train-BLL

February 20, 2025

1 Train a Bayesian Layer Model for MNIST classification

In this notebook, we will train an MNIST convolutional classifier with a Bayesian last layer. First, we train a determinstic classifier to obtain a MAP estimate of the weights. Then, we replace the last layer of this model with a Bayesian last layer and train the layer using Stochastic Gradient Hamiltonian Monte Carlo (we keep the backbone fixed).

1.1 Setup

Import libraries

[1]:
```
import importlib
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import os

import models.BLL
import models.mnist_classifier_BLL
import train
import sampler
```

Set the configuration

[59]:
```
# 1. Configuration
class Config:
    # Data
    batch_size = 512

    # Backbone training
    backbone_epochs = 12
    backbone_lr = 0.001
    backbone_patience = 5

    # BLL training
    bll_epochs = 100
    bll_lr = 0.5e-2
    bll_base_C = 0.05
    bll_gauss_sig = 0.1

    # SGHMC parameters
    burn_in = 15
    re_burn = 1e7
    sim_steps = 2       # Save model every N epochs
    N_saves = 300        # Maximum ensemble size
    resample_its = 10   # Resample momentum every N iterations
    resample_prior_its = 45
    nb_its_dev = 1      # Evaluate on validation set every epoch

    # Hardware
    device = 'cuda' if torch.cuda.is_available() \
    else 'mps' if torch.backends.mps.is_available() \
    else 'cpu'

cfg = Config()
```

Load the Datasets

```
[20]: # Load the MNIST dataset
transform = transforms.Compose([transforms.ToTensor()])
trainset = torchvision.datasets.MNIST(root='../data', train=True,␣
 ↪download=True, transform=transform)
testset = torchvision.datasets.MNIST(root='../data', train=False,␣
 ↪download=True, transform=transform)

# Split training set into train and validation
train_size = int(0.8 * len(trainset))
val_size = len(trainset) - train_size
trainset, valset = torch.utils.data.random_split(trainset, [train_size,␣
 ↪val_size])

trainloader = torch.utils.data.DataLoader(trainset, batch_size=cfg.batch_size,␣
 ↪shuffle=True, num_workers=2)
valloader = torch.utils.data.DataLoader(valset, batch_size=cfg.batch_size,␣
 ↪shuffle=False, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=cfg.batch_size,␣
 ↪shuffle=False, num_workers=2)
```

Create a models and results directory if it doesn’t exist

```
[4]: models_dir = '../model_saves'
results_dir = '../results'
os.makedirs(models_dir, exist_ok=True)
os.makedirs(results_dir, exist_ok=True)
```

1.2 Backbone

We'll start with the backbone training.

1.2.1 Backbone

Load model

```
[43]: from models.mnist_classifier_BLL import MNISTClassifierBLL
importlib.reload(models.mnist_classifier_BLL)


backbone = MNISTClassifierBLL(device=cfg.device)
print(f"Backbone will run on: {cfg.device}")
```

```
Backbone will run on: mps
```

Train model

```
[ ]: from train import train_backbone
importlib.reload(train)

cost_tr, cost_dev, err_tr, err_dev, best_err = train_backbone(
    net=backbone,
    name='mnist_backbone',
    batch_size=cfg.batch_size,
    nb_epochs=cfg.backbone_epochs,
    trainset=trainset,
    valset=valset,
    device=cfg.device,
    lr=cfg.backbone_lr,
    patience=cfg.backbone_patience,
    model_saves_dir=models_dir
)

print(f"\nBackbone training completed! Best validation error: {best_err:.4f}")
```

1.2.2 Load weights

```
[44]: backbone.load_weights(models_dir + '/mnist_backbone_models/
  ↪backbone_best_12_epochs.pt')
```

```
 [load_weights] Loaded backbone weights from
../model_saves/mnist_backbone_models/backbone_best_12_epochs.pt
```

```
/Users/conor/Documents/College terms/College/Thesis/Thesis_Code_Minimised/MyImpl
ementation/models/mnist_classifier_BLL.py:47: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value), which uses
```

```
the default pickle module implicitly. It is possible to construct malicious
pickle data which will execute arbitrary code during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  state_dict = torch.load(path, map_location=self.device)
```

1.3 Bayesian Last Layer - SGHMC

Next we'll train the Bayesian last layer on the penultimate layer representations of the backbone.

1.3.1 Load the model

[60]:
```python
from models.BLL import BayesianLastLayerCat
importlib.reload(models.BLL)

bll = BayesianLastLayerCat(
    backbone=backbone,
    input_dim=256,  # Matches backbone's encoder output
    output_dim=10,  # MNIST classes
    N_train=len(trainset),
    lr=cfg.bll_lr,
    base_C=cfg.bll_base_C,
    gauss_sig=cfg.bll_gauss_sig,
    device=cfg.device
)
```

