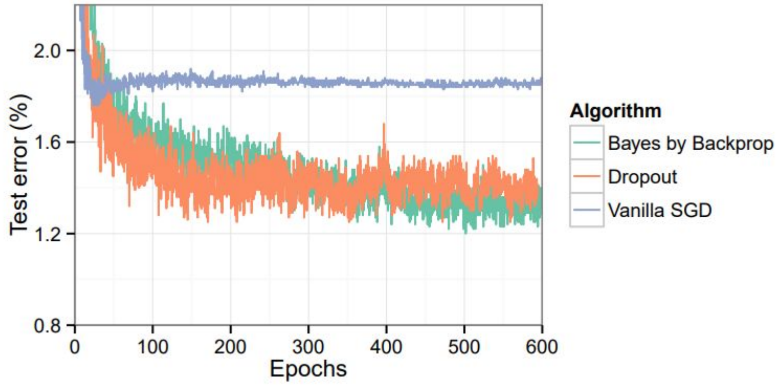
- Comparison of test errors between the paper and our model
- Our model 2 results in a better test error than model 1
- However, we will later see that model 1 is more amenable to weight pruning

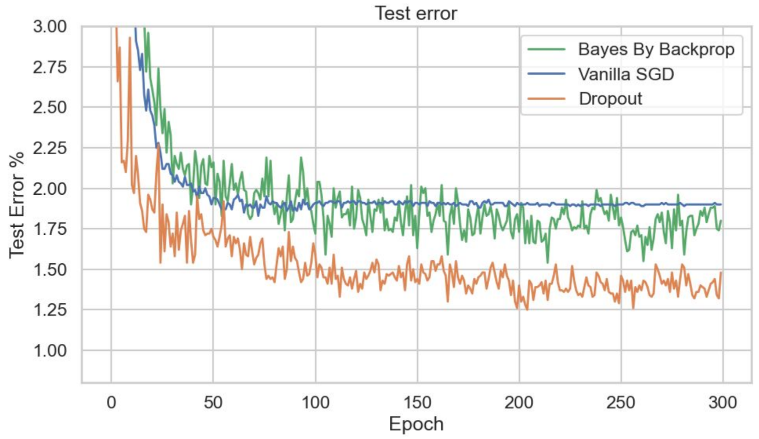


Method	Test Error (Paper)	Test Error (Ours)
Vanilla SGD	1.88 %	1.83 %
SGD, dropout	1.36%	1.25%
Bayes by Backprop, Gaussian	2.04%	1.88%
Bayes by Backprop, Scale mixture (Model 1)	1.32%	1.54%
Bayes by Backprop, Scale mixture (Model 2)		1.22%

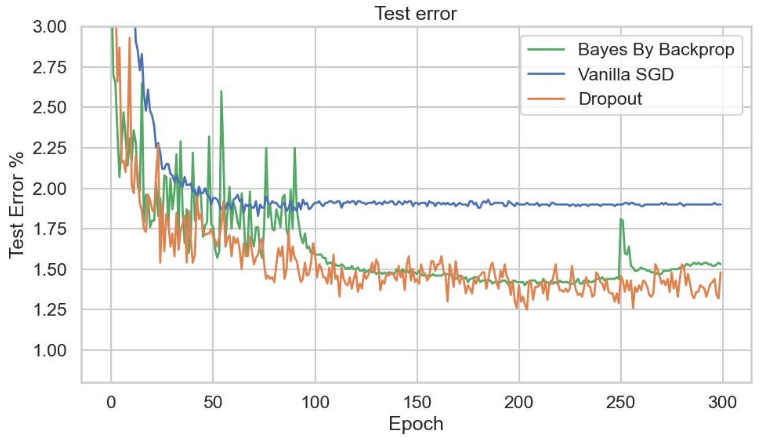
Results

- These plots show the test error as a function of the number of epochs trained
- Our model 2 resulted in a lower test error compared to model 1, and is comparable to the results shown in the paper
- Behavior shown by vanilla SGD and SGD with dropout is similar to that in the paper





