In [17]:

```
import torch
print("Torch Version: {}".format(torch.__version__))
```

```
Torch Version: 1.8.1
```

In [18]:

```
import numpy as np
print("Numpy Version: {}".format(np.__version__))
```

```
Numpy Version: 1.19.2
```

In [19]:

```
import matplotlib
print("Matplotlib Version: {}".format(matplotlib.__version__))
```

```
Matplotlib Version: 3.3.2
```

In [20]:

```
import torchvision
print("Torchvision Version: {}".format(torchvision.__version__))
```

```
Torchvision Version: 0.9.1
```

In [21]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from torch import nn, optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets,transforms
from torch.utils.data.sampler import SubsetRandomSampler

%matplotlib inline
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

In [22]:

```
batch_size = 20

valid_size = 0.2

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5), (0.5,))
    ])

train_data = datasets.MNIST('MNIST_data', train=True, download=True, transform=transform)

test_data = datasets.MNIST('MNIST_data', train=False, download=True, transform=transform)

num_train = len(train_data)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

train_loader = DataLoader(train_data, batch_size=batch_size, sampler=train_sampler)

valid_loader = DataLoader(train_data, batch_size=batch_size, sampler=valid_sampler)

test_loader = DataLoader(test_data, batch_size=batch_size)
```

In [23]:

```
class Net(nn.Module):
  def __init__(self):
    super().__init__()
    self.fc1 = nn.Linear(784, 256)
    self.fc2 = nn.Linear(256, 128)
    self.fc3 = nn.Linear(128, 64)
    self.fc4 = nn.Linear(64, 10)

  def forward(self, x):
    x = x.view(x.shape[0], -1)  # flattering the input tensor

    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.relu(self.fc3(x))
    x = F.log_softmax(self.fc4(x), dim=1)

    return x
```

In [24]:

```
model = Net()
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.003)
```

In [25]:

```
epochs = 30
train_losses, valid_losses = [], []
valid_loss_min = np.Inf

for e in range(epochs):
    train_loss, valid_loss = 0.0, 0.0
    model.train()
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()*images.size(0)

    model.eval
    for images, labels in valid_loader:
        images, labels = images.to(device), labels.to(device)

        output = model(images)
        loss = criterion(output, labels)
        valid_loss += loss.item()*images.size(0)


    train_loss = train_loss/len(train_loader.sampler)
    train_losses.append(train_loss)
    valid_loss = valid_loss/len(valid_loader.sampler)
    valid_losses.append(valid_loss)

    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        e+1, train_loss, valid_loss))

    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(model.state_dict(), 'mnist_ann.pt')
        valid_loss_min = valid_loss
```

```
Epoch: 1 	Training Loss: 1.645108 	Validation Loss: 0.756156
Validation loss decreased (inf --> 0.756156).  Saving model ...
Epoch: 2 	Training Loss: 0.553181 	Validation Loss: 0.463151
Validation loss decreased (0.756156 --> 0.463151).  Saving model ...
Epoch: 3 	Training Loss: 0.388362 	Validation Loss: 0.366195
Validation loss decreased (0.463151 --> 0.366195).  Saving model ...
Epoch: 4 	Training Loss: 0.329024 	Validation Loss: 0.319076
Validation loss decreased (0.366195 --> 0.319076).  Saving model ...
Epoch: 5 	Training Loss: 0.293908 	Validation Loss: 0.296290
Validation loss decreased (0.319076 --> 0.296290).  Saving model ...
Epoch: 6 	Training Loss: 0.267165 	Validation Loss: 0.270356
Validation loss decreased (0.296290 --> 0.270356).  Saving model ...
Epoch: 7 	Training Loss: 0.244221 	Validation Loss: 0.249475
Validation loss decreased (0.270356 --> 0.249475).  Saving model ...
Epoch: 8 	Training Loss: 0.223188 	Validation Loss: 0.230348
Validation loss decreased (0.249475 --> 0.230348).  Saving model ...
Epoch: 9 	Training Loss: 0.204602 	Validation Loss: 0.211217
Validation loss decreased (0.230348 --> 0.211217).  Saving model ...
Epoch: 10 	Training Loss: 0.187142 	Validation Loss: 0.197522
Validation loss decreased (0.211217 --> 0.197522).  Saving model ...
Epoch: 11 	Training Loss: 0.172631 	Validation Loss: 0.186119
Validation loss decreased (0.197522 --> 0.186119).  Saving model ...
Epoch: 12 	Training Loss: 0.160248 	Validation Loss: 0.175641
Validation loss decreased (0.186119 --> 0.175641).  Saving model ...
Epoch: 13 	Training Loss: 0.147675 	Validation Loss: 0.173294
Validation loss decreased (0.175641 --> 0.173294).  Saving model ...
Epoch: 14 	Training Loss: 0.138442 	Validation Loss: 0.152355
Validation loss decreased (0.173294 --> 0.152355).  Saving model ...
Epoch: 15 	Training Loss: 0.129434 	Validation Loss: 0.151776
Validation loss decreased (0.152355 --> 0.151776).  Saving model ...
Epoch: 16 	Training Loss: 0.120464 	Validation Loss: 0.142655
Validation loss decreased (0.151776 --> 0.142655).  Saving model ...
Epoch: 17 	Training Loss: 0.112771 	Validation Loss: 0.143752
Epoch: 18 	Training Loss: 0.107162 	Validation Loss: 0.128059
Validation loss decreased (0.142655 --> 0.128059).  Saving model ...
Epoch: 19 	Training Loss: 0.100594 	Validation Loss: 0.124086
Validation loss decreased (0.128059 --> 0.124086).  Saving model ...
Epoch: 20 	Training Loss: 0.094637 	Validation Loss: 0.121034
Validation loss decreased (0.124086 --> 0.121034).  Saving model ...
Epoch: 21 	Training Loss: 0.089433 	Validation Loss: 0.115384
Validation loss decreased (0.121034 --> 0.115384).  Saving model ...
Epoch: 22 	Training Loss: 0.084652 	Validation Loss: 0.114727
Validation loss decreased (0.115384 --> 0.114727).  Saving model ...
Epoch: 23 	Training Loss: 0.080095 	Validation Loss: 0.110428
Validation loss decreased (0.114727 --> 0.110428).  Saving model ...
Epoch: 24 	Training Loss: 0.075144 	Validation Loss: 0.119690
```

```
Epoch: 25       Training Loss: 0.071518         Validation Loss: 0.105376
Validation loss decreased (0.110428 --> 0.105376).  Saving model ...
Epoch: 26       Training Loss: 0.067284         Validation Loss: 0.104210
Validation loss decreased (0.105376 --> 0.104210).  Saving model ...
Epoch: 27       Training Loss: 0.064253         Validation Loss: 0.104602
Epoch: 28       Training Loss: 0.060949         Validation Loss: 0.110857
Epoch: 29       Training Loss: 0.058058         Validation Loss: 0.101158
Validation loss decreased (0.104210 --> 0.101158).  Saving model ...
Epoch: 30       Training Loss: 0.054616         Validation Loss: 0.100849
Validation loss decreased (0.101158 --> 0.100849).  Saving model ...
```