1.3.2 Train the model

[61]:
```python
from train import train_BLL_classification
importlib.reload(train)

bll_cost_tr, bll_cost_dev, bll_err_tr, bll_err_dev = train_BLL_classification(
    net=bll,
    name='mnist_bll',
    batch_size=cfg.batch_size,
    nb_epochs=cfg.bll_epochs,
    trainset=trainset,
    valset=valset,
    device=cfg.device,
    burn_in=cfg.burn_in,
    sim_steps=cfg.sim_steps,
    resample_its=cfg.resample_its,
    resample_prior_its=cfg.resample_prior_its,
    re_burn=cfg.re_burn,
    nb_its_dev=cfg.nb_its_dev,
    model_saves_dir=models_dir,
    N_saves=cfg.N_saves
)
```

```
Network:


Train:
  init cost variables:
it 0/100, Jtr_pred = 0.001837, err = 0.233313,    time: 8.568322 seconds


 [update_lr] Learning rate: 0.004950  (epoch 0)
    Jdev = 0.257514, err = 0.017167


best validation error
it 1/100, Jtr_pred = 0.000236, err = 0.009042,    time: 10.637473 seconds


 [update_lr] Learning rate: 0.004901  (epoch 1)
    Jdev = 0.056591, err = 0.004417


best validation error
it 2/100, Jtr_pred = 0.000080, err = 0.004667,    time: 8.193529 seconds


 [update_lr] Learning rate: 0.004851  (epoch 2)
    Jdev = 0.029407, err = 0.003167


best validation error
it 3/100, Jtr_pred = 0.000051, err = 0.004042,    time: 7.480894 seconds


 [update_lr] Learning rate: 0.004803  (epoch 3)
    Jdev = 0.021131, err = 0.002583


best validation error
it 4/100, Jtr_pred = 0.000040, err = 0.003479,    time: 8.091476 seconds


 [update_lr] Learning rate: 0.004755  (epoch 4)
```

```
    Jdev = 0.018149, err = 0.002250


best validation error
it 5/100, Jtr_pred = 0.000036, err = 0.003229,    time: 7.083124 seconds


 [update_lr] Learning rate: 0.004707  (epoch 5)
    Jdev = 0.016126, err = 0.002000


best validation error
it 6/100, Jtr_pred = 0.000032, err = 0.003000,    time: 8.318410 seconds


 [update_lr] Learning rate: 0.004660  (epoch 6)
    Jdev = 0.014856, err = 0.002167


it 7/100, Jtr_pred = 0.000030, err = 0.003125,    time: 8.166310 seconds


 [update_lr] Learning rate: 0.004614  (epoch 7)
    Jdev = 0.013787, err = 0.002250


it 8/100, Jtr_pred = 0.000029, err = 0.003125,    time: 7.692828 seconds


 [update_lr] Learning rate: 0.004568  (epoch 8)
    Jdev = 0.012797, err = 0.002250


it 9/100, Jtr_pred = 0.000027, err = 0.002875,    time: 6.728104 seconds


 [update_lr] Learning rate: 0.004522  (epoch 9)
    Jdev = 0.012283, err = 0.002167


it 10/100, Jtr_pred = 0.000027, err = 0.002708,    time: 8.171623 seconds


 [update_lr] Learning rate: 0.004477  (epoch 10)
    Jdev = 0.012664, err = 0.002500


it 11/100, Jtr_pred = 0.000026, err = 0.002792,    time: 6.870549 seconds


 [update_lr] Learning rate: 0.004432  (epoch 11)
    Jdev = 0.011860, err = 0.001667
```

```
best validation error
it 12/100, Jtr_pred = 0.000026, err = 0.002750,    time: 9.096029 seconds

 [update_lr] Learning rate: 0.004388  (epoch 12)
    Jdev = 0.011969, err = 0.001750

it 13/100, Jtr_pred = 0.000026, err = 0.002771,    time: 8.043585 seconds

 [update_lr] Learning rate: 0.004344  (epoch 13)
    Jdev = 0.011713, err = 0.001750

it 14/100, Jtr_pred = 0.000025, err = 0.002750,    time: 10.918128 seconds

 [update_lr] Learning rate: 0.004300  (epoch 14)
    Jdev = 0.011671, err = 0.001917

it 15/100, Jtr_pred = 0.000025, err = 0.002771,    time: 9.815231 seconds

 [update_lr] Learning rate: 0.004257  (epoch 15)
    Jdev = 0.011241, err = 0.001750

it 16/100, Jtr_pred = 0.000025, err = 0.002646,    time: 6.037055 seconds

 [update_lr] Learning rate: 0.004215  (epoch 16)
 [save_sampled_net] Ensemble size = 1
    Jdev = 0.011483, err = 0.001833

it 17/100, Jtr_pred = 0.000025, err = 0.002750,    time: 6.087959 seconds

 [update_lr] Learning rate: 0.004173  (epoch 17)
    Jdev = 0.011387, err = 0.001833

it 18/100, Jtr_pred = 0.000024, err = 0.002833,    time: 10.035904 seconds

 [update_lr] Learning rate: 0.004131  (epoch 18)
 [save_sampled_net] Ensemble size = 2
    Jdev = 0.011134, err = 0.001917
```

```
it 19/100, Jtr_pred = 0.000024, err = 0.002771,    time: 7.539050 seconds


 [update_lr] Learning rate: 0.004090  (epoch 19)
    Jdev = 0.010632, err = 0.001833


it 20/100, Jtr_pred = 0.000024, err = 0.002750,    time: 6.865726 seconds


 [update_lr] Learning rate: 0.004049  (epoch 20)
 [save_sampled_net] Ensemble size = 3
    Jdev = 0.010812, err = 0.001583


best validation error
it 21/100, Jtr_pred = 0.000023, err = 0.002604,    time: 8.444082 seconds


 [update_lr] Learning rate: 0.004008  (epoch 21)
    Jdev = 0.010896, err = 0.001500


best validation error
it 22/100, Jtr_pred = 0.000023, err = 0.002542,    time: 9.003900 seconds


 [update_lr] Learning rate: 0.003968  (epoch 22)
 [save_sampled_net] Ensemble size = 4
    Jdev = 0.010947, err = 0.001667


it 23/100, Jtr_pred = 0.000023, err = 0.002563,    time: 8.679716 seconds


 [update_lr] Learning rate: 0.003928  (epoch 23)
    Jdev = 0.010618, err = 0.001750


it 24/100, Jtr_pred = 0.000023, err = 0.002771,    time: 7.443284 seconds


 [update_lr] Learning rate: 0.003889  (epoch 24)
 [save_sampled_net] Ensemble size = 5
    Jdev = 0.010504, err = 0.001750


it 25/100, Jtr_pred = 0.000023, err = 0.002792,    time: 11.886833 seconds


 [update_lr] Learning rate: 0.003850  (epoch 25)
    Jdev = 0.010748, err = 0.001833
```

```
it 26/100, Jtr_pred = 0.000023, err = 0.002750,    time: 9.542504 seconds

 [update_lr] Learning rate: 0.003812  (epoch 26)
 [save_sampled_net] Ensemble size = 6
    Jdev = 0.010758, err = 0.002000

it 27/100, Jtr_pred = 0.000023, err = 0.002667,    time: 8.764845 seconds

 [update_lr] Learning rate: 0.003774  (epoch 27)
    Jdev = 0.010660, err = 0.002083

it 28/100, Jtr_pred = 0.000023, err = 0.002625,    time: 9.488486 seconds

 [update_lr] Learning rate: 0.003736  (epoch 28)
 [save_sampled_net] Ensemble size = 7
    Jdev = 0.010654, err = 0.002250

it 29/100, Jtr_pred = 0.000022, err = 0.002625,    time: 7.250892 seconds

 [update_lr] Learning rate: 0.003699  (epoch 29)
    Jdev = 0.010287, err = 0.002000

it 30/100, Jtr_pred = 0.000022, err = 0.002542,    time: 12.601461 seconds

 [update_lr] Learning rate: 0.003662  (epoch 30)
 [save_sampled_net] Ensemble size = 8
    Jdev = 0.010218, err = 0.001750

it 31/100, Jtr_pred = 0.000022, err = 0.002604,    time: 7.882566 seconds

 [update_lr] Learning rate: 0.003625  (epoch 31)
    Jdev = 0.010366, err = 0.001917

it 32/100, Jtr_pred = 0.000022, err = 0.002458,    time: 7.670981 seconds

 [update_lr] Learning rate: 0.003589  (epoch 32)
 [save_sampled_net] Ensemble size = 9
    Jdev = 0.010238, err = 0.001667
```

```
it 33/100, Jtr_pred = 0.000022, err = 0.002563,    time: 6.520203 seconds


 [update_lr] Learning rate: 0.003553  (epoch 33)
    Jdev = 0.010376, err = 0.002083


it 34/100, Jtr_pred = 0.000022, err = 0.002646,    time: 7.898595 seconds


 [update_lr] Learning rate: 0.003517  (epoch 34)
 [save_sampled_net] Ensemble size = 10
    Jdev = 0.010549, err = 0.002167


it 35/100, Jtr_pred = 0.000022, err = 0.002563,    time: 7.772721 seconds


 [update_lr] Learning rate: 0.003482  (epoch 35)
    Jdev = 0.010206, err = 0.001667


it 36/100, Jtr_pred = 0.000022, err = 0.002542,    time: 6.305813 seconds


 [update_lr] Learning rate: 0.003447  (epoch 36)
 [save_sampled_net] Ensemble size = 11
    Jdev = 0.010103, err = 0.001750


it 37/100, Jtr_pred = 0.000022, err = 0.002646,    time: 7.105284 seconds


 [update_lr] Learning rate: 0.003413  (epoch 37)
    Jdev = 0.010427, err = 0.002083


it 38/100, Jtr_pred = 0.000022, err = 0.002854,    time: 7.221690 seconds


 [update_lr] Learning rate: 0.003379  (epoch 38)
 [save_sampled_net] Ensemble size = 12
    Jdev = 0.010434, err = 0.002083


it 39/100, Jtr_pred = 0.000022, err = 0.002479,    time: 7.715948 seconds


 [update_lr] Learning rate: 0.003345  (epoch 39)
    Jdev = 0.010474, err = 0.002167
```

```
it 40/100, Jtr_pred = 0.000022, err = 0.002583,    time: 7.130812 seconds

 [update_lr] Learning rate: 0.003311  (epoch 40)
 [save_sampled_net] Ensemble size = 13
    Jdev = 0.010663, err = 0.002000

it 41/100, Jtr_pred = 0.000022, err = 0.002625,    time: 7.499086 seconds

 [update_lr] Learning rate: 0.003278  (epoch 41)
    Jdev = 0.010887, err = 0.002167

it 42/100, Jtr_pred = 0.000022, err = 0.002625,    time: 7.077938 seconds

 [update_lr] Learning rate: 0.003246  (epoch 42)
 [save_sampled_net] Ensemble size = 14
    Jdev = 0.010757, err = 0.002250

it 43/100, Jtr_pred = 0.000022, err = 0.002583,    time: 7.802394 seconds

 [update_lr] Learning rate: 0.003213  (epoch 43)
    Jdev = 0.010636, err = 0.002417

it 44/100, Jtr_pred = 0.000022, err = 0.002646,    time: 6.819732 seconds

 [update_lr] Learning rate: 0.003181  (epoch 44)
 [save_sampled_net] Ensemble size = 15
    Jdev = 0.010518, err = 0.002167

it 45/100, Jtr_pred = 0.000022, err = 0.002542,    time: 7.155013 seconds

 [update_lr] Learning rate: 0.003149  (epoch 45)
    Jdev = 0.010201, err = 0.002083

it 46/100, Jtr_pred = 0.000022, err = 0.002646,    time: 8.278322 seconds

 [update_lr] Learning rate: 0.003118  (epoch 46)
 [save_sampled_net] Ensemble size = 16
    Jdev = 0.009995, err = 0.002083
```

```
it 47/100, Jtr_pred = 0.000022, err = 0.002583,    time: 8.127163 seconds

 [update_lr] Learning rate: 0.003086  (epoch 47)
    Jdev = 0.010271, err = 0.002167

it 48/100, Jtr_pred = 0.000021, err = 0.002625,    time: 9.258728 seconds

 [update_lr] Learning rate: 0.003056  (epoch 48)
 [save_sampled_net] Ensemble size = 17
    Jdev = 0.009940, err = 0.002250

it 49/100, Jtr_pred = 0.000021, err = 0.002583,    time: 7.200599 seconds

 [update_lr] Learning rate: 0.003025  (epoch 49)
    Jdev = 0.009786, err = 0.002083

it 50/100, Jtr_pred = 0.000021, err = 0.002583,    time: 6.223587 seconds

 [update_lr] Learning rate: 0.002995  (epoch 50)
 [save_sampled_net] Ensemble size = 18
    Jdev = 0.009779, err = 0.001917

it 51/100, Jtr_pred = 0.000021, err = 0.002500,    time: 7.160804 seconds

 [update_lr] Learning rate: 0.002965  (epoch 51)
    Jdev = 0.009696, err = 0.001833

it 52/100, Jtr_pred = 0.000021, err = 0.002521,    time: 8.137869 seconds

 [update_lr] Learning rate: 0.002935  (epoch 52)
 [save_sampled_net] Ensemble size = 19
    Jdev = 0.009769, err = 0.001833

it 53/100, Jtr_pred = 0.000021, err = 0.002687,    time: 10.207228 seconds

 [update_lr] Learning rate: 0.002906  (epoch 53)
    Jdev = 0.010207, err = 0.002250
```

```
it 54/100, Jtr_pred = 0.000021, err = 0.002583,    time: 7.665964 seconds

 [update_lr] Learning rate: 0.002877  (epoch 54)
 [save_sampled_net] Ensemble size = 20
    Jdev = 0.009978, err = 0.002000

it 55/100, Jtr_pred = 0.000021, err = 0.002583,    time: 7.200393 seconds

 [update_lr] Learning rate: 0.002848  (epoch 55)
    Jdev = 0.010143, err = 0.002000

it 56/100, Jtr_pred = 0.000021, err = 0.002521,    time: 6.456891 seconds

 [update_lr] Learning rate: 0.002820  (epoch 56)
 [save_sampled_net] Ensemble size = 21
    Jdev = 0.009880, err = 0.001833

it 57/100, Jtr_pred = 0.000021, err = 0.002458,    time: 8.631031 seconds

 [update_lr] Learning rate: 0.002791  (epoch 57)
    Jdev = 0.009935, err = 0.001833

it 58/100, Jtr_pred = 0.000021, err = 0.002521,    time: 7.953333 seconds

 [update_lr] Learning rate: 0.002763  (epoch 58)
 [save_sampled_net] Ensemble size = 22
    Jdev = 0.010046, err = 0.002083

it 59/100, Jtr_pred = 0.000021, err = 0.002521,    time: 7.009132 seconds

 [update_lr] Learning rate: 0.002736  (epoch 59)
    Jdev = 0.010135, err = 0.002000

it 60/100, Jtr_pred = 0.000021, err = 0.002563,    time: 6.389312 seconds

 [update_lr] Learning rate: 0.002708  (epoch 60)
 [save_sampled_net] Ensemble size = 23
    Jdev = 0.010128, err = 0.002083
```

```
it 61/100, Jtr_pred = 0.000022, err = 0.002542,    time: 7.022122 seconds

 [update_lr] Learning rate: 0.002681  (epoch 61)
    Jdev = 0.010603, err = 0.002250

it 62/100, Jtr_pred = 0.000022, err = 0.002500,    time: 7.280527 seconds

 [update_lr] Learning rate: 0.002655  (epoch 62)
 [save_sampled_net] Ensemble size = 24
    Jdev = 0.010571, err = 0.002167

it 63/100, Jtr_pred = 0.000022, err = 0.002375,    time: 10.490560 seconds

 [update_lr] Learning rate: 0.002628  (epoch 63)
    Jdev = 0.010483, err = 0.002250

it 64/100, Jtr_pred = 0.000022, err = 0.002479,    time: 9.263145 seconds

 [update_lr] Learning rate: 0.002602  (epoch 64)
 [save_sampled_net] Ensemble size = 25
    Jdev = 0.010182, err = 0.002333

it 65/100, Jtr_pred = 0.000021, err = 0.002500,    time: 9.151130 seconds

 [update_lr] Learning rate: 0.002576  (epoch 65)
    Jdev = 0.010035, err = 0.002000

it 66/100, Jtr_pred = 0.000021, err = 0.002521,    time: 7.058984 seconds

 [update_lr] Learning rate: 0.002550  (epoch 66)
 [save_sampled_net] Ensemble size = 26
    Jdev = 0.010128, err = 0.001917

it 67/100, Jtr_pred = 0.000021, err = 0.002437,    time: 7.969620 seconds

 [update_lr] Learning rate: 0.002524  (epoch 67)
    Jdev = 0.010227, err = 0.002083
```

