CS 6190: Probabilistic Machine Learning Spring 2022

Homework 5

Handed out: 15 Apr, 2022
Due: 11:59pm, 2 May, 2022

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 10 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.

Practice [100 points + 70 bonus]

1. [30 points] For warm-up, let us deal with the same scalar distribution in the last homework,

$$p(z) \propto \exp(-z^2)\sigma(10z+3).$$

You will implement MCMC algorithms to sample from this distribution. To reach the burn-in stage, please run your chain for $100K$ iterations(i.e., generate $100K$ samples). Then continue to run $50K$ iterations, pick every 10-th sample to obtain your final samples. Set your initial sample to 0.

(a) [14 points] Implement Metroplis-Hasting, with Gaussian proposal, $q(z_{n+1}|z_n) = \mathcal{N}(z_{n+1}|z_n, \tau)$. Vary τ from $\{0.01, 0.1, 0.2, 0.5, 1\}$. Run your chain. For each setting of τ, record the acceptance rate (i.e., how many candidate samples are accepted/the total number of candidate samples generated). Draw a figure, where the x-axis represents the setting of τ, and y-axis the acceptance rate. What do you observe? For each setting of τ, draw a figure, show a normalized hist-gram of the $5K$ samples you collected. Please set the number of bins to 50. If you use Python, please use matplotlib and look up the API at `https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.hist.html`. You can set the parameter "bins" to 50 and "density" to true. Also, please draw the ground-truth density curve (obtained via quadrature — you did that in the last homework). Now what do you observe?

<u>**Answer**</u>

You can see the results in Figures 1, 2, 3, 4 and 5. Apart from this, you can see the acceptance ratios for all cases in Figure 6. We can observe the acceptance ratio decreasing with increasing τ.

(b) [14 points] Implement Hybrid Monte-Carlo sampling with Leapfrog. Let us fix $L = 10$, and vary ϵ from $\{0.005, 0.01, 0.1, 0.2, 0.5\}$. Run your chain. Similar to the above, for each setting of ϵ, record the acceptance rate,and draw a figure showing ϵ *v.s.* acceptance rate. What do

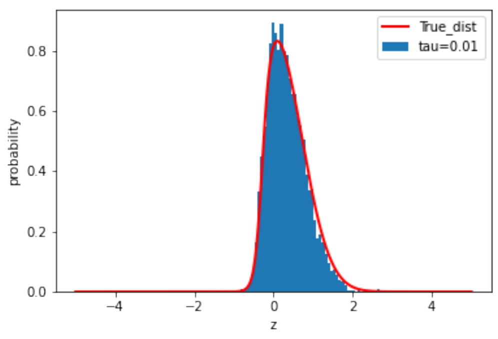


Figure 1: Metroplis Hasting with tau 0.01.

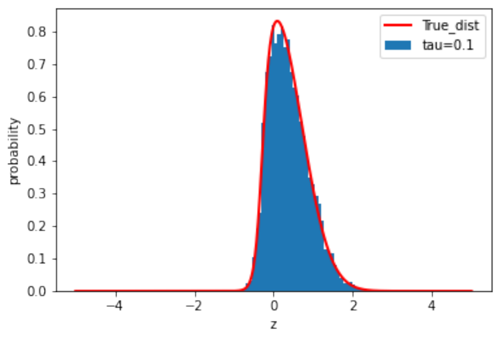


Figure 2: Metroplis Hasting with tau 0.1.

you observe? For each setting of ϵ, draw the normalized hist-gram (50 bins)of collected $5K$ samples. what do you observe? You can leverage the third-party implementation of leap-frog algorithm if you feel it is too difficult to implement it by yourself. One example is given by `https://people.sc.fsu.edu/~jburkardt/py_src/leapfrog/leapfrog.py`. However, using the third-party implementation of HMC (*e.g.,* hamiltorch) is NOT allowed. We expect you do understand and are able to implement the algorithmic steps of HMC.

Answer

You can see the results in Figures 7, 8, 9, 10 and 11. Apart from this, you can see the acceptance ratios for all cases in Figure 12. We can observe the acceptance ratio decreasing with increasing τ, however the decrease is not that steep here.

(c) [2 points] Now compare the results from the two MCMC algorithms, what do you observe and conclude?

Answer

From the above results we can see the acceptance ratio decreasing with increasing τ or ϵ value. This does not mean that we should set these values to very small numbers because we get very poor approximations if we do so. Hence we have to have a happy middle ground.

