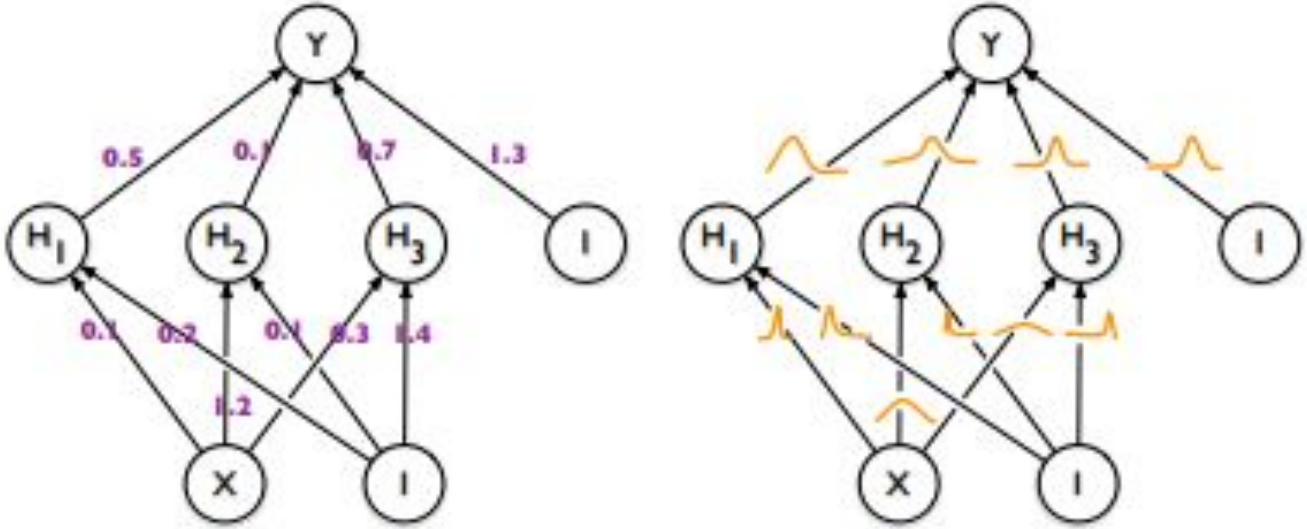
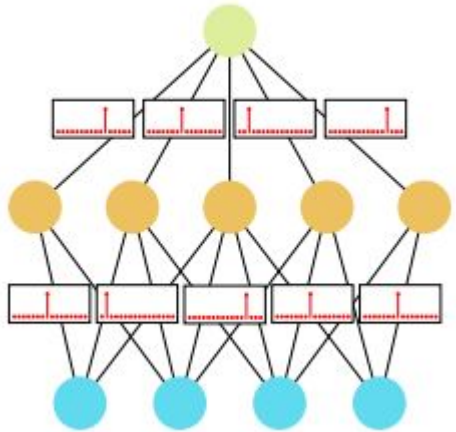
Bayesian Neural Network