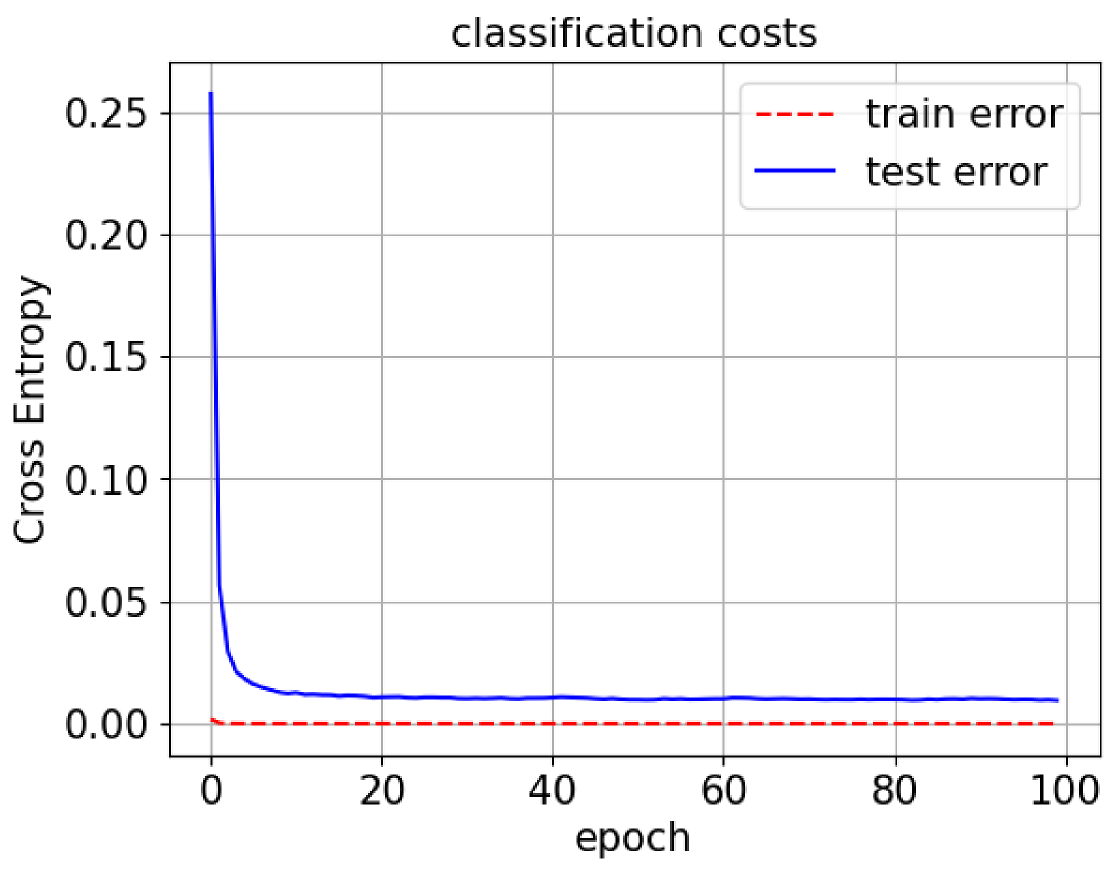
```
it 68/100, Jtr_pred = 0.000021, err = 0.002521,    time: 9.047228 seconds


 [update_lr] Learning rate: 0.002499  (epoch 68)
 [save_sampled_net] Ensemble size = 27
    Jdev = 0.010119, err = 0.002000


it 69/100, Jtr_pred = 0.000021, err = 0.002375,    time: 7.312837 seconds


 [update_lr] Learning rate: 0.002474  (epoch 69)
    Jdev = 0.009981, err = 0.001750


it 70/100, Jtr_pred = 0.000021, err = 0.002458,    time: 7.364047 seconds


 [update_lr] Learning rate: 0.002449  (epoch 70)
 [save_sampled_net] Ensemble size = 28
    Jdev = 0.010034, err = 0.001917


it 71/100, Jtr_pred = 0.000021, err = 0.002396,    time: 7.184331 seconds


 [update_lr] Learning rate: 0.002425  (epoch 71)
    Jdev = 0.009862, err = 0.001917


it 72/100, Jtr_pred = 0.000021, err = 0.002458,    time: 6.814541 seconds


 [update_lr] Learning rate: 0.002401  (epoch 72)
 [save_sampled_net] Ensemble size = 29
    Jdev = 0.009786, err = 0.001583


it 73/100, Jtr_pred = 0.000021, err = 0.002542,    time: 8.177941 seconds


 [update_lr] Learning rate: 0.002377  (epoch 73)
    Jdev = 0.009860, err = 0.001417


best validation error
it 74/100, Jtr_pred = 0.000021, err = 0.002417,    time: 7.047439 seconds


 [update_lr] Learning rate: 0.002353  (epoch 74)
 [save_sampled_net] Ensemble size = 30
    Jdev = 0.009812, err = 0.001417
```

```
it 75/100, Jtr_pred = 0.000021, err = 0.002437,    time: 9.925330 seconds

 [update_lr] Learning rate: 0.002329  (epoch 75)
    Jdev = 0.009800, err = 0.001583

it 76/100, Jtr_pred = 0.000021, err = 0.002375,    time: 7.381509 seconds

 [update_lr] Learning rate: 0.002306  (epoch 76)
 [save_sampled_net] Ensemble size = 31
    Jdev = 0.009941, err = 0.001750

it 77/100, Jtr_pred = 0.000021, err = 0.002458,    time: 6.341314 seconds

 [update_lr] Learning rate: 0.002283  (epoch 77)
    Jdev = 0.009814, err = 0.001750

it 78/100, Jtr_pred = 0.000021, err = 0.002333,    time: 8.338304 seconds

 [update_lr] Learning rate: 0.002260  (epoch 78)
 [save_sampled_net] Ensemble size = 32
    Jdev = 0.009905, err = 0.001833

it 79/100, Jtr_pred = 0.000021, err = 0.002333,    time: 7.562090 seconds

 [update_lr] Learning rate: 0.002238  (epoch 79)
    Jdev = 0.009915, err = 0.001750

it 80/100, Jtr_pred = 0.000021, err = 0.002396,    time: 8.392739 seconds

 [update_lr] Learning rate: 0.002215  (epoch 80)
 [save_sampled_net] Ensemble size = 33
    Jdev = 0.009888, err = 0.001750

it 81/100, Jtr_pred = 0.000021, err = 0.002354,    time: 6.346603 seconds

 [update_lr] Learning rate: 0.002193  (epoch 81)
    Jdev = 0.009806, err = 0.001583
```