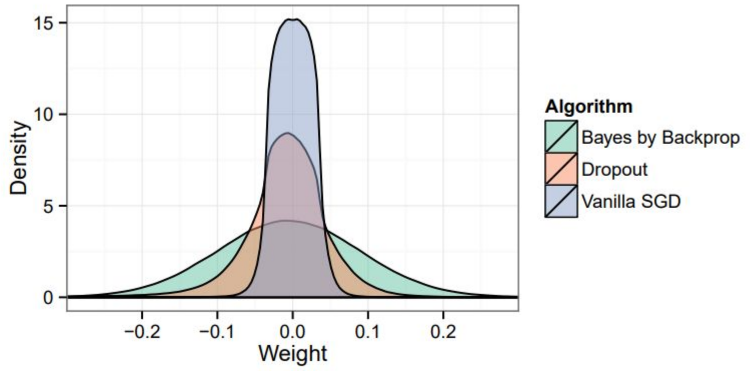
Model 1

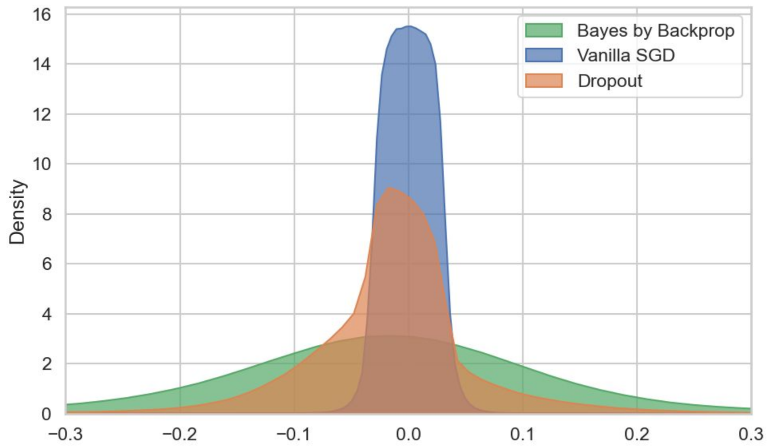


Model 2

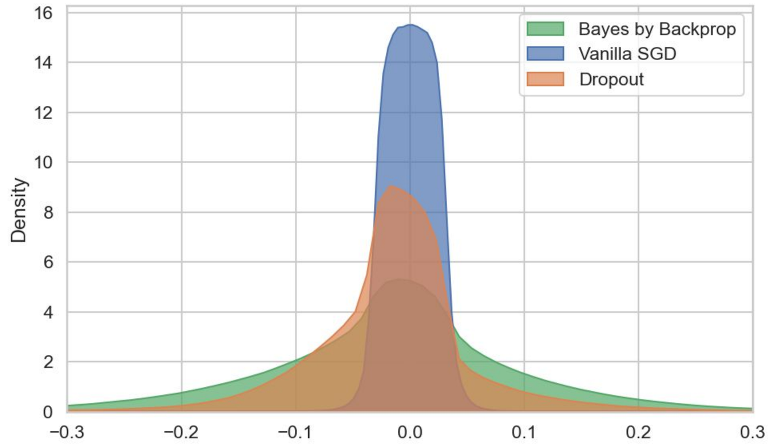
Results

- These plots show the histogram of the learned weights of the model
- Due to the introduced uncertainty, the distribution is much flatter in the Bayes by Backprop case
- We did notice that decreasing σ_1 and σ_2 results in a sharper distribution