2. [20 points] Let us work with a 2-dimensional Gaussian distribution,

$$p(z_1, z_2) = \mathcal{N}(\left[\begin{array}{c} z_1 \\ z_2 \end{array}\right] | \left[\begin{array}{c} 0 \\ 0 \end{array}\right], \left[\begin{array}{cc} 3 & 2.9 \\ 2.9 & 3 \end{array}\right])$$

(a) [2 point] Draw 500 samples from this distribution and show the scatter plot. What do you observe?

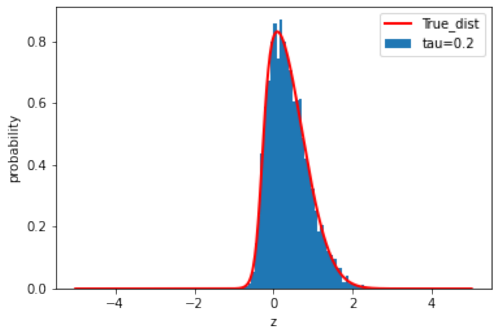


Figure 3: Metroplis Hasting with tau 0.2.

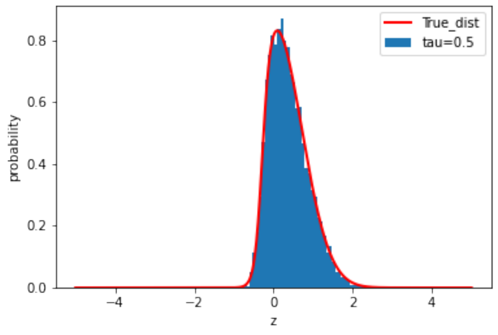


Figure 4: Metroplis Hasting with tau 0.5.

Answer

From Figure 13, we can see what the samples from the defined distribution look like.

(b) [10 points] Implement Gibbs sampling to alternatively sample z_1 and z_2. Set your initial sample to $(-4, -4)$. Run your Gibbs sampler for 100 iterations. Draw the trajectory of the samples. What do you observe?

Answer

Now, let's try to sample using Gibbs Sampling. The result from this method are shown in Figure 14. To observe any differences, let's look at true samples and gibbs samples together as shown in Figure 15. From this figure, we can see that the coverage isn't that good. It's located in a small area.

(c) [8 points] Implement HMC with Leapfrog, set $\epsilon = 0.1$ and $L = 20$. Run your HMC for 100 iterations. Set your initial sample to $(-4, -4)$. Draw the trajectory of the samples. What do you observe? Compare with the results of Gibbs sampling, what do you conclude?

Answer

Now, let's try to sample using Monte Carlo Sampling. The result from this method are shown in Figure 16. To observe any differences, let's look at true samples, Gibbs samples along with Monte Carlo sampling as shown in Figure 17. From this figure, we can see that the coverage is good here as compared to Gibbs Sampling. This is the main difference, however it takes a lot of time to run.

3. [70 points] Let us work on a real-world dataset we have met before. Please download the data from the folder"data/bank-note". The features and labels are listed in the file "data-desc.txt". The

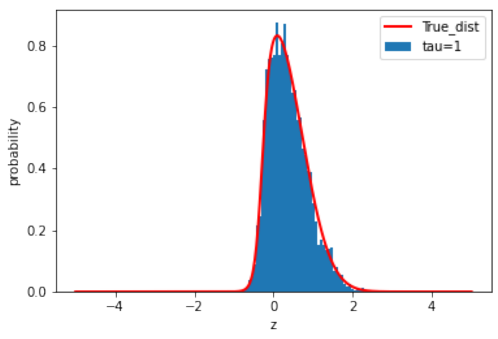


Figure 5: Metroplis Hasting with tau 1.

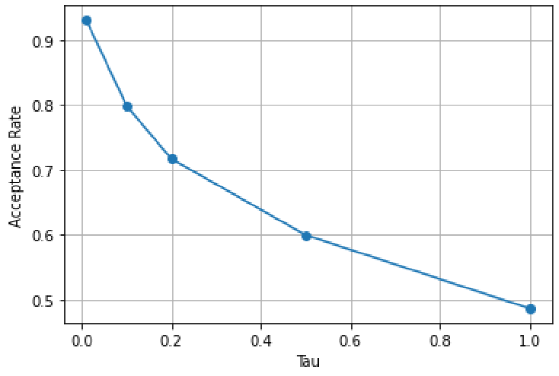


Figure 6: Acceptance Ratios.

training data are stored in the file "train.csv", consisting of 872 examples. The test data are stored in "test.csv", and comprise of 500 examples. In both the training and testing datasets, feature values and labels are separated by commas.