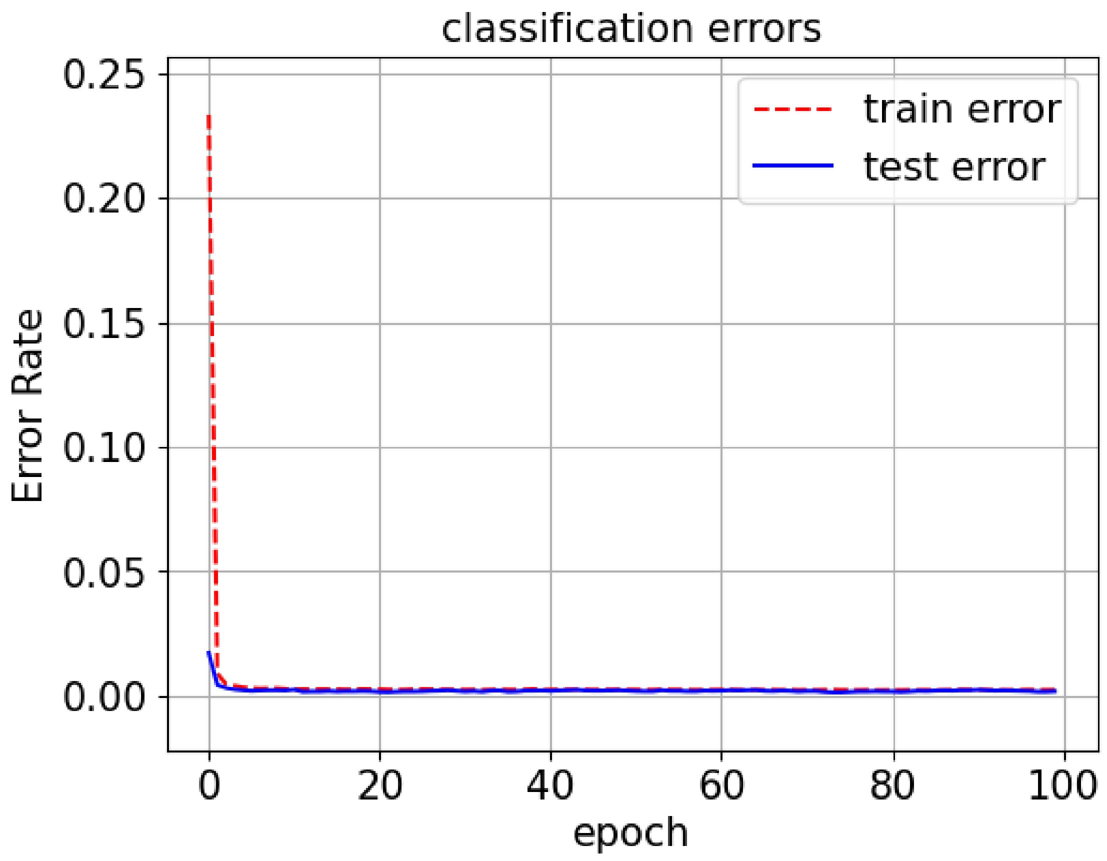
```
it 82/100, Jtr_pred = 0.000021, err = 0.002396,    time: 9.669320 seconds

 [update_lr] Learning rate: 0.002171  (epoch 82)
 [save_sampled_net] Ensemble size = 34
    Jdev = 0.009591, err = 0.001750

it 83/100, Jtr_pred = 0.000021, err = 0.002417,    time: 6.911044 seconds

 [update_lr] Learning rate: 0.002149  (epoch 83)
    Jdev = 0.009674, err = 0.002000

it 84/100, Jtr_pred = 0.000021, err = 0.002437,    time: 7.011087 seconds

 [update_lr] Learning rate: 0.002128  (epoch 84)
 [save_sampled_net] Ensemble size = 35
    Jdev = 0.009983, err = 0.001917

it 85/100, Jtr_pred = 0.000021, err = 0.002312,    time: 6.303456 seconds

 [update_lr] Learning rate: 0.002107  (epoch 85)
    Jdev = 0.009799, err = 0.002083

it 86/100, Jtr_pred = 0.000021, err = 0.002396,    time: 7.515413 seconds

 [update_lr] Learning rate: 0.002086  (epoch 86)
 [save_sampled_net] Ensemble size = 36
    Jdev = 0.010072, err = 0.002167

it 87/100, Jtr_pred = 0.000021, err = 0.002500,    time: 9.219244 seconds

 [update_lr] Learning rate: 0.002065  (epoch 87)
    Jdev = 0.010133, err = 0.002083

it 88/100, Jtr_pred = 0.000021, err = 0.002604,    time: 9.733516 seconds

 [update_lr] Learning rate: 0.002044  (epoch 88)
 [save_sampled_net] Ensemble size = 37
    Jdev = 0.009985, err = 0.002250
```

```
it 89/100, Jtr_pred = 0.000021, err = 0.002542,    time: 5.839701 seconds

 [update_lr] Learning rate: 0.002024  (epoch 89)
    Jdev = 0.010317, err = 0.002250

it 90/100, Jtr_pred = 0.000021, err = 0.002563,    time: 7.058231 seconds

 [update_lr] Learning rate: 0.002003  (epoch 90)
 [save_sampled_net] Ensemble size = 38
    Jdev = 0.010142, err = 0.002417

it 91/100, Jtr_pred = 0.000021, err = 0.002479,    time: 6.981652 seconds

 [update_lr] Learning rate: 0.001983  (epoch 91)
    Jdev = 0.010219, err = 0.002250

it 92/100, Jtr_pred = 0.000021, err = 0.002417,    time: 7.209603 seconds

 [update_lr] Learning rate: 0.001964  (epoch 92)
 [save_sampled_net] Ensemble size = 39
    Jdev = 0.010144, err = 0.002083

it 93/100, Jtr_pred = 0.000021, err = 0.002396,    time: 6.994222 seconds

 [update_lr] Learning rate: 0.001944  (epoch 93)
    Jdev = 0.009936, err = 0.002167

it 94/100, Jtr_pred = 0.000020, err = 0.002396,    time: 8.678849 seconds

 [update_lr] Learning rate: 0.001924  (epoch 94)
 [save_sampled_net] Ensemble size = 40
    Jdev = 0.009803, err = 0.002083

it 95/100, Jtr_pred = 0.000021, err = 0.002458,    time: 7.589891 seconds

 [update_lr] Learning rate: 0.001905  (epoch 95)
    Jdev = 0.009879, err = 0.002000
```

```
it 96/100, Jtr_pred = 0.000021, err = 0.002542,    time: 6.999519 seconds

 [update_lr] Learning rate: 0.001886  (epoch 96)
 [save_sampled_net] Ensemble size = 41
    Jdev = 0.009859, err = 0.001917

it 97/100, Jtr_pred = 0.000021, err = 0.002437,    time: 7.958044 seconds

 [update_lr] Learning rate: 0.001867  (epoch 97)
    Jdev = 0.009635, err = 0.001667

it 98/100, Jtr_pred = 0.000021, err = 0.002437,    time: 6.257087 seconds

 [update_lr] Learning rate: 0.001849  (epoch 98)
 [save_sampled_net] Ensemble size = 42
    Jdev = 0.009768, err = 0.001667

it 99/100, Jtr_pred = 0.000021, err = 0.002500,    time: 10.213789 seconds

 [update_lr] Learning rate: 0.001830  (epoch 99)
    Jdev = 0.009518, err = 0.001833

   average time: 13.680366 seconds

 [save_checkpoint] Saved model state to
../model_saves/mnist_bll_models/BLL_checkpoint_9163.pth
Saved final model to: ../model_saves/mnist_bll_models/BLL_checkpoint_9163.pth
<Figure size 640x480 with 0 Axes>
```