In [26]:

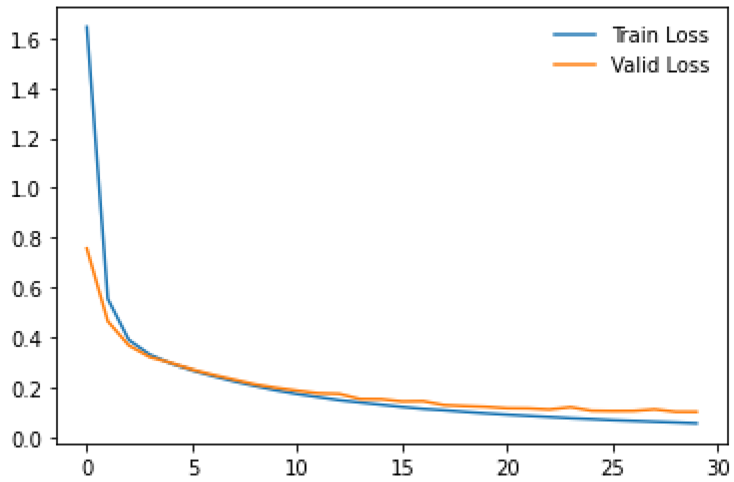
```
model.load_state_dict(torch.load('mnist_ann.pt', map_location=torch.device(device)))
```

Out[26]:

```
<All keys matched successfully>
```

In [30]:

```
plt.plot(train_losses, label="Train Loss")
plt.plot(valid_losses, label="Valid Loss")
plt.legend(frameon=False)
plt.show()
```



In [28]:

```
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model.eval()

for images, labels in test_loader:
    images, labels = images.to(device), labels.to(device)

    output = model(images)
    loss = criterion(output, labels)
    test_loss += loss.item()*images.size(0)
    _, pred = torch.max(output, 1)
    correct = np.squeeze(pred.eq(labels.data.view_as(pred)))

    for i in range(batch_size):
        label = labels.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1


test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %2s: %2d%% (%2d/%2d)' % (
            str([i]), 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 0.090483

Test Accuracy of [0]: 98% (965/980)
Test Accuracy of [1]: 99% (1127/1135)
Test Accuracy of [2]: 97% (1011/1032)
Test Accuracy of [3]: 95% (964/1010)
Test Accuracy of [4]: 97% (957/982)
Test Accuracy of [5]: 96% (858/892)
Test Accuracy of [6]: 98% (942/958)
Test Accuracy of [7]: 96% (991/1028)
Test Accuracy of [8]: 97% (945/974)
Test Accuracy of [9]: 95% (960/1009)

Test Accuracy (Overall): 97% (9720/10000)
```

In [34]:

```python
images, labels = next(iter(train_loader))
image, labels = images.to(device), labels.to(device)

img = images[1].view(1,784)

with torch.no_grad():
  log_ps = model(img)

ps = F.softmax(log_ps, dim=1)
fig = plt.figure(figsize=(6,9))
ax1 = plt.subplot(121)
ax1.imshow(img.resize_(1, 28, 28).cpu().numpy().squeeze())
ax1.axis('off')

ax2 = plt.subplot(122)
ax2.set_yticks(np.arange(10))
ax2.set_xticks(np.arange(0,1.1,0.25))
ax2.set_xlim(0,1.1)
ax2.set_aspect(0.1)
ax2.set_title('Class Probability')
plt.barh(np.arange(10),ps.data.cpu().numpy().squeeze())

plt.tight_layout()

```