(a) [20 points] We assign the feature weight vector $\mathbf{w}$ a standard normal prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Write down the joint probability of the Bayesian logistic regression model. Now, implement HMC with Leapfrog. Set your initial sample to be $\mathbf{0}$. Run your chain for $100K$ iterations to reach the burn-in state; then continue to run $10K$ iterations, pick every 10-th sample to obtain your posterior samples. Vary ϵ from $\{0.005, 0.1, 0.2, 0.5\}$ and L from $\{10, 20, 50\}$. As we did before, we can test the predictive accuracy and predictive log-likelihood. Both can involve the posterior distribution of the weights. How? To evaluate the accuracy, for each posterior sample of $\mathbf{w}$ in hand ($1K$ in total), we can use it to make the predictions (1 or 0) on all the test examples and calculate the accuracy. Then we average the prediction accuracy of all the posterior samples of $\mathbf{w}$. In the same way, we can compute the predictive likelihood. Now you can sense how convenient with posterior samples — the integration over posterior distribution is turned to the average across posterior samples! List a table showing different combinations of ϵ and $\mathbf{L}$ and the resulting predictive accuracy, predictive log-likelihood, and acceptance rate. What do you observe and conclude?

<u>**Answer**</u>

The result can be seen in the Table 1. From these results we can conclude that a proper choice of ϵ is needed for algorithm i.e. it should be small. However, there is a wide range of ϵ values between 0.005 and 0.02 for which there's convergence but smaller the value, the better the accuracy. At $\epsilon = 0.05$, we don't see any convergence. As far as acceptance rate is concerned, it

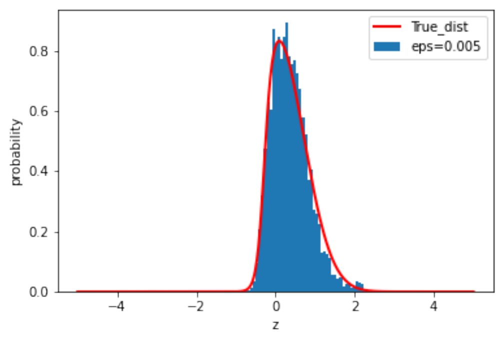


Figure 7: Monte Carlo with eps 0.005.

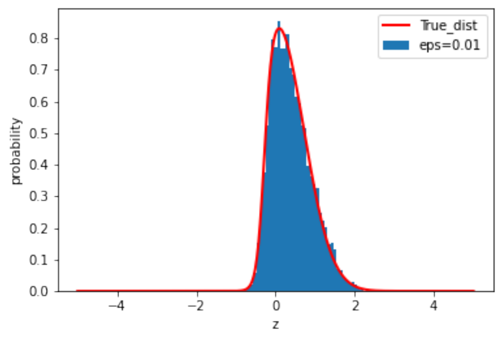


Figure 8: Monte Carlo with eps 0.01.

looks to be dependent on the combination of values of ϵ and L.

ϵ	L	Accuracy	Likelihood	Acceptance Rate
0.005	10	99.00	-0.04	0.001400
0.005	20	99.02	-0.03	0.050300
0.005	50	96.60	-0.09	0
0.01	10	99.06	-0.03	0.041400
0.01	20	98.97	-0.03	0.126800
0.01	50	98.98	-0.03	0.087400
0.02	10	98.95	-0.03	0.126500
0.02	20	98.98	-0.03	0.078500
0.02	50	98.97	-0.03	0.1066
0.05	10	44.20	-0.69	0
0.05	20	44.20	-0.69	0
0.05	50	44.20	-0.69	0

Table 1: Results Table.

(b) [20 points][**Bonus**]. We will implement Gibbs sampling for linear classification. However, Bayesian logistic regression does not allow tractable conditional posteriors. So we will use the Bayesian probit model with augmented variables. We have discussed it before in our class, if you

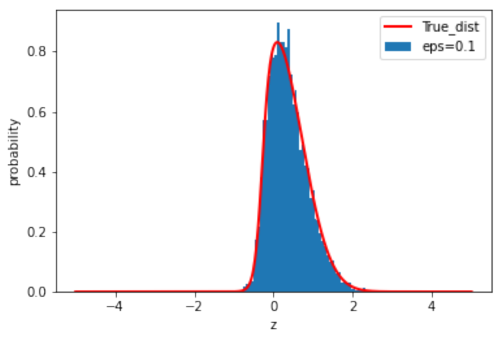


Figure 9: Monte Carlo with eps 0.1.