classification costs
train error
test error
Cross Entropy
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
epoch

```
<Figure size 640x480 with 0 Axes>
```



1.3.3 Load the model again

```
[54]: bll.load_checkpoint(models_dir + '/mnist_bll_models/BLL_checkpoint_200_epochs.
      ↪pth')
```

```
[load_checkpoint] Loaded checkpoint from
../model_saves/mnist_bll_models/BLL_checkpoint_200_epochs.pth
```

1.3.4 Test the model

```
[62]: bll.shuffle_ensemble()
```

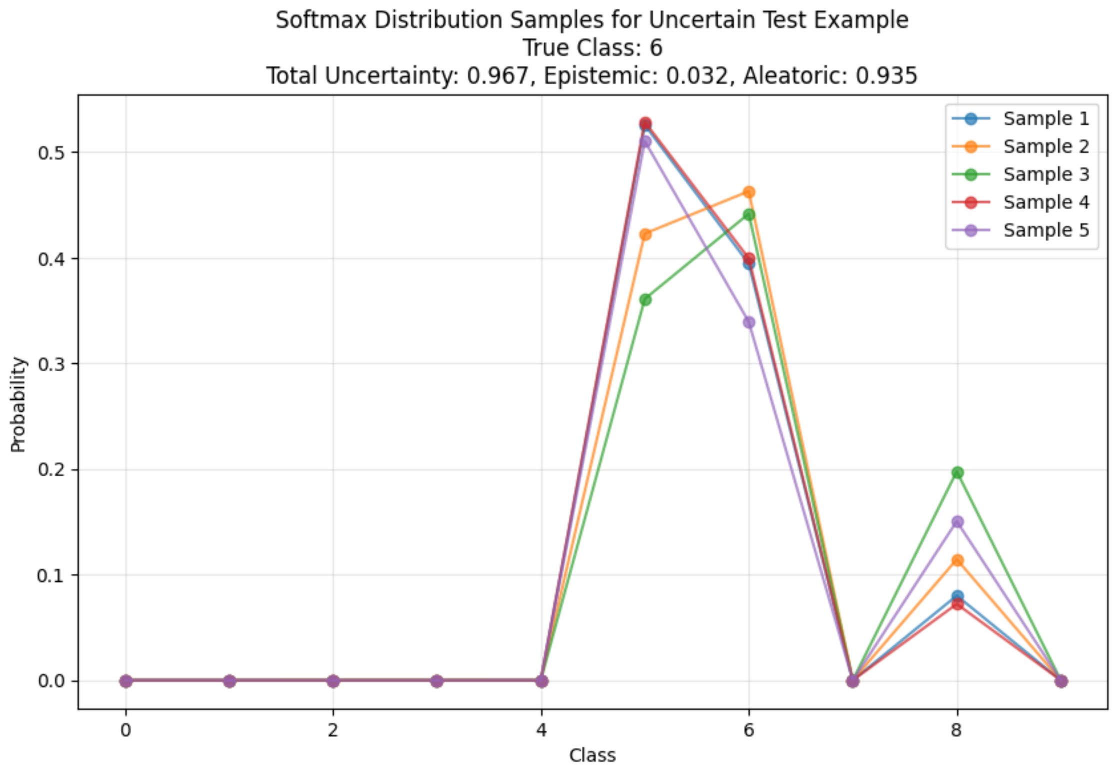
```
[shuffle_ensemble] Shuffled ensemble of 42 last layers
```

To check for Bayesian behaviour, we find an uncertain prediction and visualize multiple samples from the posterior.

