In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import sampler

import torchvision.datasets as dset
import torchvision.transforms as T

%matplotlib inline
```

In [2]:
```
plt.rcParams['figure.figsize'] = [7, 7]
```

Dataset preparation

Load MNIST

In [3]:
```
# Subtracts the pre-calculated mean of training set from train, validation and te
transform = T.Compose([T.ToTensor(), T.Normalize((0.13087,), (1,))])

# Dataset
mnist_train = dset.MNIST('../datasets', train=True, download=True, transform=tran
mnist_test = dset.MNIST('../datasets', train=False, download=True, transform=tran
```

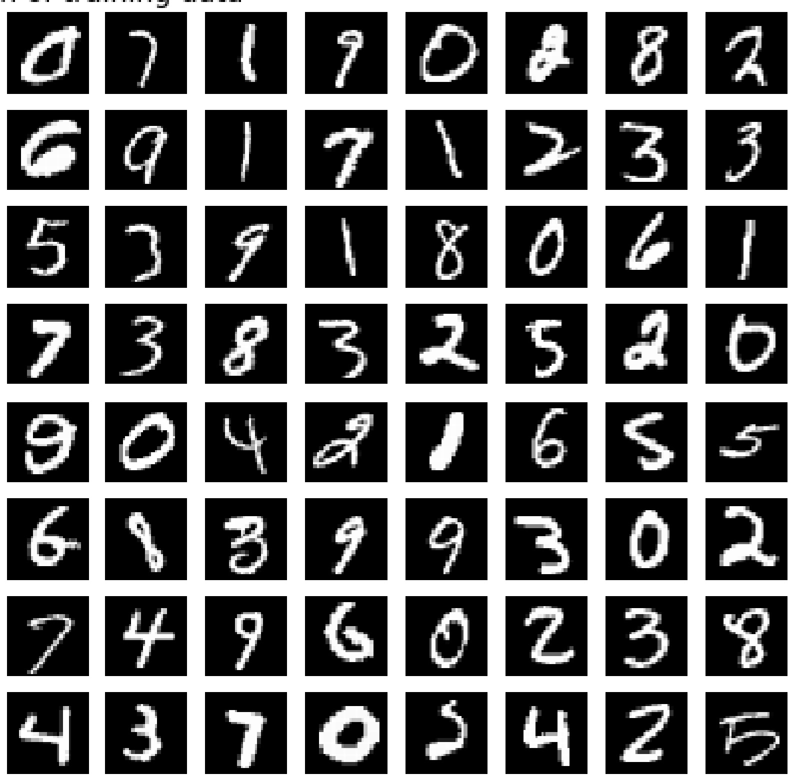
Seperate train, validation, and test set

In [4]:
```
# DataLoaders
num_train = 48000
loader_train = DataLoader(mnist_train, batch_size=64,
                          sampler=sampler.SubsetRandomSampler(range(num_train)))
loader_val = DataLoader(mnist_train, batch_size=64,
                        sampler=sampler.SubsetRandomSampler(range(num_train, 6000
loader_test = DataLoader(mnist_test, batch_size=64)
```

Visualize a minibatch of data

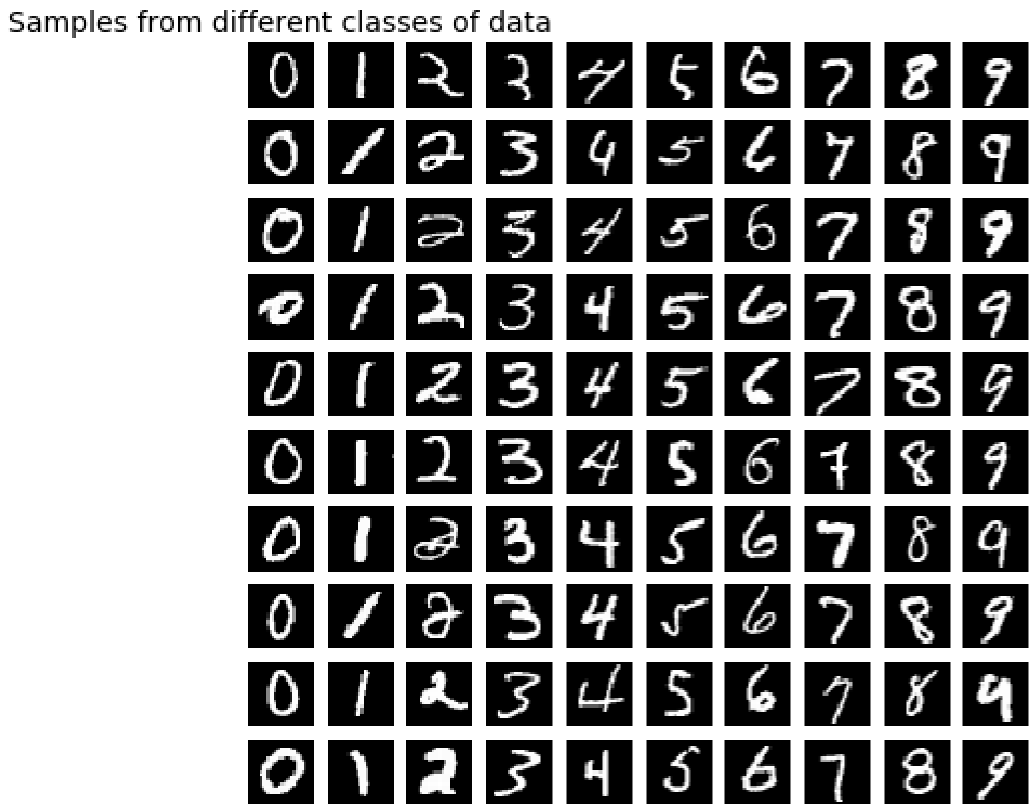
In [5]:
```
X, y = iter(loader_train).next()
data = X.numpy()