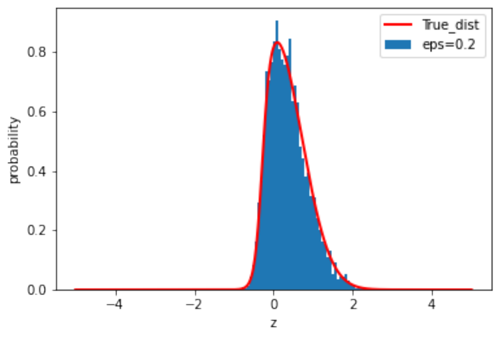


Figure 10: Monte Carlo with eps 0.2.

cannot remember, please check our slides regarding generalized linear models. Again, we will assign a standard normal prior over $\mathbf{w}$. For each training sample n, we introduce an auxiliary variable z_n. The joint probability is given by

$$p(\mathbf{w}, \mathbf{z}, \mathcal{D}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{I}) \prod_n \mathcal{N}(z_n|\mathbf{w}^\top \mathbf{x}_n, 1) \mathbb{1}\big((2y_n - 1)z_n \geq 0\big) \tag{1}$$

where $\mathbf{z} = [z_1, z_2, \ldots, z_N]^\top$, and $\mathbb{1}(\cdot)$ is the indicator function. Note that if we marginalize out $\mathbf{z}$, we will recover the original Probit model,

$$\begin{aligned} p(\mathbf{w}, \mathbf{z}, \mathcal{D}) &= \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{I}) \prod_n \phi\big((2y_n - 1)\mathbf{w}^\top \mathbf{x}_n\big) \\ &= \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{I}) \prod_n \mathrm{Bern}\big(y_n|\phi(\mathbf{w}^\top \mathbf{x}_n)\big), \end{aligned} \tag{2}$$

where $\phi(\cdot)$ is the CDF of the standard normal distribution, $\phi(x) = \int_{\infty}^{x} \mathcal{N}(t|0, 1)\mathrm{d}t$. Note that Bayesian logistic regression just replaces $\phi(\cdot)$ by the Sigmoid activation function. Now implement your Gibbs sampling algorithm based on the augmented version (1). Alternatively sample $\mathbf{w}$ and each z_n. Note that the conditional posterior of each z_n will be a truncated Gaussian. You can use `scipy.stats.truncnorm` to generate samples (or implement by yourself). Before coding, please list your derivation of the conditional posteriors. Run your chain for $100K$ iterations to reach the burn-in stage; then continue to run $10K$ iterations, pick every 10-th sample to obtain your final posterior samples. Now compute and report the predictive accuracy and log-likelihood with your posterior samples. Note that your predictive log-likelihood should be based on the

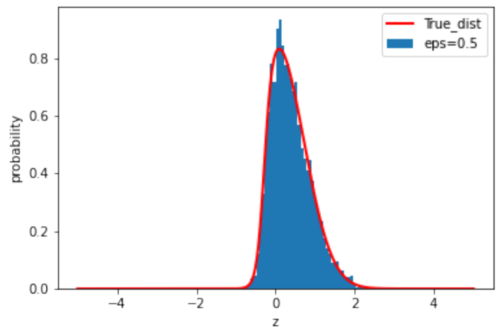


Figure 11: Monte Carlo with eps 0.5.

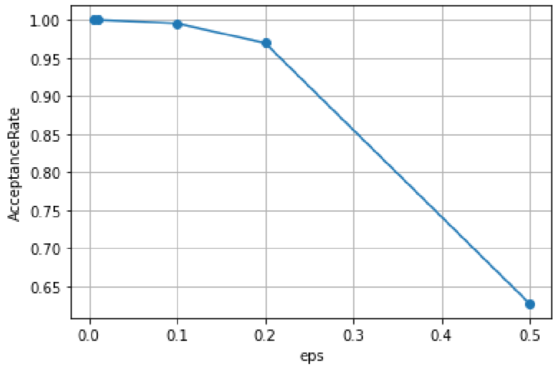


Figure 12: Monte Carlo acceptance ratios.

original model (2), rather than the augmented one. How does the performance compare with HMC on Bayesian logistic regression model?