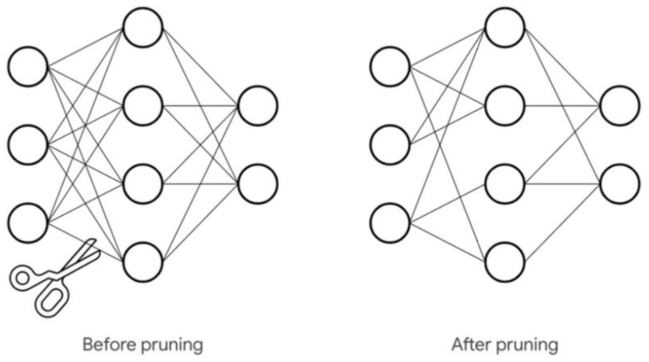
Model 1



Model 2

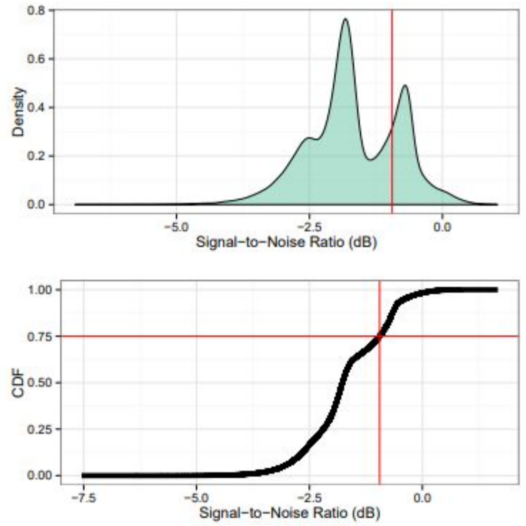
Weight Pruning

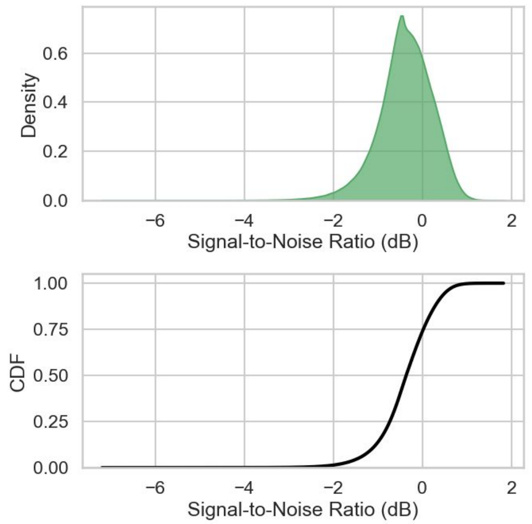
- Methodically eliminate parameters from an existing network
- Used for minimizing the resource requirements at test time
- **Goal:** Convert a large network to a smaller network with equivalent accuracy
- General procedure:
  - Determine the significance of each neuron
  - Prioritize them based on their value
  - Remove the neuron that is the least significant
  - Determine whether to prune further based on the accuracy
- In this paper, the authors use the signal to noise ratio $\frac{|\mu_i|}{\sigma_i}$ of the weights to determine the significance of each weight



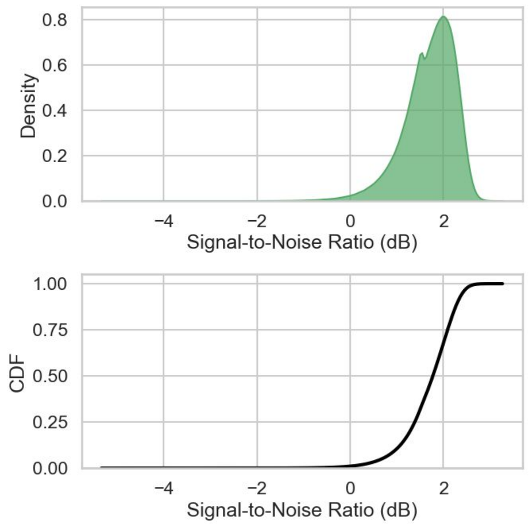
Weight Pruning

- We did not observe the two peaks in the signal-to-noise ratio distribution
- There is a small bump in the plot of model 2, which may make it more amenable to weight pruning





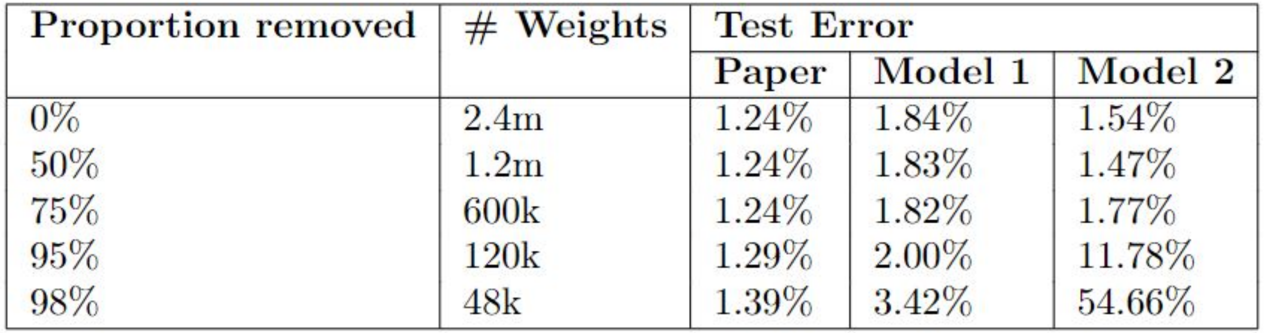
Model 1



Model 2

Weight Pruning

- The table below shows the test error as a function of the proportion of weights removed from the network
- As is seen, our models do not perform as well as the one reported in the paper
- Some possible reasons for this are:
  - Hyperparameter tuning
  - Number of epochs trained
  - Optimizer used