Present by Linjun Huang

Neural Networks

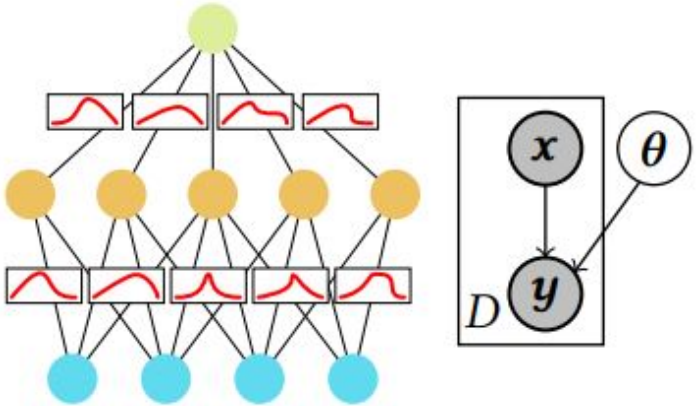


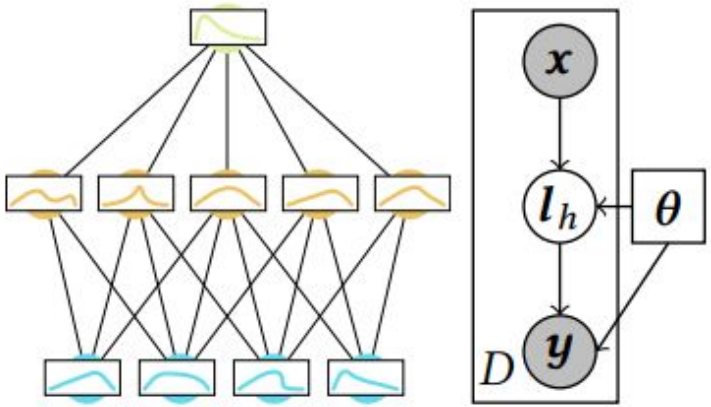
Regular Neural Network



$$
\begin{aligned}
\boldsymbol{l}_0 &= \boldsymbol{x}, \\
\boldsymbol{l}_i &= nl_i(\boldsymbol{W}_i \boldsymbol{l}_{i-1} + \boldsymbol{b}_i) \quad \forall i \in [1, n], \\
\boldsymbol{y} &= \boldsymbol{l}_n.
\end{aligned}
$$

Bayesian Neural Network

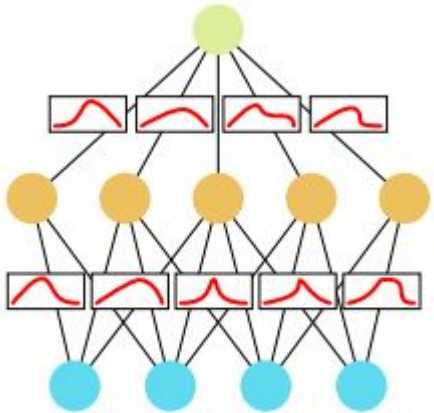




$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta}),$$
$$\boldsymbol{y} = NN_{\boldsymbol{\theta}}(\boldsymbol{x}) + \boldsymbol{\epsilon}$$

$$p(\boldsymbol{\theta}|D) = \frac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}'} \propto p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}).$$

Define BNN from a PGM



$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \Sigma),$$
$$\boldsymbol{y} \sim p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(NN_{\boldsymbol{\theta}}(\boldsymbol{x}), \Sigma).$$

$$p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta}) = \prod_{(\boldsymbol{x},\boldsymbol{y})\in D} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}).$$

$$\boldsymbol{l}_0 = \boldsymbol{x},$$
$$\boldsymbol{l}_i \sim p(\boldsymbol{l}_i|\boldsymbol{l}_{i-1}) = nl_i(\mathcal{N}(\boldsymbol{W}_i\boldsymbol{l}_{i-1} + \boldsymbol{b}_i, \Sigma)) \quad \forall i \in [1, n],$$
$$\boldsymbol{y} = \boldsymbol{l}_n.$$

$$p(D_{\boldsymbol{y}}, \boldsymbol{l}_{[1,n-1]}|D_{\boldsymbol{x}}) = \prod_{(\boldsymbol{l}_0,\boldsymbol{l}_n)\in D} \left(\prod_{i=1}^{n} p(\boldsymbol{l}_i|\boldsymbol{l}_{i-1}) \right).$$

$$\boldsymbol{W} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{W}}, \Sigma_{\boldsymbol{W}}),$$
$$\boldsymbol{b} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{b}}, \Sigma_{\boldsymbol{b}}),$$
$$\boldsymbol{l} = nl(\boldsymbol{W}\boldsymbol{l}_{-1} + \boldsymbol{b})$$

Optimization on Neural Networks

Does not place a prior upon weights w

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

Place a prior upon weights w

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

Bayesian Estimation

Posterior Distribution $$P(\mathbf{w}|\mathcal{D}) = \frac{P(\mathbf{w}, \mathcal{D})}{P(\mathcal{D})} = \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{P(\mathcal{D})}$$

Predictive Distribution $$P(\hat{\mathbf{y}}|\hat{\mathbf{x}}) = \mathbb{E}_{P(\mathbf{w}|\mathcal{D})}[P(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w})]$$

Optimization via Kullback-Leibler divergence

Use a distribution $q(\mathbf{w}|\theta)$ to approximate the posterior distribution $P(\mathbf{w}|\mathcal{D})$

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

Denote objective function: $\mathcal{F}(\mathcal{D}, \theta) = D_{\mathrm{KL}}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})]$

Prior Distribution

Scale mixture gaussian prior:

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

gaussian prior:

$$P(\mathbf{w}) = \prod_j \mathcal{N}\left(\mathbf{w}_j | 0, \sigma^2\right)$$

Reparametrize

Reparameterize process:

$$z \sim \mathcal{N}(\mu, \sigma^2)$$

$$z = \sigma\epsilon + \mu$$

$$\epsilon \sim \mathcal{N}(0, 1)$$

The distribution of ϵ satisfies $q(\epsilon)d\epsilon = q(\mathbf{w}|\theta)d\mathbf{w}$

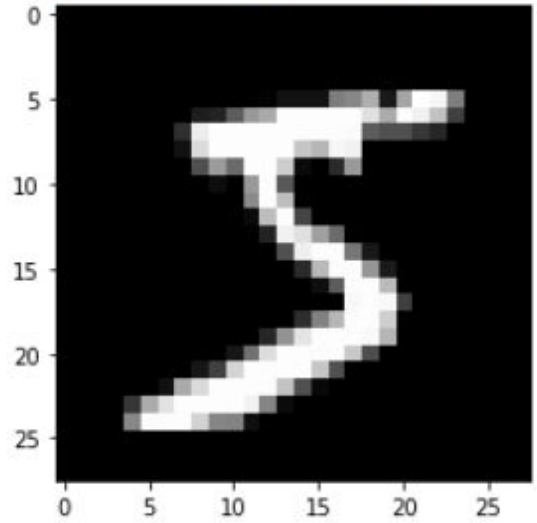
Unbiased Monte Carlo gradients

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathbb{E}_{q(\mathbf{w}|\theta)}[f(\mathbf{w},\theta)] &= \frac{\partial}{\partial\theta}\int f(\mathbf{w},\theta)q(\mathbf{w}|\theta)\mathrm{d}\mathbf{w} \\ &= \frac{\partial}{\partial\theta}\int f(\mathbf{w},\theta)q(\epsilon)\mathrm{d}\epsilon \\ &= \mathbb{E}_{q(\epsilon)}\left[\frac{\partial f(\mathbf{w},\theta)}{\partial\mathbf{w}}\frac{\partial\mathbf{w}}{\partial\theta}+\frac{\partial f(\mathbf{w},\theta)}{\partial\theta}\right]\end{aligned}$$

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

Numerical Results

```
Loading MNIST dataset
X_train.shape = (60000, 28, 28, 1)
y_train.shape = (60000, 10)
X_test.shape = (10000, 28, 28, 1)
y_test.shape = (10000, 10)
<matplotlib.image.AxesImage at 0x7f9d0d465cc0>
```