(c) [28 points] Finally, we will implement a Bayesian neural network (NN). We will use two intermediate layers, and each layer has the same number of nodes. Vary the number of nodes from $[10, 20, 50]$. Vary the activation function from {`tanh`, `RELU`}. We will use a factorized Gaussian posterior for the NN weights (check out the slides). We will assign a standard normal prior over each weight. The output of the NN will be thrown into a Sigmoid activation function, with which we obtain a Bernoulli likelihood. Please use PyTorch or TensorFlow to implement the `Bayes by BP` algorithm based on the reparameterization trick plus stochastic optimization. If you prefer other automatic differential library (*e.g.,* JAX) , that is totally OK. But please do NOT implement BP by yourself — it is a waste of time. Note that you do not need to sample mini-batches in this problem — because the training set is small. Please use Adam algorithm for stochastic optimization. You can tune the base learning rate from {1e-3, 0.5e-3, 1e-4, 1e-5} to find the best result for each layer-width and activation function setting. Initialize posterior mean and variance of each weight to be 0 and 1, respectively. Run the Adam algorithm for 1,000 iterations. For each layer width and activation function setting, calculate the predictive log-likelihood and accuracy. How? Use Monte-Carlo approximation. Check out the slides. Use the variational posterior to generate 100 samples for the weight vector; With each sample, you can calculate the prediction accuracy and log-likelihood on the test dataset; finally, you report the average results. To check the behaviour of your BNN, let us pick up one setting: the number of node is 20 and the activation function is `tanh`. For each iteration, please use the current posterior mean of the weights to compute the average log-likelihood on the training set and test set respectively. Draw two plots, one plot showing how the training log-likelihood varies along

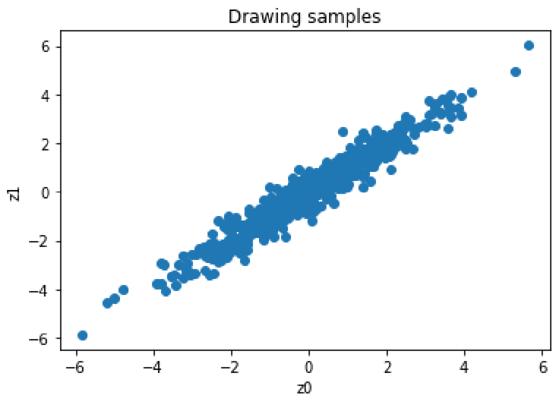


Figure 13: Sampling 500 points from the defined distribution.

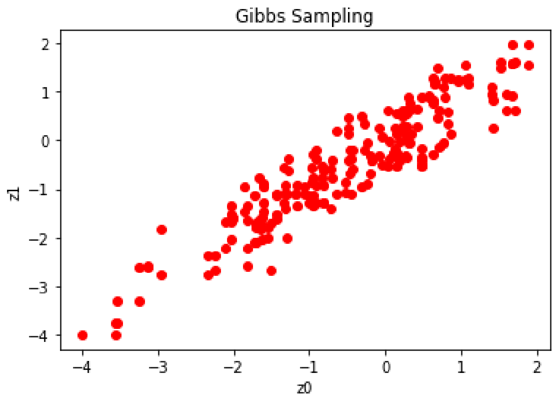


Figure 14: Sampling using Gibbs Sampling.

with the number of iterations, the other showing how the test log-likelihood varies along with the number of iterations. What do you observe and conclude?

Answer

After learning rate optimisation, I found 0.001 as the best. Using this learning rate the results can be seen in Table 2, and the plots are shown in Figure 18, 19, 20, 21, 22, 23. From these we can see that the BNN results are better and the reason for that is the non linearity. Apart from this, I also saw that BNN trains slowly first and then the accuracy shoots suddenly. The more neurons, the early this shoot happens.

Activation	Hidden Neurons	Accuracy	Likelihood
relu	10	98.3460	-0.1529
relu	20	99.3380	-0.1019
relu	50	99.6260	-0.0790
tanh	10	99.3360	-0.0222
tanh	20	99.8940	-0.0093
tanh	50	99.9280	-0.0054

Table 2: BNN Results Table.

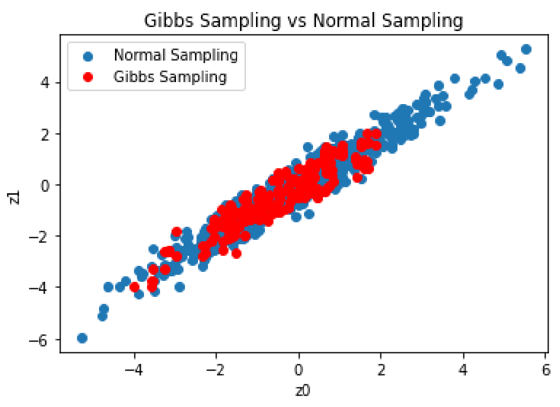


Figure 15: True Sampling vs Gibbs Sampling.

