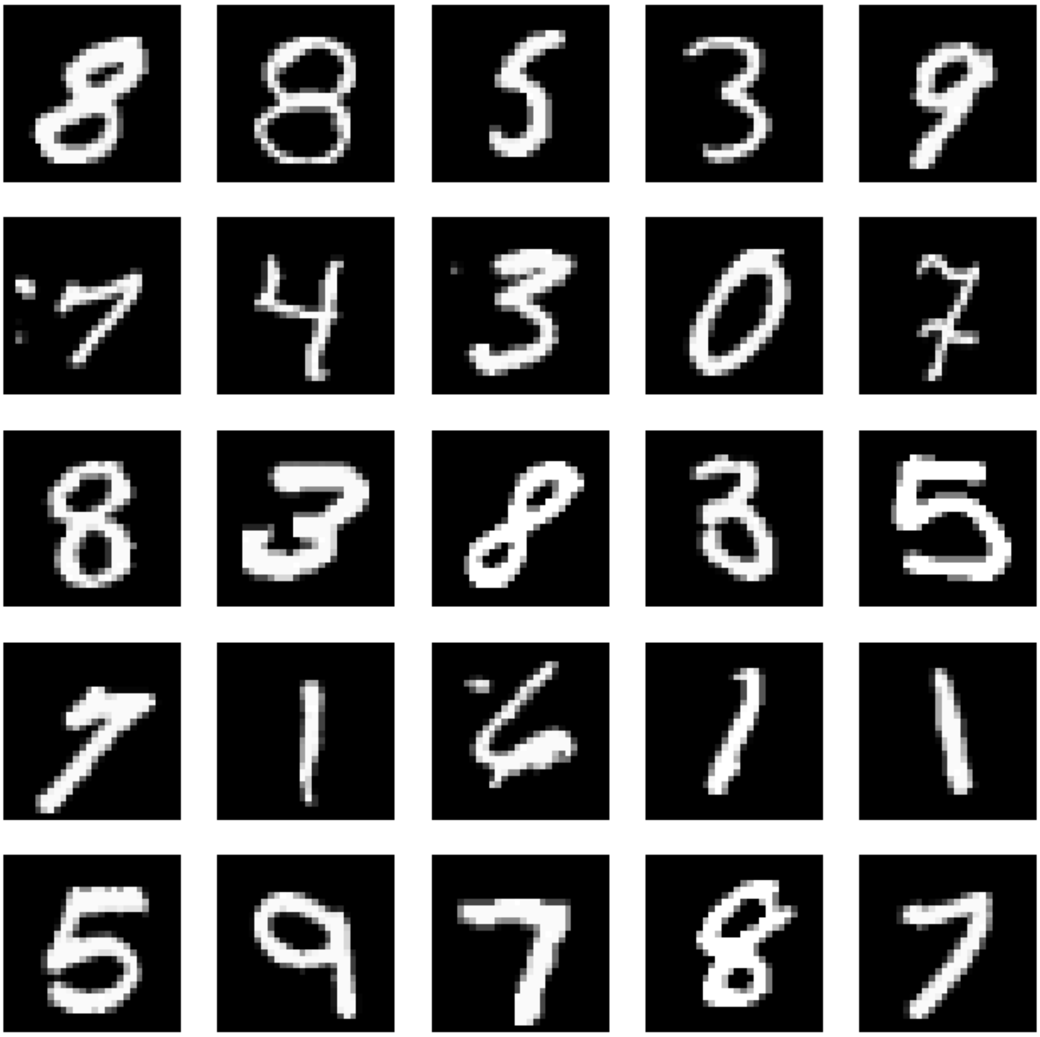
Generate print hello world using rot13 Close

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, SubsetRandomSampler
from torchvision import datasets, transforms
import time


BATCH_SIZE = 64
NUM_EPOCHS = 20
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')


transform = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))
])
train_dataset = datasets.MNIST(root='./data',
train=True,
download=True,
transform=transform)
valid_dataset = datasets.MNIST(root='./data',
train=True,
transform=transform)
test_dataset = datasets.MNIST(root='./data',
train=False,
transform=transform)


validation_fraction = 0.1
num = int(validation_fraction * 60000)
train_indices = torch.arange(0, 60000 - num)
valid_indices = torch.arange(60000 - num, 60000)


train_sampler = SubsetRandomSampler(train_indices)
valid_sampler = SubsetRandomSampler(valid_indices)


train_loader = DataLoader(dataset=train_dataset,
batch_size=BATCH_SIZE,
drop_last=True,
sampler=train_sampler)
valid_loader = DataLoader(dataset=valid_dataset,
batch_size=BATCH_SIZE,
sampler=valid_sampler)
test_loader = DataLoader(dataset=test_dataset,
batch_size=BATCH_SIZE,
shuffle=False)


