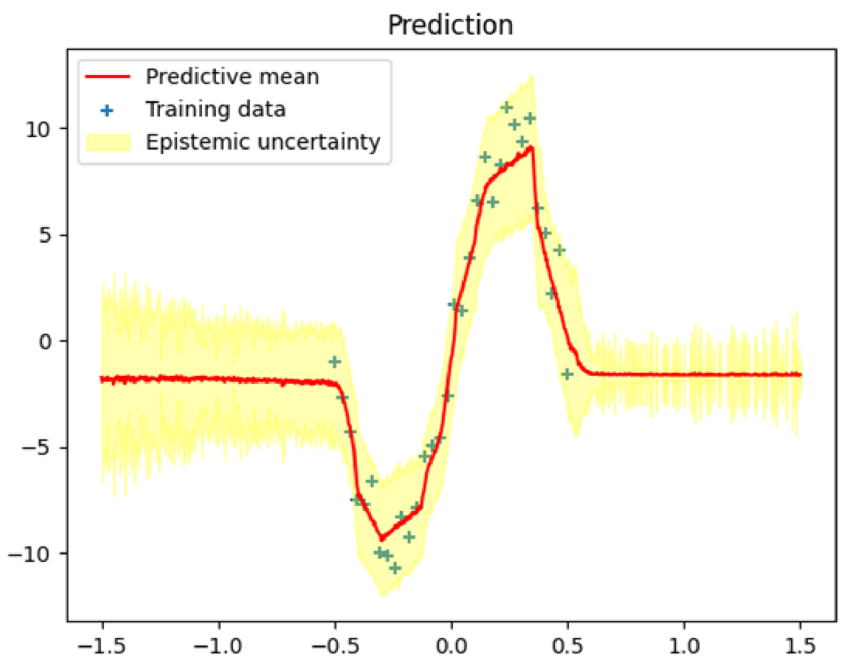
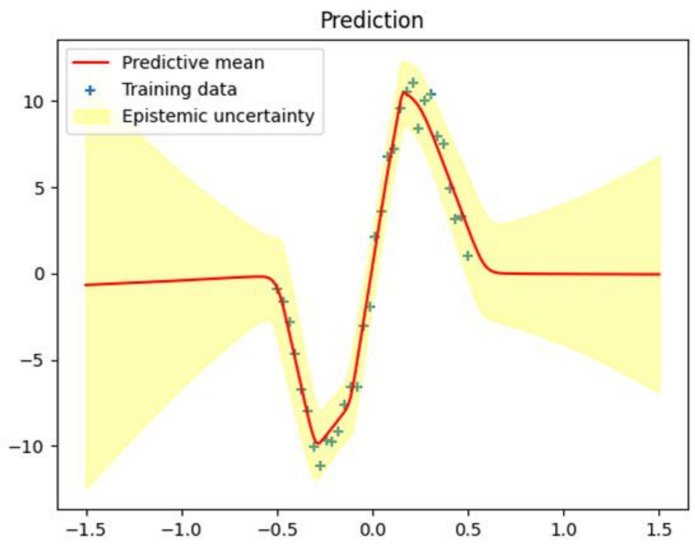
Task 1



Above is an example of a typical output figure from running Monte Carlo Dropout. We can see that the predictive mean fits rather well to the training data in the inner range where the training data is sampled (-0.5 to 0.5), with a low uncertainty particularly in the steepest parts of the slope. The latter makes sense because in these parts, the variation in y value between x values close to each other is at its highest, while at the top and bottom parts of the wave, the gradient is close to zero, meaning more of the variation in y values comes only from the noise from when the training data is generated.

The mean is only a flat line close to zero in the outer ranges where the model hasn't observed any data during training (-1.5 to -0.5 and 0.5 to 1.5). The uncertainty is also significantly higher here than in the inner range. Both of these results seem sensible; outside of the range of the training data, the model hasn't learned any pattern, so the predicted y value for some x value in the outer ranges tends to be random without any particular bias, as witnessed by the mean being flat and close to zero. The variance in these ranges also being higher makes sense as the lack of training data here makes the outputs of the model unstable.

Task 2



Above is an example of a typical output figure from running Variational Inference. When compared to MC-dropout, we notice that the uncertainty tends to be lower in the inner range where the training data was generated, and that the line of the predictive mean is much smoother than in MC-dropout. Interestingly, the variance increases much more rapidly here as the x values go further away from the inner range, compared to above.

Task 3
The re-parametrization trick is performed in lines 59 and 62, to the kernel and bias weights respectively. This is where the weights w are first set to the mu values from their respective variational distributions as seen in equation 2. Then, some noise epsilon_s is sampled from the N(0, 1) distribution, representing the noise added in the re-parametrization trick. Finally, this noise is multiplied by sigma and added to the weight now labeled w_s, where sigma is the standard deviation corresponding to that specific weight. The mu and sigma values here form the lambda = (mu, sigma) parameter set for specific weights w. After the trick, we now have w_s = w(lambda, epsilon_s).

Task 4
The first and third loss components in equation 3 are added to the computational graph in lines 64 and 65. The weights and their lambda=(mu, sigma) values are sent to the var_loss function (lines 69-71), where the first term is calculated using variational_dist, and the third term is calculated using the log_prior_prob function (lines 73-77).

Task 5
The second loss component is added to the computation graph in line 40 or 42 of vi_train_test.py, where the model is compiled with the neg_log_likelihood function from common.py (line 30-32) which calculates the second loss component.

Task 6
As mentioned in the lecture, an advantage of MCMC is that it is asymptotically exact, meaning its estimation is guaranteed to converge towards being exact if one runs it long enough. This is not guaranteed in the case of variational inference. However, it is also computationally expensive, and variational inference can be faster.