for i in range(8):
    for j in range(8):
        plt_idx = i * 8 + j + 1
        ax = plt.subplot(8, 8, plt_idx)
        plt.imshow(data[j+(i*8),0]+0.13087, cmap='gray', vmin=0, vmax=1)
        plt.axis('off')
        if i == 0 and j == 0:
            plt.title('A minibatch of training data', size=14)
plt.show()
```

A minibatch of training data



Visualize the different classes of data

In [6]:
```
from utils.visualization_utils import pltClasses

categorized_data = {0:[],1:[], 2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[]}
for X, y in loader_train:
    data = X.numpy()
    label = y.numpy().astype(int)
    for num, category in enumerate(label):
        categorized_data[category].append([data[num]])
    category_lens = np.array([len(samples) for category,samples in categorized_da
    if np.min(category_lens)>10:
        break

pltClasses(categorized_data,'Samples from different classes of data')
```



Training the Model

Select device

In [7]:
```
# CUDA for PyTorch
use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")
print('Using device=GPU') if use_cuda else print('Using device=CPU')
```

```
Using device=GPU
```

Create Model

In [8]:
```
from classifiers.cnn import ConvNet
model = ConvNet()
```

Optimal Hyperparameter Search

Optimal hyperparameters found: learning_rate = 0.2, weight_decay = 0.002.

In [9]:
```
# from utils.training_utils import train_model
# print('~~~ Training with GPU ~~~\n') if use_cuda else print('~~~ Training with (

# # Experiment Settings
# print_every = 200

# # Hyperparameters
# momentum = 0
# num_epochs = 1

# for i in range(100):
#     learning_rate = 10**np.random.uniform(-1.5,-0.5)
#     weight_decay = 10**np.random.uniform(-4,-2)
#     print('lr: %.4f, reg: %.4f\n' % (learning_rate, weight_decay))
#     model = ConvNet()
#     optimizer = optim.SGD(model.parameters(), lr=learning_rate,
#                           momentum=momentum, weight_decay=weight_decay)
#     train_history = train_model(model, optimizer, loader_train,
#                                 loader_val, num_epochs,
#                                 print_every, device)
```

Start Training

In [10]:
```
# Experiment Settings
print_every = 100

# Hyperparameters
learning_rate = 0.2
momentum = 0
weight_decay = 0.002
num_epochs = 10
```

In [11]:
```
from utils.training_utils import train_model
print('~~~ Training with GPU ~~~') if use_cuda else print('~~~ Training with CPU 
num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print('Model has %.2fK trainable parameters.\n' % (num_params/1000))

optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=momentum, weight_decay=weight_decay)
train_history = train_model(model, optimizer, loader_train,
                            loader_val, num_epochs,
                            print_every, device)
```