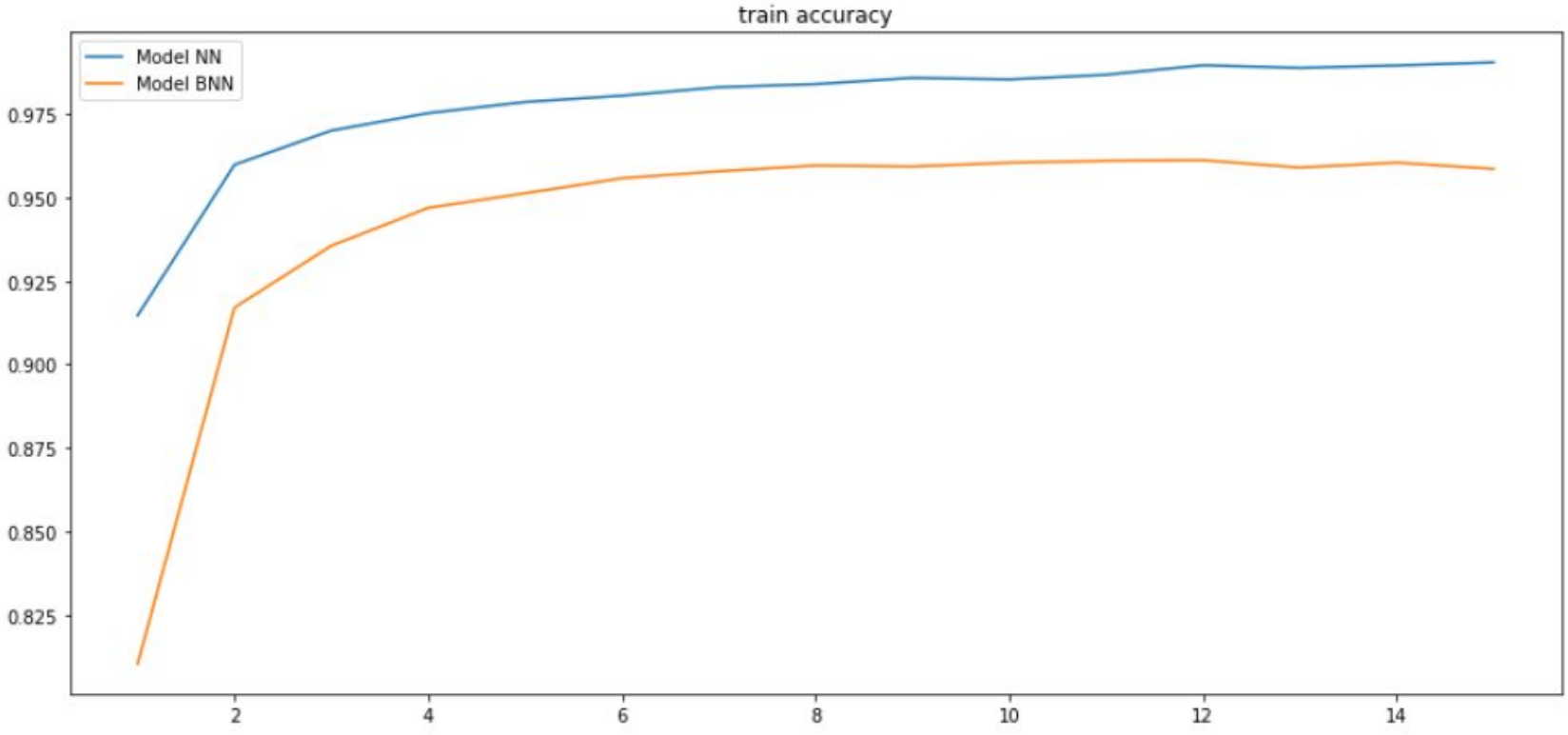
Numerical Results

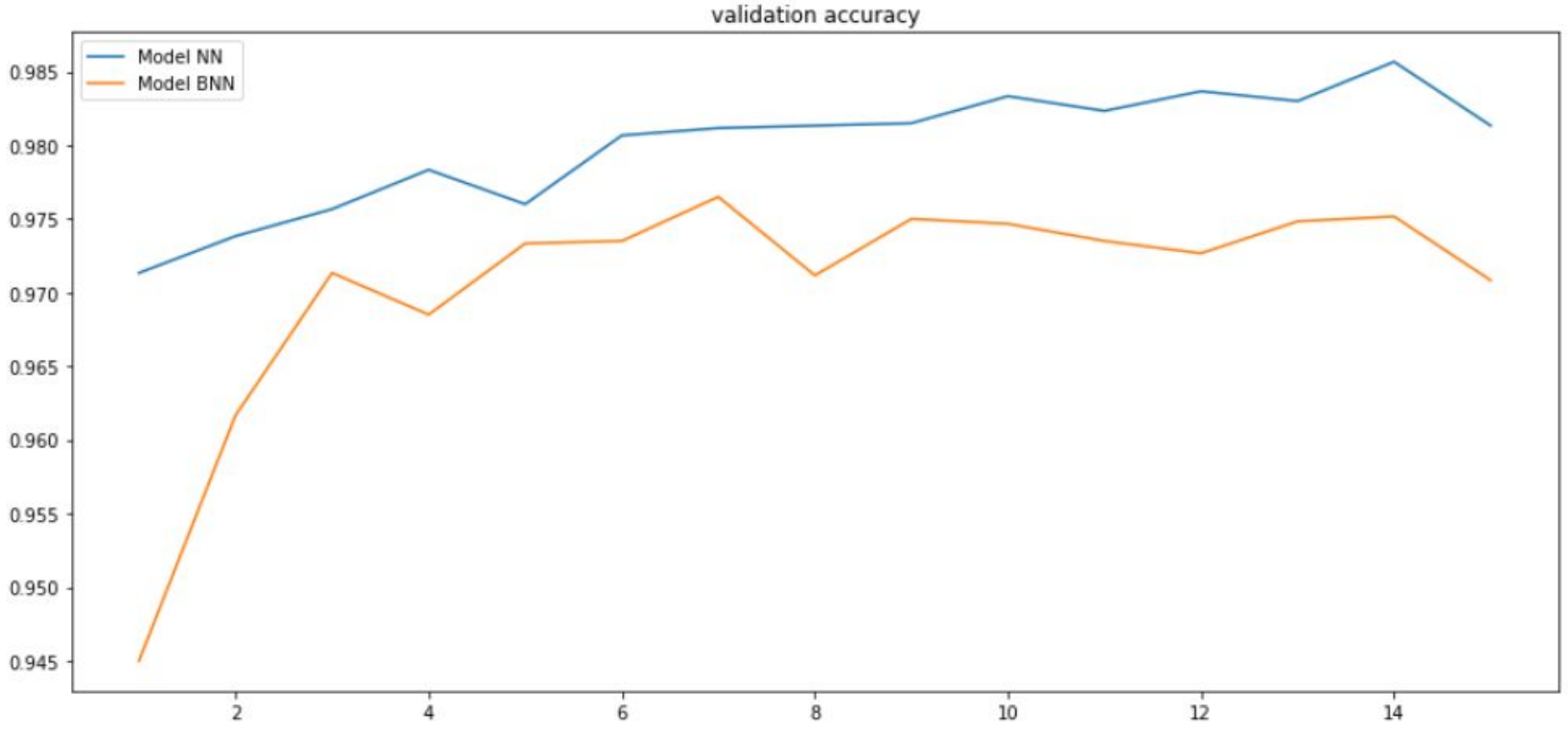
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 28, 28, 1)]	0
flatten (Flatten)	(None, 784)	0
dense_flipout (DenseFlipout)	(None, 512)	803329
batch_normalization (BatchNo	(None, 512)	2048
dropout (Dropout)	(None, 512)	0
dense_flipout_1 (DenseFlipou	(None, 256)	262401
batch_normalization_1 (Batch	(None, 256)	1024
dropout_1 (Dropout)	(None, 256)	0
dense_flipout_2 (DenseFlipou	(None, 128)	65665
dropout_2 (Dropout)	(None, 128)	0
dense_flipout_3 (DenseFlipou	(None, 10)	2571