# Checking the dataset
for images, labels in train_loader:
  print('Image batch dimensions:', images.shape)
  print('Image label dimensions:', labels.shape)
  break
```

```
Image batch dimensions: torch.Size([64, 1, 28, 28])
Image label dimensions: torch.Size([64])
```

```
import matplotlib.pyplot as plt
# Display a grid of sample images
plt.figure(figsize=(10, 10))
for i, (images, labels) in enumerate(train_loader):
  for j in range(25):
    plt.subplot(5, 5, j + 1)
    plt.imshow(images[j].squeeze(), cmap='gray')
    plt.axis('off')
  break
plt.show()
```



```
class MLP(nn.Module):
  def __init__(self, num_features, num_hidden_1, num_hidden_2, num_classes):
    super().__init__()
    self.network = torch.nn.Sequential(
    # 1st hidden layer
    torch.nn.Flatten(),
    torch.nn.Linear(num_features, num_hidden_1),
    torch.nn.BatchNorm1d(num_hidden_1),
    torch.nn.ReLU(),
    torch.nn.Dropout(0.5),
    # 2nd hidden layer
    torch.nn.Linear(num_hidden_1, num_hidden_2),
    torch.nn.BatchNorm1d(num_hidden_2),
    torch.nn.ReLU(),
    torch.nn.Dropout(0.3),
    # output layer
    torch.nn.Linear(num_hidden_2, num_classes)
    )
  def forward(self, x):
    logits = self.network(x)
    return logits
```

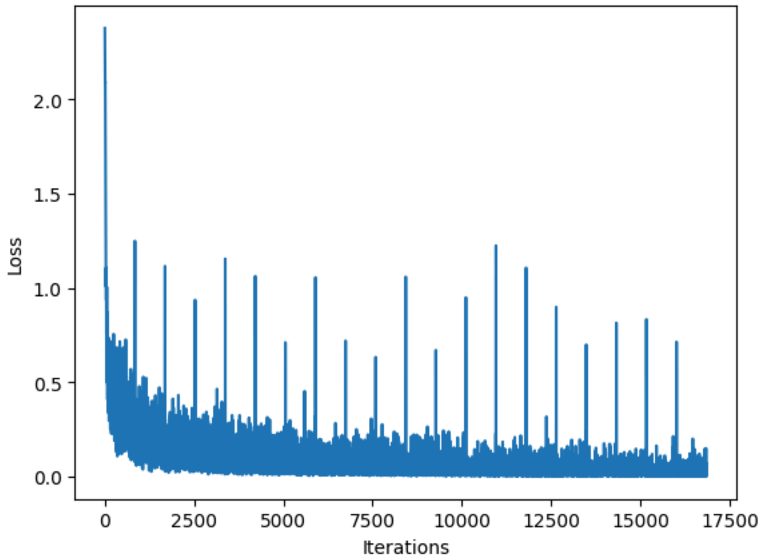
```
model = MLP(num_features=28*28,
num_hidden_1=128,
num_hidden_2=64,
num_classes=10)
model = model.to(DEVICE)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9,
weight_decay=0.0001)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
factor=0.1,
mode='min')
```

```
# Define your optimizer (using Adam)
optimizer = optim.Adam(model.parameters(), lr=0.001)


import torch.optim as optim
from torch.optim import lr_scheduler
# Create the scheduler with step decay
scheduler = lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)


# Create the scheduler with exponential decay
scheduler = lr_scheduler.ExponentialLR(optimizer, gamma=0.95)


def compute_accuracy(data_loader):
  with torch.no_grad():
    correct_pred, num_examples = 0, 0
    for i, (features, targets) in enumerate(data_loader):
      features = features.to(DEVICE)
      targets = targets.float().to(DEVICE)
      logits = model(features)
      _, predicted_labels = torch.max(logits, 1)
      num_examples += targets.size(0)
      correct_pred += (predicted_labels == targets).sum()
  return correct_pred.float()/num_examples * 100



start_time = time.time()
minibatch_loss_list, train_acc_list, valid_acc_list = [], [], []
for epoch in range(NUM_EPOCHS):