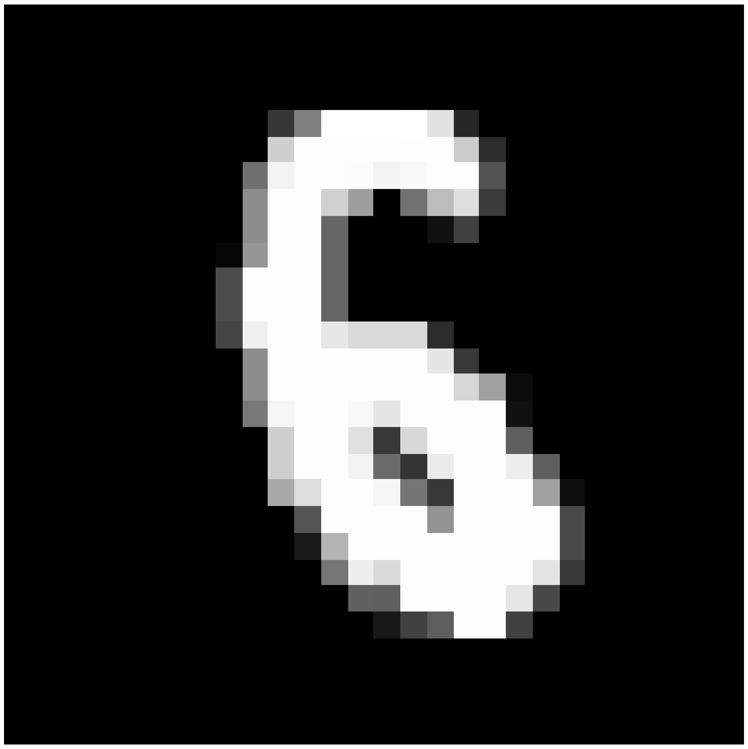
```
[49]: # Search for a test example with uncertain predictions
      test_iter = iter(testloader)
      found_uncertain = False
      max_tries = 100
```

```
num_samples = 5

try:
    while not found_uncertain:
        x, y = next(test_iter)
        # Look at each example in the batch
        for i in range(len(x)):
            x_single = x[i:i+1]
            y_single = y[i:i+1]

            # Get predictions and uncertainty measures
            mean_probs, uncertainties = bll.predict_with_uncertainty(x_single)
            probs = bll.sample_predict(x_single, Nsamples=num_samples)  # Shape:
 ↪ [5, 1, num_classes]
            probs = probs.squeeze(1)  # Remove batch dimension -> [5,␣
 ↪num_classes]

            # Check if predictions are not all highly confident
            max_probs = probs.max(dim=1)[0]
            if max_probs.mean() < 0.5:  # If average confidence is less than 99.
 ↪9%
                found_uncertain = True
                break

except StopIteration:
    if not found_uncertain:
        print("Could not find uncertain prediction in entire test set")

# Plot the softmax distributions
plt.figure(figsize=(10, 6))
x_axis = range(probs.shape[1])  # Range over number of classes

for i in range(num_samples):
    plt.plot(x_axis, probs[i].detach().cpu().numpy(), 'o-', alpha=0.7,␣
 ↪label=f'Sample {i+1}')

plt.xlabel('Class')
plt.ylabel('Probability')
plt.title(f'Softmax Distribution Samples for Uncertain Test Example\n' +
          f'True Class: {y_single.item()}\n' +
          f'Total Uncertainty: {uncertainties["total_entropy"].item():.3f}, ' +
          f'Epistemic: {uncertainties["epistemic_entropy"].item():.3f}, ' +
          f'Aleatoric: {uncertainties["aleatoric_entropy"].item():.3f}')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

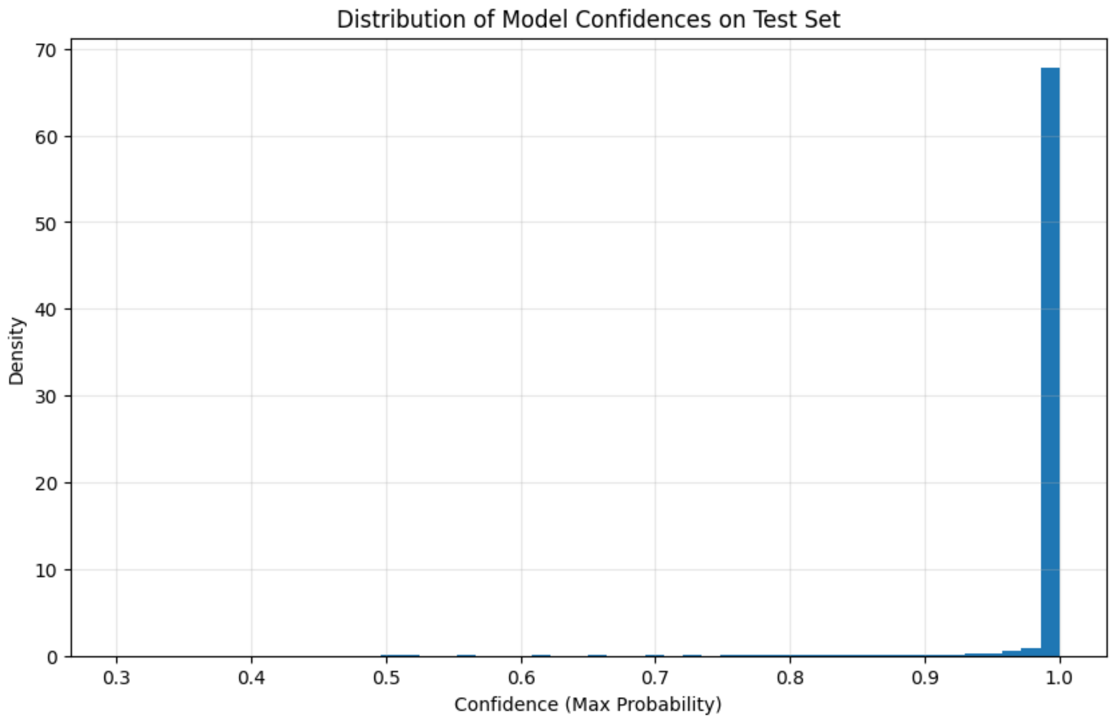
```
# Display the uncertain image
plt.figure(figsize=(5, 5))
img = x_single.squeeze().cpu()
if img.shape[0] == 1:  # If grayscale, remove channel dimension
    img = img.squeeze(0)
plt.imshow(img, cmap='gray' if len(img.shape) == 2 else None)
plt.axis('off')
plt.title(f'Uncertain Image (True Class: {y_single.item()})')
plt.show()
```



Uncertain Image (True Class: 6)

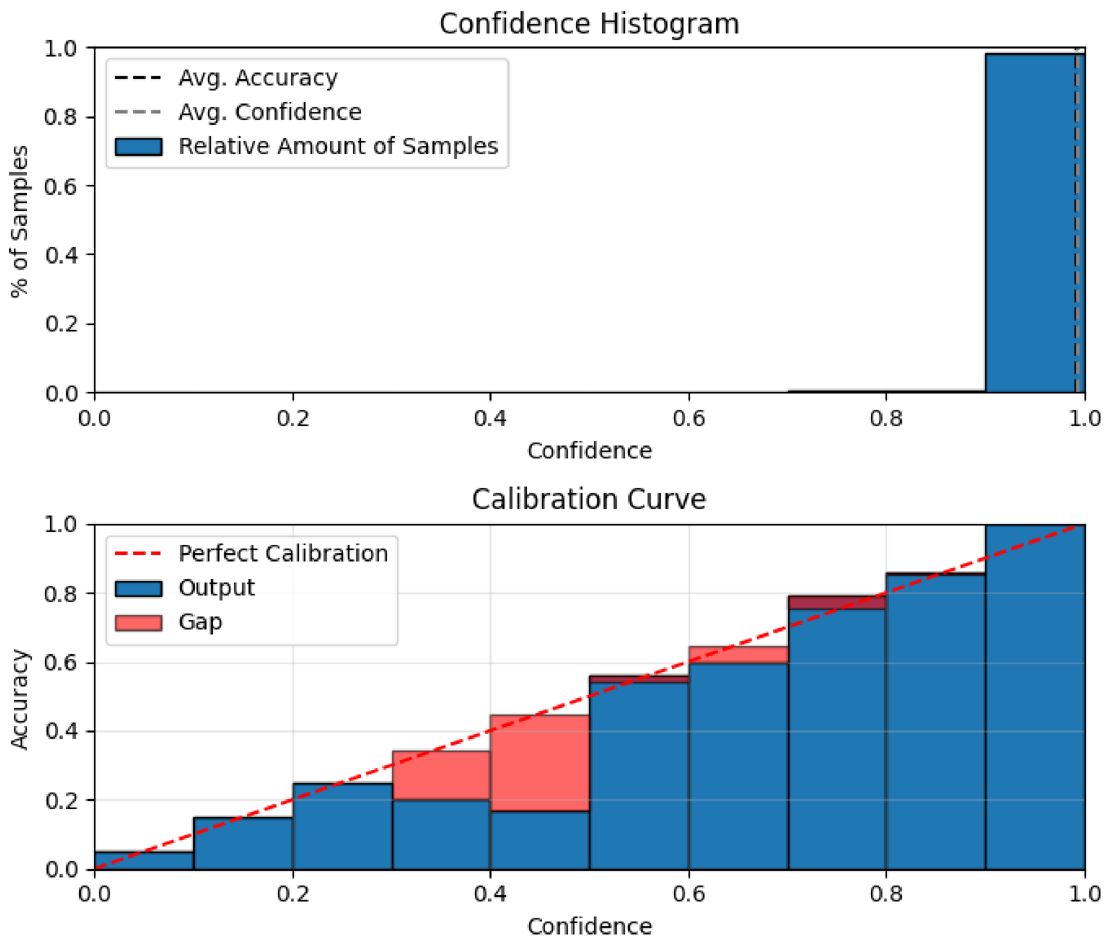


[11]:
```
# Get predictions and confidences for entire test set
all_confidences = []