Total params: 1,137,038
Trainable params: 1,135,498
Non-trainable params: 1,540

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 28, 28, 1)]	0
flatten_1 (Flatten)	(None, 784)	0
dense (Dense)	(None, 512)	401920
batch_normalization_2 (Batch	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
batch_normalization_3 (Batch	(None, 256)	1024
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1290

Total params: 570,506
Trainable params: 568,970
Non-trainable params: 1,536


train accuracy
Model NN
Model BNN
0.975
0.950
0.925
0.900
0.875
0.850
0.825
2
4
6
8
10
12
14


validation accuracy
Model NN
Model BNN
0.985
0.980
0.975
0.970
0.965
0.960
0.955
0.950
0.945
2
4
6
8
10
12
14

Test accuracy

```
preds_nn = nn_model.evaluate(X_test, y_test)
print ("Loss NN = {}" + str(preds_nn[0]))
print ("Test Accuracy NN = " + str(preds_nn[1]))
```

```
313/313 [==============================] - 1s 2ms/step - loss:
Loss NN = {}0.07821856439113617
Test Accuracy NN = 0.9800000190734863
```

```
preds_bnn = bnn_model.evaluate(X_test, y_test)
print ("Loss BNN = {}" + str(preds_bnn[0]))
print ("Test Accuracy BNN = " + str(preds_bnn[1]))
```