    model.train()

    for batch_idx, (features, targets) in enumerate(train_loader):

        features = features.to(DEVICE)
        targets = targets.to(DEVICE)
# ## FORWARD AND BACK PROP
        logits = model(features)
#loss = F.cross_entropy(logits, targets)
        loss = criterion(logits, targets)
        optimizer.zero_grad()
        loss.backward()
# ## UPDATE MODEL PARAMETERS
        optimizer.step()
# ## LOGGING
        minibatch_loss_list.append(loss.item())
        logging_interval = 100
        if not batch_idx % logging_interval:
            print("Epoch: ", epoch+1,"/", NUM_EPOCHS,"| Batch ",batch_idx,
            "/",len(train_loader), f'| Loss: {loss:.4f}')
            model.eval()
            with torch.no_grad():# save memory during inference

                train_acc = compute_accuracy(train_loader)
                valid_acc = compute_accuracy(valid_loader)
                print("Epoch: ", epoch+1, "/",NUM_EPOCHS,
                f'| Train: {train_acc :.2f}% '
                f'| Validation: {valid_acc :.2f}%')
                train_acc_list.append(train_acc.item())
                valid_acc_list.append(valid_acc.item())
                elapsed = (time.time() - start_time)/60
                print("Time elapsed: ",elapsed, " min")
                scheduler.step(minibatch_loss_list[-1])
                elapsed = (time.time() - start_time)/60
                print(f'Total Training Time: {elapsed:.2f} min')
                test_acc = compute_accuracy(test_loader)
                print(f'Test accuracy {test_acc :.2f}%')
```

```
Total Training Time: 58.74 min
Test accuracy 97.78%
Epoch:  20 / 20 | Batch  100 / 843 | Loss: 0.0402
Epoch:  20 / 20 | Train: 99.29% | Validation: 98.13%
Time elapsed:  59.08817561070124  min
Total Training Time: 59.09 min
Test accuracy 97.66%
Epoch:  20 / 20 | Batch  200 / 843 | Loss: 0.0071
Epoch:  20 / 20 | Train: 99.30% | Validation: 98.12%
Time elapsed:  59.415673883756  min
Total Training Time: 59.42 min
Test accuracy 97.77%
Epoch:  20 / 20 | Batch  300 / 843 | Loss: 0.0114
Epoch:  20 / 20 | Train: 99.51% | Validation: 98.10%
Time elapsed:  59.76093515952428  min
Total Training Time: 59.76 min
Test accuracy 97.82%
Epoch:  20 / 20 | Batch  400 / 843 | Loss: 0.0095
Epoch:  20 / 20 | Train: 99.31% | Validation: 97.77%
Time elapsed:  60.11885837713877  min
Total Training Time: 60.12 min
Test accuracy 97.63%
Epoch:  20 / 20 | Batch  500 / 843 | Loss: 0.0156
Epoch:  20 / 20 | Train: 99.31% | Validation: 97.77%
Time elapsed:  60.45131976207097  min
Total Training Time: 60.45 min
Test accuracy 97.72%
Epoch:  20 / 20 | Batch  600 / 843 | Loss: 0.0118
Epoch:  20 / 20 | Train: 99.19% | Validation: 97.85%
Time elapsed:  60.7944786588351  min
Total Training Time: 60.79 min
Test accuracy 97.56%
Epoch:  20 / 20 | Batch  700 / 843 | Loss: 0.0392
Epoch:  20 / 20 | Train: 99.47% | Validation: 98.08%
Time elapsed:  61.13306200106938  min
Total Training Time: 61.13 min
Test accuracy 97.92%
Epoch:  20 / 20 | Batch  800 / 843 | Loss: 0.0284
Epoch:  20 / 20 | Train: 99.43% | Validation: 97.95%
Time elapsed:  61.4774317463239  min
Total Training Time: 61.48 min
Test accuracy 97.82%
```

```python
import matplotlib.pyplot as plt
plt.plot(range(len(minibatch_loss_list)), minibatch_loss_list)
plt.xlabel('Iterations')
plt.ylabel('Loss')
```

```
Text(0, 0.5, 'Loss')
```




```python
import numpy as np
num_epochs = len(train_acc_list)
plt.plot(np.arange(1, num_epochs+1),
train_acc_list, label='Training')
plt.plot(np.arange(1, num_epochs+1),
valid_acc_list, label='Validation')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa516a962c0>
```