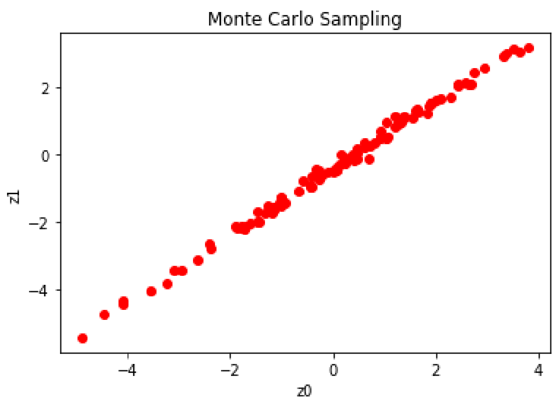


Figure 16: Sampling using Monte Carlo Sampling.

(d) [2 points] Compare the results from Bayesian linear classification and NN models. What do you observe and conclude?

Answer

Bayesian NN works better than Bayesian linear classification because of the non-linearity added at each layer. This leads to a model which can learn highly non linear structure in data, however needs hyperparameter tuning.

4. [50 points][**Bonus**] If you feel addictive to neural networks, this is a good chance to do more state-of-the-art work. Please come to the website `http://yann.lecun.com/exdb/mnist/` and download the MNIST dataset provided by LeCun et. al. It is a famous benchmark dataset. Your goal is to classify hand-written digits. The dataset includes 60,000 training and 10,000 testing pixel images of size 28×28. Each image is labelled with its ground-truth number (0-9 inclusive). Please preprocess the data by dividing each pixel values by 126. You will implement two versions of Bayesian neural networks. To select the hyper-parameters, please from the training data randomly select 50,000 digits for training and use the remaining 10,000 digits for validation. After the best hyper-parameters are identified, you train on the whole 60,000 digits again. Your NN output will be 10 dimensional, which are used to construct the categorical likelihood (i.e., softmax) — check out the slides if you are unclear. Please use Adam for stochastic optimization. We consider to tune the (base) learning rate from {1e-3, 1e-4, 1e-5}. We will use two hidden layers, and each layer has the same number of nodes.

(a) [15 points] First, we consider to place an independent standard normal prior over each weight. We use factorized Gaussian posterior (check the slides). Implement your Bayes by BP algorithm.

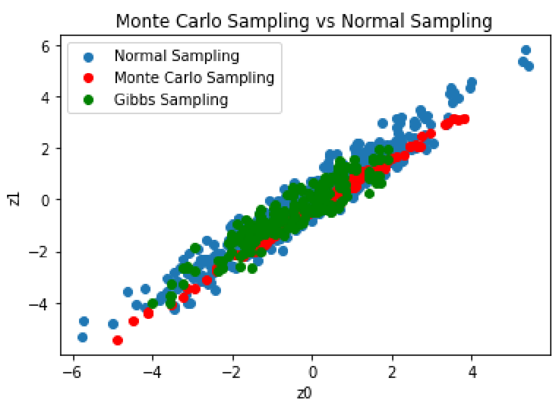


Figure 17: True Sampling vs Gibbs Sampling vs Monte Carlo Sampling.

Please run your algorithm for 1,000 epochs and report the predictive accuracy. The way to compute the predictive accuracy is the same as in Problem 3. Use your variational posterior of the weights to sample 10 sets of weights, each weight set are used to predict the labels of the test samples (you choose the label with the largest probability in your model as the prediction) and calculate the accuracy; finally you average the 10 accuracy values. Vary the number of nodes in each hidden layer from $\{400, 800, 1200\}$. Try both the `tanh` and `RELU` activation functions. Report the prediction accuracy for each combination.

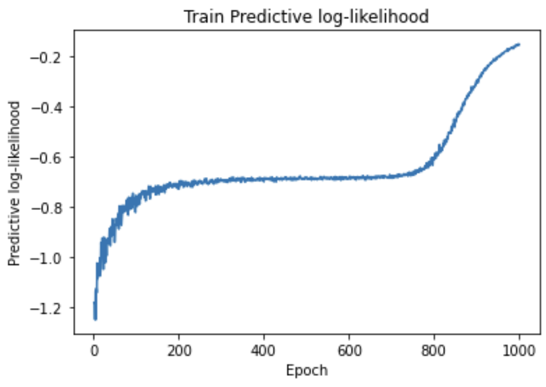
(b) [20 points] Then we consider to assign a spike and slab prior over each weight,

$$p(w_i) = \pi\mathcal{N}(w_i|0, \sigma_1^2) + (1-\pi)\mathcal{N}(w_i|0, \sigma_2^2)$$

where we tune $-\log(\sigma_1) \in \{0, 1, 2\}$ and $-\log(\sigma_2) \in \{6, 7, 8\}, \pi \in \{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$. We still use the factorized Gaussian posterior. Implement your Bayes by BP algorithm. Run your algorithm for 1,000 epochs. Report the predictive accuracy when the number of nodes in each hidden layer is in $\{400, 800, 1200\}$, and the activation function is `tanh` or `RELU`. Do you observe many weights have very small posterior variances and means close to 0? Why did this happen?