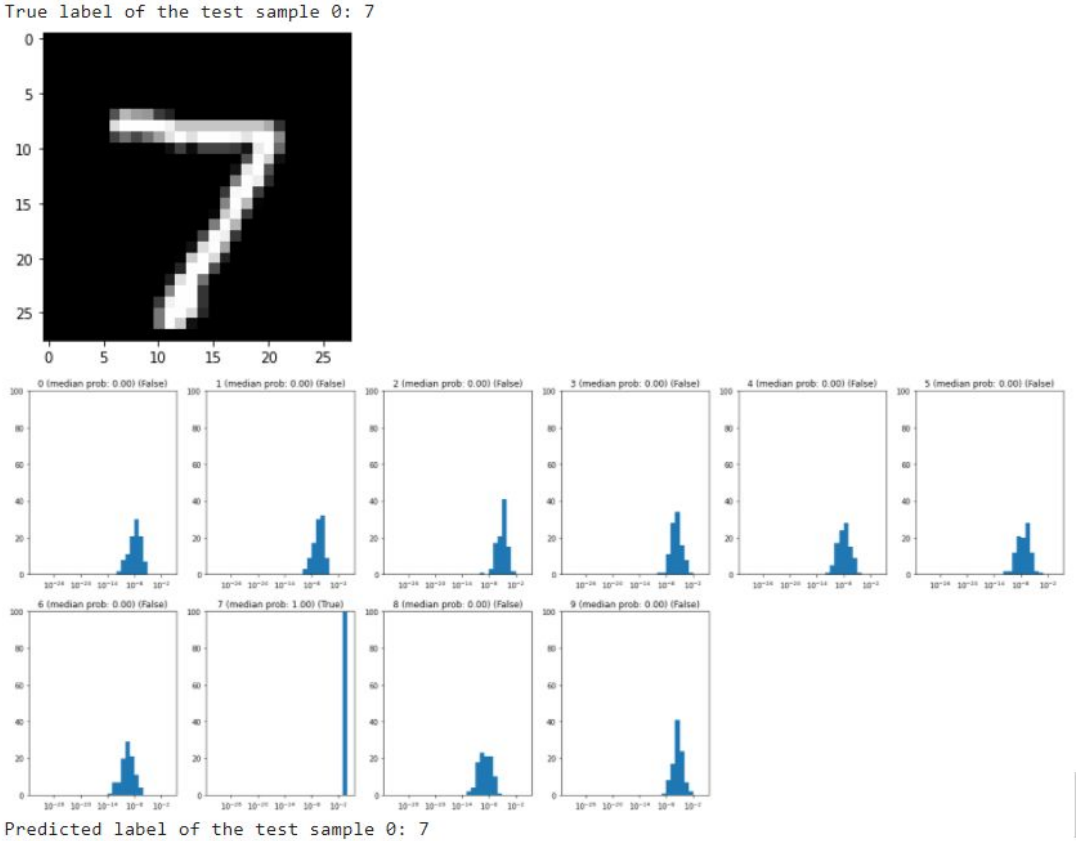
```
313/313 [==============================] - 1s 4ms/step - loss:
Loss BNN = {}8.128716468811035
Test Accuracy BNN = 0.9689000248908997
```

Quantify the uncertainty in predictions for bnn

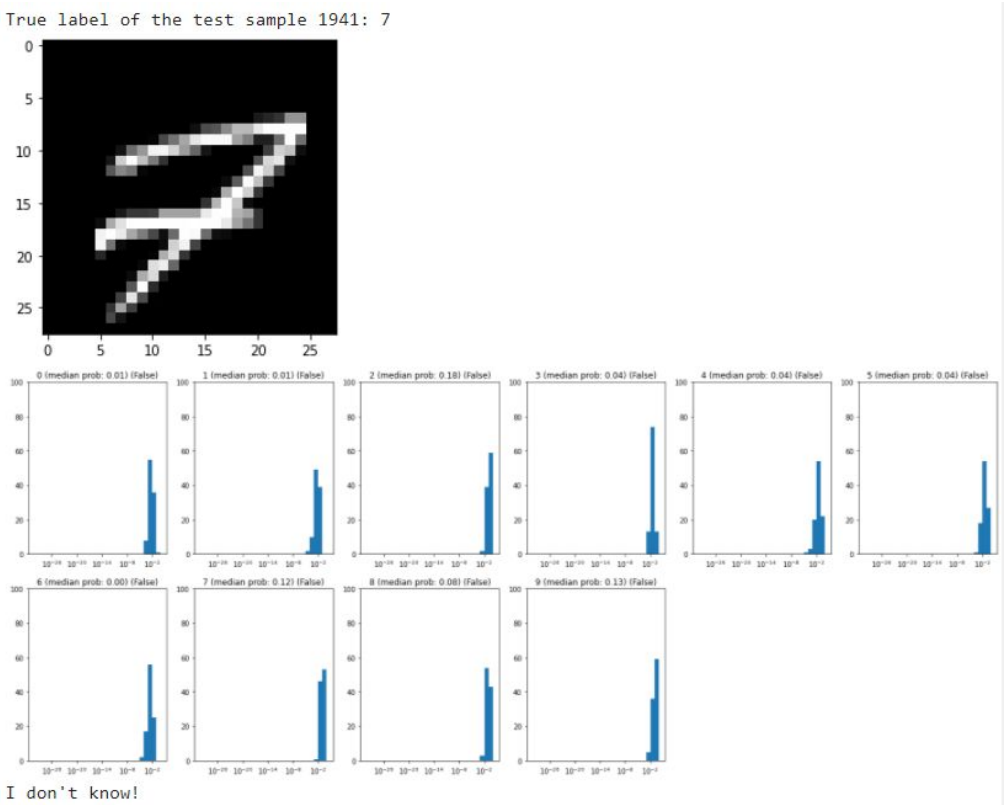
```
n_mc_run = 100
med_prob_thres = 0.2


y_pred_logits_list = [bnn_model.predict(X_test) for _ in range(n_mc_run)]   # a list of predicted logits
y_pred_prob_all = np.concatenate([softmax(y, axis=-1)[:, :, np.newaxis] for y in y_pred_logits_list], axis=-1)
y_pred = [[int(np.median(y) >= med_prob_thres) for y in y_pred_prob] for y_pred_prob in y_pred_prob_all]
y_pred = np.array(y_pred)

idx_valid = [any(y) for y in y_pred]
print('Number of recognizable samples:', sum(idx_valid))

idx_invalid = [not any(y) for y in y_pred]
print('Unrecognizable samples:', np.where(idx_invalid)[0])

print('Test accuracy on MNIST (recognizable samples):',
          sum(np.equal(np.argmax(y_test[idx_valid], axis=-1), np.argmax(y_pred[idx_valid], axis=-1))) / len(y_test[idx_valid]))

print('Test accuracy on MNIST (unrecognizable samples):',
          sum(np.equal(np.argmax(y_test[idx_invalid], axis=-1), np.argmax(y_pred[idx_invalid], axis=-1))) / len(y_test[idx_invalid]))
```

```
Number of recognizable samples: 9996
Unrecognizable samples: [1941 2266 6625 9634]
Test accuracy on MNIST (recognizable samples): 0.9746898759503801
Test accuracy on MNIST (unrecognizable samples): 0.25
```

