

# MCMC BinaryNet

by Danylo Ulianych

## Abstract

Training a binary neural network, using modern tools of gradient backpropagation optimization, is tricky and hard to follow. MCMC is less affected to dramatic loss changes as a model learns its hidden representation. In this work we claim that MCMC sampling is well suited in tracing the distribution space of unknown parameters in the case when a parameters space is finite and discrete as in a binary neural network.

## Introduction

Deep learning neural networks (NN) suffer from deployment scalability. Current architectures require powerful GPUs and can be hardly run in real time on smartphones. To facilitate NNs usability on wearable devices, several tricks are invented. One among them is weights binarization – a special case of NN quantization. In theory, storing 1 bit of information instead of 32-bit floating point number reduces memory storage requirements in 32 times, while boosting NN's forward pass in  $\sim 8x$  due to fast XNOR convolution operation of two binary vectors.

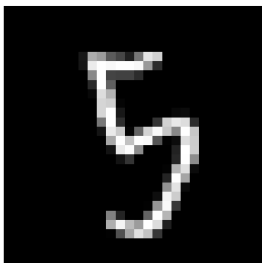
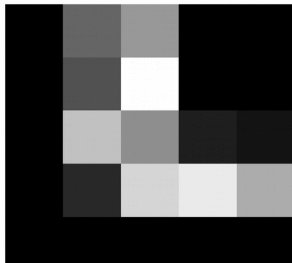
The problem in fitting binary neural networks is that derivative of sign operation is zero almost everywhere and thus the gradients are vanished immediately. MCMC sampling leverages an advantage of exploring a finite (and possibly small) space of underlying data distribution  $p(\mathbf{X})$  and weight connections  $p(\mathbf{W}_{ij})$  in the case when both data  $\mathbf{X}$  and weights  $\mathbf{W}$  are binary.

The main topic of this work is to show that training (or rather *tuning*) a binary NN with MCMC is feasible. The code is available at [https://github.com/dizcza/MCMC\\_BinaryNet](https://github.com/dizcza/MCMC_BinaryNet).

*Weights, weight connections, variables, parameters* are used interchangeably in this work.

## Data

To answer the question “Can binary neural networks be trained with MCMC sampling?”, the well-known MNIST dataset of handwritten digits (<http://yann.lecun.com/exdb/mnist/>) has been chosen. We'll use MNIST56 – a subset of MNIST that contains labels ‘5’ and ‘6’, resized to 5x5 pixels. Below is an example of 28x28 original image with label ‘5’, resized to 5x5. Number of train and test samples is 11339 and 1850 respectively.

28 x 28	5 x 5	sign(5 x 5)																																				
		<table><tr><td></td><td>[,1]</td><td>[,2]</td><td>[,3]</td><td>[,4]</td><td>[,5]</td></tr><tr><td>[1,]</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>[2,]</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>[3,]</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>[4,]</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>[5,]</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>		[,1]	[,2]	[,3]	[,4]	[,5]	[1,]	0	1	1	0	0	[2,]	0	1	1	0	0	[3,]	0	1	1	1	1	[4,]	0	1	1	1	1	[5,]	0	0	0	0	0
	[,1]	[,2]	[,3]	[,4]	[,5]																																	
[1,]	0	1	1	0	0																																	
[2,]	0	1	1	0	0																																	
[3,]	0	1	1	1	1																																	
[4,]	0	1	1	1	1																																	
[5,]	0	0	0	0	0																																	

We define sign operation as

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

We could define sign as +1 and -1 as well, but then we would need to deal with data at  $x == 0$  and introduce a dummy variable  $m$  to define weights as  $W_{ij} = 2 \cdot m_{ij} - 1, m_{ij} \sim \text{Bern}(0.5)$  (see section “Postulate a model”).

Clearly, resized images have lost visual clues what the digit it is, yet there are enough remained information to be able to distinguish digits 5 from 6, using their signs only! To illustrate this, we'll also fit a logistic regression in "Results" section.

Most of the samples (original 28x28 and resized 5x5) have zero-valued corners at (1, 1) and (5, 5). Logistic regression p-values will also indicate those variables as insignificant:  $\Pr(>|t|) = 0.35$ .

## Model

For simplicity, we'll use a fully connected linear neural network with direct connections from the input to the output. Here the input is a vector of 25 flattened binary values of 5x5 image and the output is a vector of 2 values, indicating the probability of being label 5 or 6. In total, we have  $25 \times 2 = 50$  parameters (weights). Output 0 will indicate label 5 and output 1 – label 6.

We could use a multilayer perceptron or add a few convolutional layers, but training them in a straightforward way, like we do, will take a long time. This is the main drawback of using MCMC over gradient-based parallel optimizations.

## Postulate a model

We assume that both data and weights follow a Bernoulli distribution.

$$y_i | \vec{x}_i, \hat{W} \sim \text{Bern}(h(\vec{x}_i, \hat{W})), \quad x_i^{(k)} \in \{0, 1\}, \quad k=0..24, i=1..n$$

$$h(\vec{x}, \hat{W}) = \text{softmax}(\hat{W} \cdot \vec{x}) \stackrel{\text{def}}{=} \frac{\exp(z_0)}{\exp(z_0) + \exp(z_1)} = \frac{1}{1 + \exp(-(z_1 - z_0))} \stackrel{\text{def}}{=} \text{logistic}(z_1 - z_0)$$

$$\vec{z} \leftarrow \hat{W} \cdot \vec{x}$$

$$W_{ij} \sim \text{Bern}(0.5), \quad i=0..1, j=0..24$$

In JAGS it looks like:

```
model {
  for (i in 1:length(y)) {
    y[i] ~ dbern(p[i])
    logit(p[i]) = z[i,2] - z[i,1]
    z[i,1:2] = w %*% x[i,]
  }
  for (i_output in 1:2) {
    for (j_input in 1:25) {
      w[i_output, j_input] ~ dbern(0.5)
    }
  }
}
```