(c) [10 points] Implement the vanilla NN training with SGD — namely, we only perform MAP estimation. Run 1,000 epochs. Report the prediction accuracy for the same settings as above. Are your results consistent with Table 1 in the paper `https://arxiv.org/pdf/1505.05424.pdf`? Compared with the two version of BNNs, what do you observe and conclude?

(d) [5 points] Fix the number of nodes per layer to 1,200 and use `RELU` activation function. For each algorithm, use the validation dataset to choose the best hyper-parameters (e.g., learning rate). With the best hyper-parameter(s), re-run your algorithms on the whole training set, and draw the learning curve — how does the test accuracy vary along with the number of epochs. Is it consistent with Fig. 2 in the paper `https://arxiv.org/pdf/1505.05424.pdf`? What do you conclude and observe?



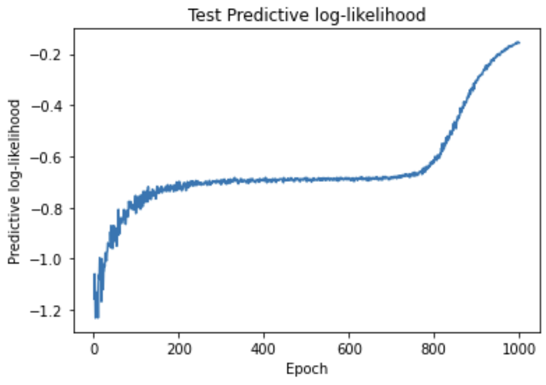


Figure 18: ReLU Activation with 10 hidden neurons.

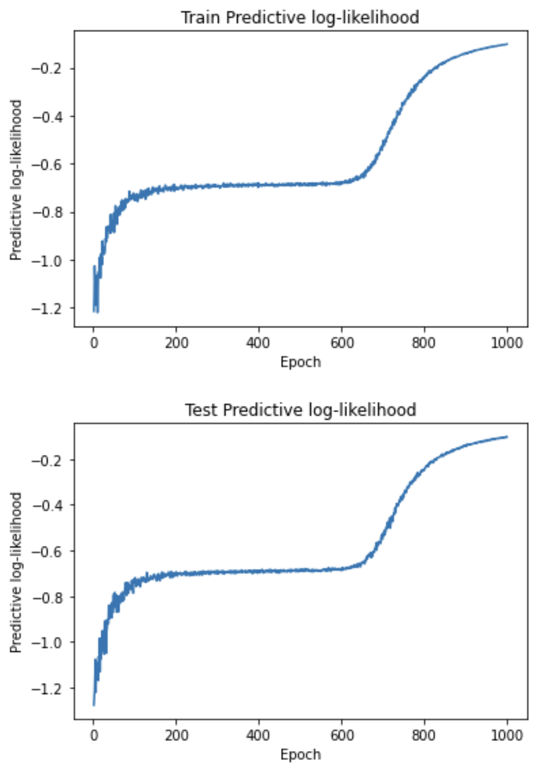


Figure 19: ReLU Activation with 20 hidden neurons.

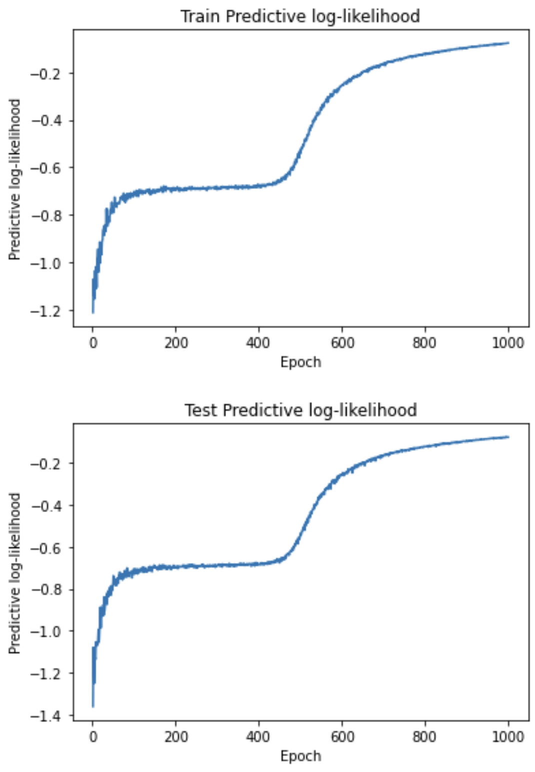


Figure 20: ReLU Activation with 50 hidden neurons.