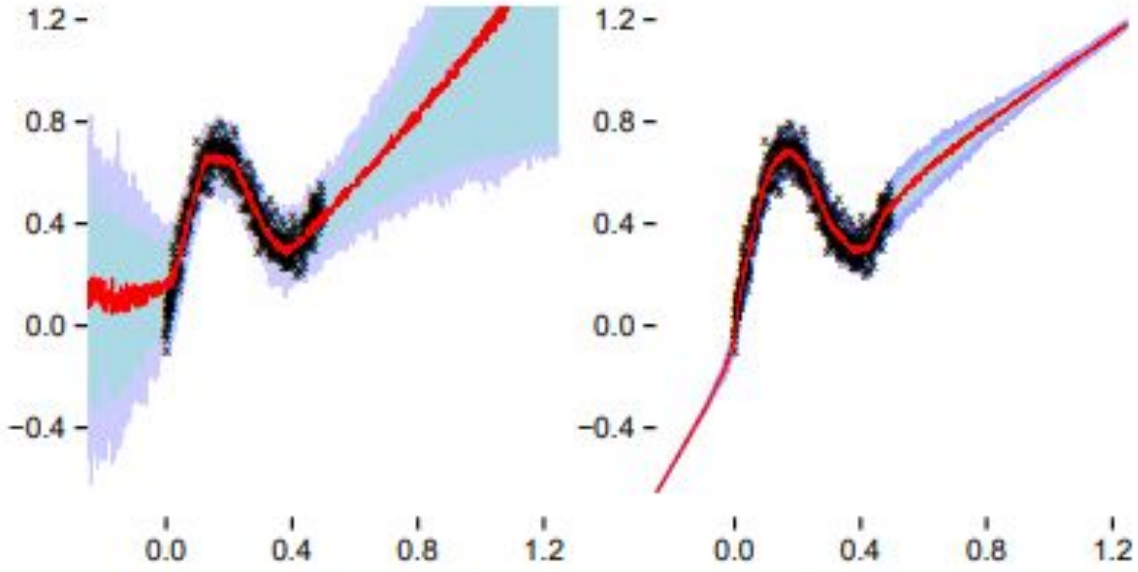
A recognizable example



Unrecognizable example



Example in paper



Reference

[1] Laurent Valentin Jospin, etc, Hands-on Bayesian Neural Network - a Tutorial for Deep Learning Users.

[2] Charles Blundell, etc, Weight Uncertainty in Neural Networks.