for x, y in testloader:
    # Get predictions from ensemble
    probs = bll.sample_predict(x)  # Shape: [n_ensemble, batch_size,␣
↪num_classes]

    # Average probabilities across ensemble members
    mean_probs = probs.mean(dim=0)  # Shape: [batch_size, num_classes]

    # Get maximum probability (confidence) for each prediction
    confidences = mean_probs.max(dim=1)[0]
    all_confidences.extend(confidences.cpu().numpy())

# Plot histogram of confidences
plt.figure(figsize=(10, 6))
plt.hist(all_confidences, bins=50, density=True)
plt.xlabel('Confidence (Max Probability)')
plt.ylabel('Density')
plt.title('Distribution of Model Confidences on Test Set')
plt.grid(True, alpha=0.3)
plt.show()
```



[12]:
```
# Get predictions and ground truth labels for calibration curve
all_probs = []
all_labels = []

for x, y in testloader:
    # Get predictions from ensemble
    probs = bll.sample_predict(x)  # Shape: [n_ensemble, batch_size,␣
 ↪num_classes]

    # Average probabilities across ensemble members
    mean_probs = probs.mean(dim=0)  # Shape: [batch_size, num_classes]

    all_probs.extend(mean_probs.cpu().numpy())
    all_labels.extend(y.cpu().numpy())

all_probs = np.array(all_probs)
all_labels = np.array(all_labels)

# Create reliability diagram
from netcal.presentation import ReliabilityDiagram
```

```
# Initialize reliability diagram with 10 bins
diagram = ReliabilityDiagram(10)

# Plot calibration curve
diagram.plot(all_probs, all_labels)
plt.title('Calibration Curve')
plt.grid(True, alpha=0.3)
plt.show()
```



1.4 Bayesian Last Layer - VI

Next we'll train a variational inference version of the Bayesian last layer using the same backbone.

1.4.1 Load the model

[12]:
```python
from models.BLL_VI import BayesianLastLayerVI
importlib.reload(models.BLL_VI)

import os
os.environ['PYTORCH_ENABLE_MPS_FALLBACK'] = '1'

bll_vi = BayesianLastLayerVI(
    backbone=backbone,
    input_dim=256,  # Matches backbone's encoder output
    output_dim=10,  # MNIST classes
    prior_mu=0.0,
    prior_sigma=0.1,
    kl_weight=0.1,  # Weight of KL divergence term
    device=cfg.device
)

# Verify all model components are on the correct device
print(f"Backbone device: {next(bll_vi.backbone.parameters()).device}")
print(f"Last layer device: {next(bll_vi.last_layer.parameters()).device}")
```

```
Backbone device: mps:0
Last layer device: mps:0
```

1.4.2 Train the model

[13]:
```python
from train import train_BLL_VI_classification
importlib.reload(train)

vi_cost_tr, vi_cost_dev, vi_err_tr, vi_err_dev, vi_kl =␣
 ↪train_BLL_VI_classification(
    net=bll_vi,
    name='mnist_bll_vi',
    batch_size=cfg.batch_size,
    nb_epochs=10,
    trainset=trainset,
    valset=valset,
    device=cfg.device,
    lr=1e-3,
    patience=5,
    nb_its_dev=1,
    model_saves_dir=models_dir
)
```

```
Bayesian Last Layer (VI):
```

```
it 0/10, Jtr = 0.002, err = 0.271, KL = 0.088,    time: 7.991567 seconds

    Jdev = 0.033, err = 0.010

best validation error
it 1/10, Jtr = 0.000, err = 0.050, KL = 0.122,    time: 7.038792 seconds

    Jdev = 0.022, err = 0.007

best validation error
it 2/10, Jtr = 0.000, err = 0.036, KL = 0.141,    time: 7.381385 seconds

    Jdev = 0.016, err = 0.005

best validation error
it 3/10, Jtr = 0.000, err = 0.023, KL = 0.156,    time: 9.354598 seconds

    Jdev = 0.016, err = 0.005

best validation error
it 4/10, Jtr = 0.000, err = 0.020, KL = 0.167,    time: 7.653568 seconds

    Jdev = 0.015, err = 0.005

it 5/10, Jtr = 0.000, err = 0.019, KL = 0.179,    time: 7.126051 seconds

    Jdev = 0.013, err = 0.004

best validation error
it 6/10, Jtr = 0.000, err = 0.017, KL = 0.187,    time: 7.302498 seconds

    Jdev = 0.012, err = 0.004

best validation error
it 7/10, Jtr = 0.000, err = 0.014, KL = 0.195,    time: 7.921500 seconds