Prior Bernoulli distribution for weights is justified by the constraint that weights have to be binary. Beta prior also works if we average (along iteration axis) and threshold posterior coefficients

to be greater than 0.5 as follows:  $w = \begin{cases} 1 & \text{if } \text{mean}(w) > 0.5 \\ 0 & \text{otherwise} \end{cases}$  (rule 1)

## Fit the model

As we've constrained our weights to be binary, we expect a relatively small number of iterations will suffice to reach its stationary distribution (at least in order of magnitude smaller than, for example, if we choose continuous (normal) distribution with a large variance). Fitting a postulated model is straightforward in JAGS:

```
mod = jags.model(/path/to/model_def, data=data_train, n.chains=3)
mod_sim = coda.samples(model=mod, variable.names = 'w', n.iter = 500)
mod_csim = as.mcmc(do.call(rbind, mod_sim))
```

## Check the model

We ran 3 different chains each of 500 iterations. Chain convergence diagnostic is made with the help of accuracy VS iteration plot, as it's usually done in machine learning. The appealing side of hierarchical (layered) binary weights representation is that, once converged, each draw of all

weights can be used as an optimal representation, no need to average over iterations. Profound analysis of weight trace plots shows that although at some iteration several variables can flip their signs (we note, it happens rarely – most of candidate proposals are rejected), it doesn't change the overall performance. To confirm this hypothesis, we monitor accuracy of

- 1) parameters, averaged over iterations  $1 \dots i$  and applied *rule 1*;
- 2) parameters at iteration  $i$

for all  $i=1 \dots n$  (only the former experiment is shown in fig. 1). These accuracies as well as convergence iteration point appear to be the same for both modes within a chain. Each chain reaches its own stationary distribution, as it's seen in fig. 1, suggesting the existence of many more similar (in terms of accuracy) stationary distributions. In fig. 1 you can also see how the accuracy is saturated after 100 epochs for all chains.

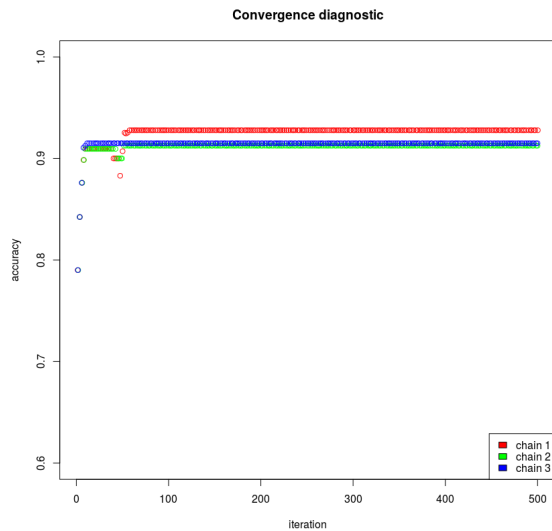


Figure 1. Convergence diagnostic.

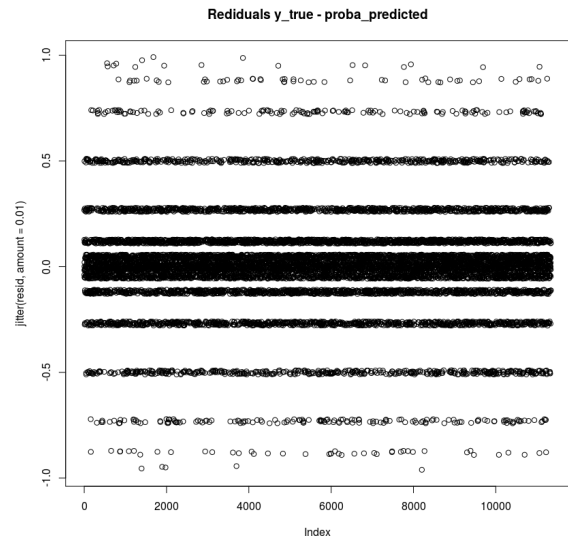


Figure 2. Residuals.

Residuals are shown in figure 2 as the difference between true labels  $y$  and fitted probabilities.

## Results

After a 100 of iterations the model is capable to correctly identify 91.4 % of the test digits, which were not present in the train data. Similar train accuracy of 91.5 % tells us the model has converged and generalizes well. BTW, generalization is not an issue for such a simple binary NN since signal binarization loses (or rather compresses) information and can be viewed as a dropout. Gradient-based optimizations converge such a binary model in a few epochs, yielding the same accuracy of ~91 % (shown [here](#) as “MNIST56 TrainerGradBinary Mar-23”; they keep a model copy in full precision and apply sign operation on model weights before each forward pass.) A non-informative logistic regression `glm(y ~ x, data=data_train)` applied to our data yields 94.6 % test accuracy without any constraints on the weight values.

## Conclusions

The fact that two completely different optimization techniques – widely used gradient-based backpropagation and our MCMC sampling – reach the same level of performance confirms that our model has reached its maximum capability, justifying the choice of MCMC sampler to train binary neural networks.

Furthermore, Bayesian techniques gives us not only posterior mean parameters but also their full distribution that makes the results more interpretable and provides a powerful tool to assess our confidence in estimated coefficients.