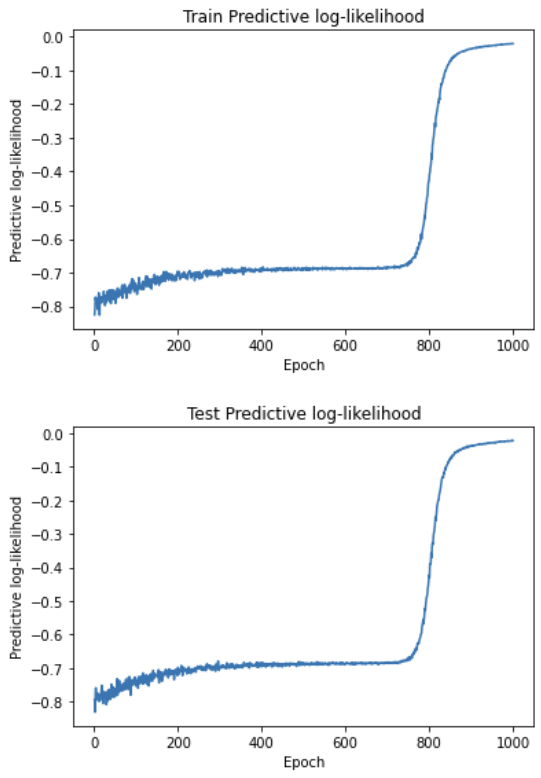


Figure 21: tanh Activation with 10 hidden neurons.

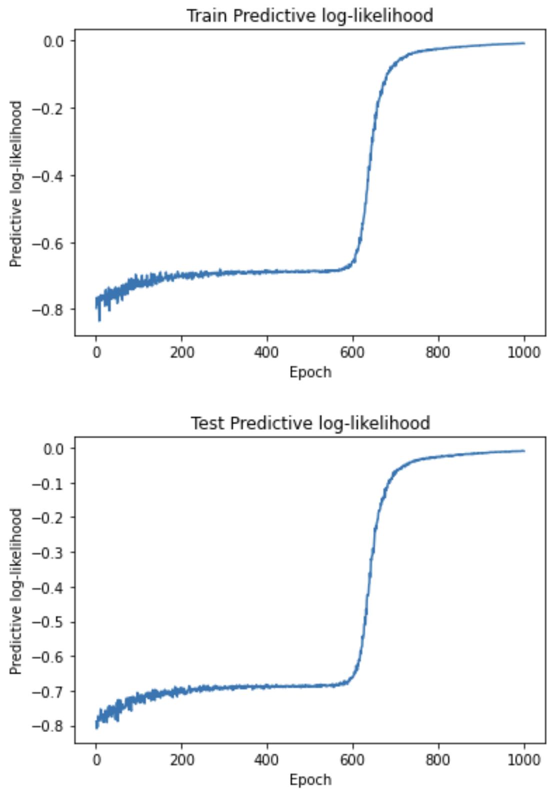


Figure 22: tanh Activation with 20 hidden neurons.

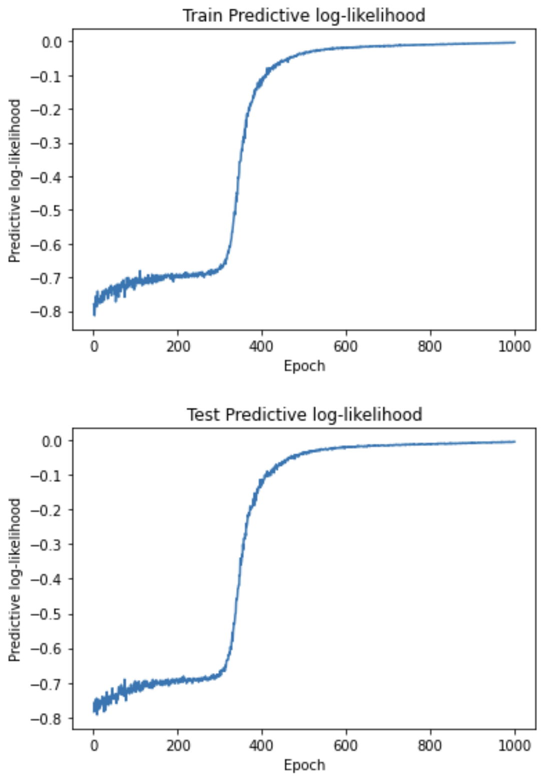


Figure 23: tanh Activation with 50 hidden neurons.