    Jdev = 0.012, err = 0.004
```

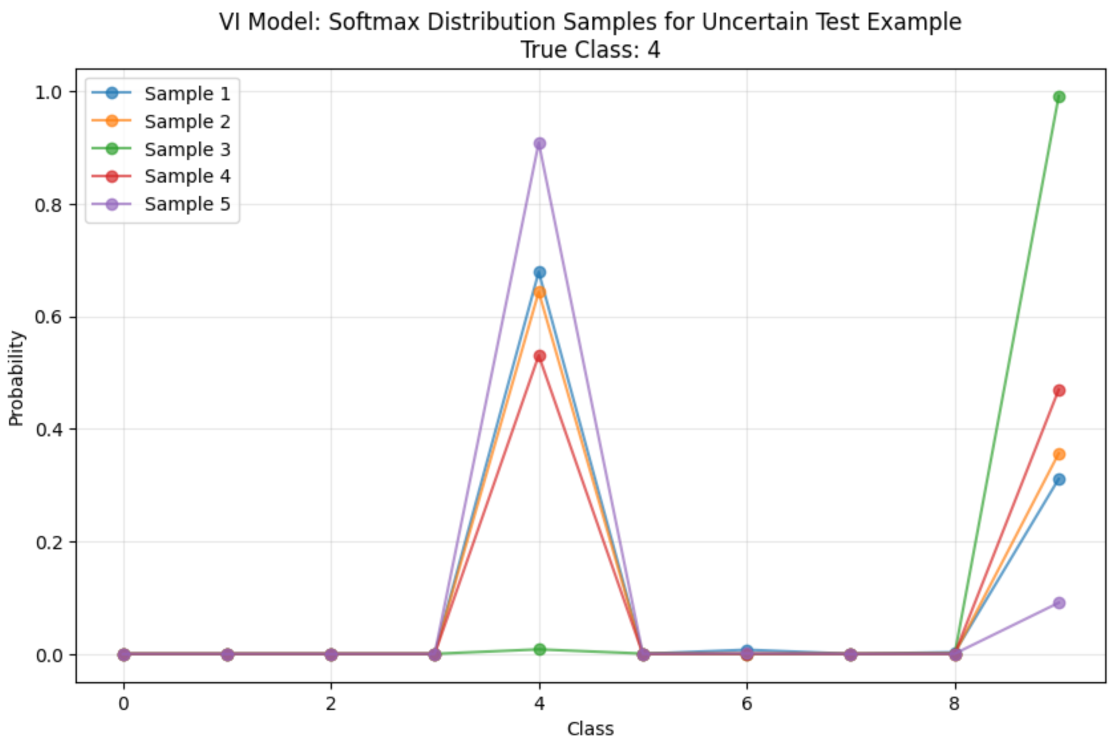
```
best validation error
it 8/10, Jtr = 0.000, err = 0.014, KL = 0.202,    time: 7.961693 seconds


    Jdev = 0.012, err = 0.004


it 9/10, Jtr = 0.000, err = 0.012, KL = 0.207,    time: 7.084738 seconds


    Jdev = 0.011, err = 0.004


best validation error
   average time: 15.772453 seconds


 [save_checkpoint] Saved model state to
../model_saves/mnist_bll_vi_models/BLL_VI_best_8583.pt
Saved best model to: ../model_saves/mnist_bll_vi_models/BLL_VI_best_8583.pt
```

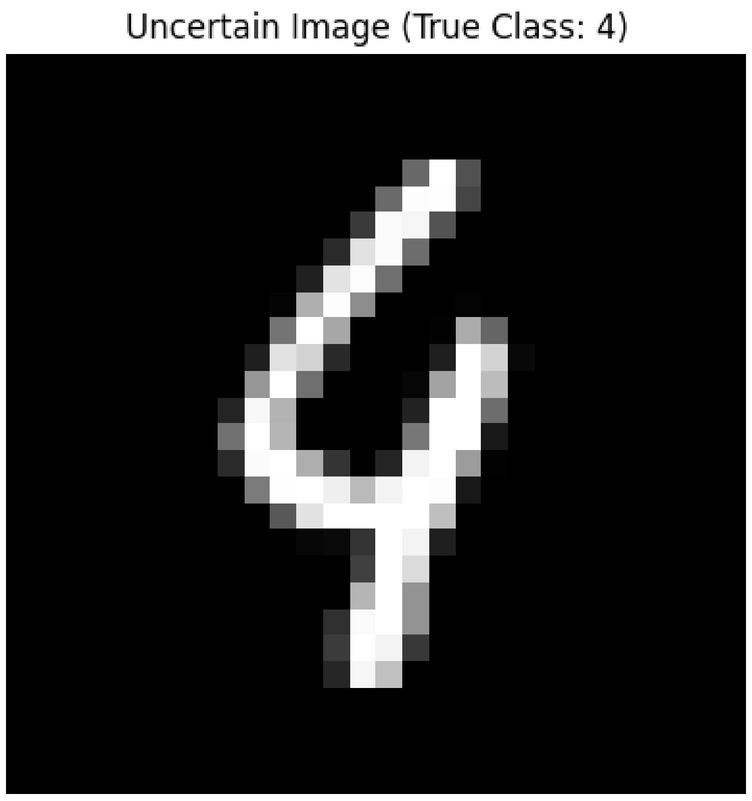
```
<Figure size 640x480 with 0 Axes>
```

1.4.3 Test the model

To check for Bayesian behaviour, we find an uncertain prediction and visualize multiple samples from the posterior.

[17]:
```python
import torch.nn.functional as F

# Search for an uncertain prediction and visualize multiple samples
test_iter = iter(testloader)
found_uncertain = False
max_tries = 100
num_samples = 5

try:
    while not found_uncertain:
        x, y = next(test_iter)
        # Look at each example in the batch
        for i in range(len(x)):
            x_single = x[i:i+1]
            y_single = y[i:i+1]

            # Get multiple predictions for this single example
            outputs = []
            with torch.no_grad():  # Add no_grad context
                for _ in range(num_samples):
                    logits = bll_vi(x_single)
                    probs = F.softmax(logits, dim=1)
                    outputs.append(probs)

            # Stack predictions
            probs = torch.stack(outputs)  # Shape: [num_samples, 1, num_classes]
            probs = probs.squeeze(1)  # Remove batch dimension -> [num_samples,␣
 ↪num_classes]

            # Check if predictions are not all highly confident
            max_probs = probs.max(dim=1)[0]
            if max_probs.mean() < 0.8:  # If average confidence is less than 60%
                found_uncertain = True
                break

except StopIteration:
    if not found_uncertain:
        print("Could not find uncertain prediction in entire test set")

# Plot the softmax distributions
plt.figure(figsize=(10, 6))
x_axis = range(probs.shape[1])  # Range over number of classes

for i in range(num_samples):
    plt.plot(x_axis, probs[i].detach().cpu().numpy(), 'o-', alpha=0.7,␣
 ↪label=f'Sample {i+1}')

plt.xlabel('Class')
plt.ylabel('Probability')
plt.title(f'VI Model: Softmax Distribution Samples for Uncertain Test␣
 ↪Example\nTrue Class: {y_single.item()}')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# Display the uncertain image
plt.figure(figsize=(5, 5))
img = x_single.squeeze().cpu()
if img.shape[0] == 1:  # If grayscale, remove channel dimension
    img = img.squeeze(0)
plt.imshow(img, cmap='gray' if len(img.shape) == 2 else None)
plt.axis('off')
plt.title(f'Uncertain Image (True Class: {y_single.item()})')
plt.show()
```


VI Model: Softmax Distribution Samples for Uncertain Test Example
True Class: 4
Sample 1
Sample 2
Sample 3
Sample 4
Sample 5
Probability
Class
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8


Uncertain Image (True Class: 4)