Advanced topics on machine learning
Assignment 2: Bayesian Neural Network
UNIVERSIDAD TECNOLÓGICA DE PEREIRA
Mauricio A. Álvarez Phd
TA: Cristian D. Guarnizo PhD and Hernan F. García PhD (c)

1. A Bayesian neural network is a neural network with a prior distribution on its weights [4, 2].

   Consider a data set $\{(\mathbf{x}_n, y_n)\}$, where each data point comprises of features $\mathbf{x}_n \in \mathbb{R}^D$ and output $y_n \in \mathbb{R}$. Define the likelihood for each data point as

$$p(y_n \mid \mathbf{w}, \mathbf{x}_n, \sigma^2) = \text{Normal}(y_n \mid \text{NN}(\mathbf{x}_n \; ; \; \mathbf{w}), \sigma^2), \tag{1}$$

   where NN is a neural network whose weights and biases form the latent variables $\mathbf{w}$. Assume σ^2 is a known variance. Define the prior on the weights and biases $\mathbf{w}$ to be the standard normal

$$p(\mathbf{w}) = \text{Normal}(\mathbf{w} \mid \mathbf{0}, \mathbf{I}). \tag{2}$$

   Let's build the model in Pytorch. We seek to analyse two Bayesian neural network approaches:

   - **Bayes by Backprop (BBP)**: Example with two hidden and one output layer [1]. Example code available at `https://github.com/JavierAntoran/Bayesian-Neural-Networks/blob/master/notebooks/regression/bbp_homo.ipynb`

```
class BayesLinear_Normalq(nn.Module):
def __init__(self, input_dim, output_dim, prior):
super(BayesLinear_Normalq, self).__init__()
self.input_dim = input_dim
self.output_dim = output_dim
self.prior = prior

scale = (2/self.input_dim)**0.5
rho_init = np.log(np.exp((2/self.input_dim)**0.5) - 1)
self.weight_mus = nn.Parameter(torch.Tensor(self.input_dim,
self.output_dim).uniform_(-0.05, 0.05))
self.weight_rhos = nn.Parameter(torch.Tensor(self.input_dim,
self.output_dim).uniform_(-2, -1))

self.bias_mus = nn.Parameter(torch.Tensor(self.output_dim).uniform_(-0.05, 0.05))
self.bias_rhos = nn.Parameter(torch.Tensor(self.output_dim).uniform_(-2, -1))


class BBP_Homoscedastic_Model(nn.Module):
def __init__(self, input_dim, output_dim, no_units, init_log_noise):
```

```
super(BBP_Homoscedastic_Model, self).__init__()

self.input_dim = input_dim
self.output_dim = output_dim

# network with two hidden and one output layer
self.layer1 = BayesLinear_Normalq(input_dim, no_units, gaussian(0, 1))
self.layer2 = BayesLinear_Normalq(no_units, output_dim, gaussian(0, 1))

# activation to be used between hidden layers
self.activation = nn.ReLU(inplace = True)
self.log_noise = nn.Parameter(torch.cuda.FloatTensor([init_log_noise]))
```

- **MC Dropout**: Dropout as a Bayesian Approximation (2 layer network)[3]. Example code available at `https://github.com/JavierAntoran/Bayesian-Neural-Networks/blob/master/notebooks/regression/mc_dropout_homo.ipynb`.

```
class MC_Dropout_Model(nn.Module):
def __init__(self, input_dim, output_dim, num_units, drop_prob):
super(MC_Dropout_Model, self).__init__()

self.input_dim = input_dim
self.output_dim = output_dim
self.drop_prob = drop_prob

# network with two hidden and one output layer
self.layer1 = nn.Linear(input_dim, num_units)
self.layer2 = nn.Linear(num_units, 2*output_dim)

self.activation = nn.ReLU(inplace = True)


def forward(self, x):

x = x.view(-1, self.input_dim)

x = self.layer1(x)
x = self.activation(x)

x = F.dropout(x, p=self.drop_prob, training=True)

x = self.layer2(x)

return x
```

You can visit the Pytorch implementations for the above inference methods: `https://github.com/JavierAntoran/Bayesian-Neural-Networks`

- For this problem you need to set different experiments for the posterior parameters to evaluate the posterior draws after the inference process.
- You should think carefully about how to set the hyperparameters in the inference process.
- Try to set 3 different networks architectures and report accuracy performances based on MSE and R^2 coefficients.
- Finally, test your implementation on real-world datasets for regression problems such as:

Real-world Datasets

Here are some possible datasets you could play with you are curious. Each one is a 'real' dataset with a univariate input x and univariate output y, so you could easily fit a BNN to the dataset using your existing code.

- **Carbon dioxide over time** We measure C02 over time at Mauna Loa observatory. The input x is the time since 1958, the output y is the C02 concentration in parts-per-million.
  - **Dataset description:** `http://scrippsco2.ucsd.edu/data/atmospheric_co2/primary_mlo_co2_record`
  - **Dataset file (.csv)** : `http://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/in_situ_co2/weekly/weekly_in_situ_co2_mlo.csv`
  - **Key question:** What values should we forecast in year 2020?
- **Revenue of Harry Potter films over time.** At each week (x) since the film's release, we have the total revenue in dollars y.
  - **Dataset description:** `http://users.stat.ufl.edu/~winner/data/harrypotter.txt`
  - **Dataset file (.csv):** `http://users.stat.ufl.edu/~winner/data/harrypotter.csv`
  - Key question: Given data from weeks 1-15, how accurate can we predict revenue in week 16 or 17? How much worse would we do if we only had weeks 1-10?

References

[1] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1613–1622. JMLR.org, 2015.

[2] Yarin Gal. *Uncertainty in Deep Learning.* PhD thesis, University of Cambridge, 2016.

[3] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR.

[4] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996.