Proportion removed	# Weights	Test Error		
		Paper	Model 1	Model 2
0%	2.4m	1.24%	1.84%	1.54%
50%	1.2m	1.24%	1.83%	1.47%
75%	600k	1.24%	1.82%	1.77%
95%	120k	1.29%	2.00%	11.78%
98%	48k	1.39%	3.42%	54.66%

Future Work and Applications

- This general procedure can be used to convert most neural networks from a frequentist approach to a Bayesian approach
- An example given in this paper is in the case of regression curves
- A deterministic neural network (on the right) is over-confident in unobserved regions
- A stochastic neural network, on the other hand, provides wider confidence intervals, signifying the uncertainty in unobserved regions

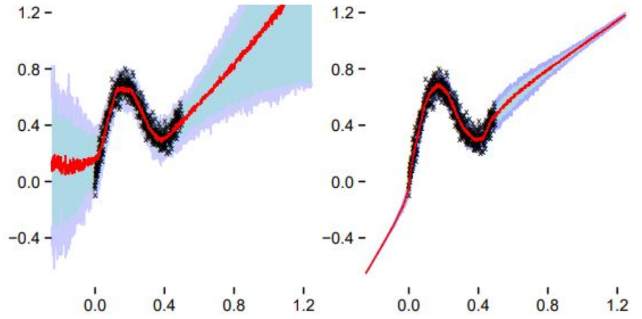


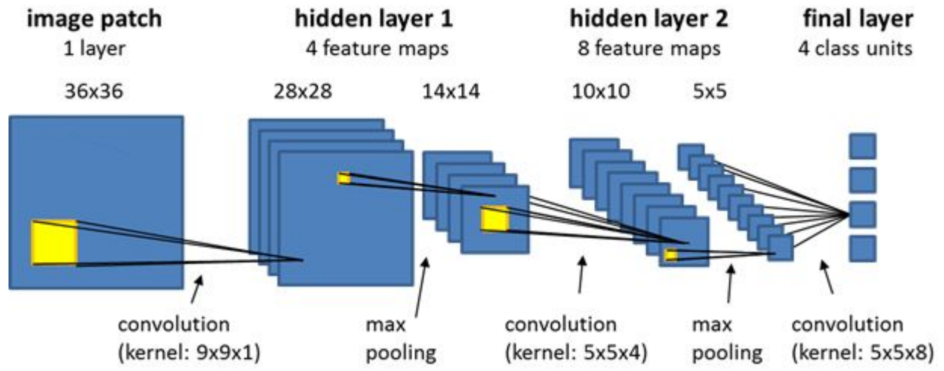
Figure 5. Regression of noisy data with interquatile ranges. Black crosses are training samples. Red lines are median predictions. Blue/purple region is interquartile range. Left: Bayes by Backprop neural network, Right: standard neural network.

$$y = x + 0.3\sin(2\pi(x+\epsilon)) + 0.3\sin(4\pi(x+\epsilon)) + \epsilon$$

$$\epsilon \sim \mathcal{N}(0, 0.02).$$

Future Work and Applications

- Convolutional Neural Networks (CNNs) are highly efficient architectures for various image processing tasks
- Gal and Ghahramani, 2016 use Bernoulli Approximate Variational Inference to reduce the number of parameters to be trained compared to this paper
  - They provide a framework for implementing this in CNNs
- Shridhar et al., 2019 provide an efficient way of implementing Bayes by Backprop on CNNs
  - Demonstrations are provided for image classification, super resolution, and generative tasks



Summary

- Introduced an algorithm ‘Bayes by Backprop’ to efficiently learn distributions on the weights of a neural network in a Bayesian way
- Provided conditions on the parameters of the variational posterior that enable Monte Carlo estimation of gradients
    - This helps to evaluate the gradients on the Monte Carlo estimates of the loss function
    - It removes the necessity of a Gaussian prior on the weights
- Demonstrated implementation of this algorithm on a fully connected network for an image classification task
    - Achieved performance similar to that of SGD with dropout, while limiting the overconfidence of the neural network
- Provided examples for regression curves and bandit problems using this algorithm

References

1. Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, & Daan Wierstra. (2015). Weight Uncertainty in Neural Networks.
2. Yarin Gal, & Zoubin Ghahramani. (2016). Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference.
3. Kumar Shridhar, Felix Laumann, & Marcus Liwicki. (2019). A Comprehensive guide to Bayesian Convolutional Neural Network with Variational Inference.