```
~~~ Training with GPU ~~~
Model has 661.13K trainable parameters.

Epoch 1, Iteration 0:
Training data: loss = 2.3085, accuracy = 9.38
Validation data: loss = 2.3043, accuracy = 9.89

Epoch 1, Iteration 100:
Training data: loss = 2.2945, accuracy = 14.06
Validation data: loss = 2.2930, accuracy = 10.60

Epoch 1, Iteration 200:
Training data: loss = 0.4439, accuracy = 87.50
Validation data: loss = 0.4010, accuracy = 86.92

Epoch 1, Iteration 300:
Training data: loss = 0.2322, accuracy = 92.19
Validation data: loss = 0.1958, accuracy = 93.52

Epoch 1, Iteration 400:
Training data: loss = 0.0924, accuracy = 96.88
Validation data: loss = 0.1293, accuracy = 95.88

Epoch 1, Iteration 500:
Training data: loss = 0.0858, accuracy = 96.88
Validation data: loss = 0.1008, accuracy = 96.83

Epoch 1, Iteration 600:
Training data: loss = 0.1344, accuracy = 95.31
Validation data: loss = 0.0805, accuracy = 97.42

Epoch 1, Iteration 700:
Training data: loss = 0.1082, accuracy = 96.88
Validation data: loss = 0.0846, accuracy = 97.40

-----------
Epoch 2, Iteration 800:
Training data: loss = 0.0060, accuracy = 100.00
Validation data: loss = 0.0717, accuracy = 97.84

Epoch 2, Iteration 900:
Training data: loss = 0.1213, accuracy = 96.88
Validation data: loss = 0.0890, accuracy = 97.28

Epoch 2, Iteration 1000:
Training data: loss = 0.0457, accuracy = 96.88
```

```
Validation data: loss = 0.0739, accuracy = 97.84

Epoch 2, Iteration 1100:
Training data: loss = 0.1476, accuracy = 96.88
Validation data: loss = 0.0593, accuracy = 98.13

Epoch 2, Iteration 1200:
Training data: loss = 0.1176, accuracy = 96.88
Validation data: loss = 0.0789, accuracy = 97.58

Epoch 2, Iteration 1300:
Training data: loss = 0.0387, accuracy = 98.44
Validation data: loss = 0.0685, accuracy = 97.78

Epoch 2, Iteration 1400:
Training data: loss = 0.0202, accuracy = 100.00
Validation data: loss = 0.0523, accuracy = 98.18

-----------
Epoch 3, Iteration 1500:
Training data: loss = 0.0145, accuracy = 100.00
Validation data: loss = 0.0571, accuracy = 98.22

Epoch 3, Iteration 1600:
Training data: loss = 0.0091, accuracy = 100.00
Validation data: loss = 0.0506, accuracy = 98.42

Epoch 3, Iteration 1700:
Training data: loss = 0.0193, accuracy = 100.00
Validation data: loss = 0.0520, accuracy = 98.51

Epoch 3, Iteration 1800:
Training data: loss = 0.1016, accuracy = 96.88
Validation data: loss = 0.0544, accuracy = 98.45

Epoch 3, Iteration 1900:
Training data: loss = 0.0167, accuracy = 100.00
Validation data: loss = 0.0538, accuracy = 98.38

Epoch 3, Iteration 2000:
Training data: loss = 0.0280, accuracy = 100.00
Validation data: loss = 0.0542, accuracy = 98.41

Epoch 3, Iteration 2100:
Training data: loss = 0.0662, accuracy = 98.44
Validation data: loss = 0.0654, accuracy = 98.01

Epoch 3, Iteration 2200:
Training data: loss = 0.1083, accuracy = 96.88
Validation data: loss = 0.0532, accuracy = 98.48

-----------
Epoch 4, Iteration 2300:
Training data: loss = 0.0774, accuracy = 95.31
Validation data: loss = 0.0681, accuracy = 98.01

Epoch 4, Iteration 2400:
```

```
Training data: loss = 0.0667, accuracy = 95.31
Validation data: loss = 0.0458, accuracy = 98.59

Epoch 4, Iteration 2500:
Training data: loss = 0.0097, accuracy = 100.00
Validation data: loss = 0.0440, accuracy = 98.62

Epoch 4, Iteration 2600:
Training data: loss = 0.0069, accuracy = 100.00
Validation data: loss = 0.0460, accuracy = 98.62

Epoch 4, Iteration 2700:
Training data: loss = 0.0109, accuracy = 100.00
Validation data: loss = 0.0634, accuracy = 98.00

Epoch 4, Iteration 2800:
Training data: loss = 0.1348, accuracy = 96.88
Validation data: loss = 0.0618, accuracy = 98.12

Epoch 4, Iteration 2900:
Training data: loss = 0.0756, accuracy = 98.44
Validation data: loss = 0.0740, accuracy = 97.59

-----------
Epoch 5, Iteration 3000:
Training data: loss = 0.1012, accuracy = 98.44
Validation data: loss = 0.0455, accuracy = 98.49

Epoch 5, Iteration 3100:
Training data: loss = 0.1252, accuracy = 96.88
Validation data: loss = 0.0530, accuracy = 98.42

Epoch 5, Iteration 3200:
Training data: loss = 0.1135, accuracy = 95.31
Validation data: loss = 0.0548, accuracy = 98.47

Epoch 5, Iteration 3300:
Training data: loss = 0.0138, accuracy = 100.00
Validation data: loss = 0.0453, accuracy = 98.59

Epoch 5, Iteration 3400:
Training data: loss = 0.0069, accuracy = 100.00
Validation data: loss = 0.0451, accuracy = 98.58

Epoch 5, Iteration 3500:
Training data: loss = 0.0321, accuracy = 98.44
Validation data: loss = 0.0583, accuracy = 98.09

Epoch 5, Iteration 3600:
Training data: loss = 0.0826, accuracy = 96.88
Validation data: loss = 0.0501, accuracy = 98.39

Epoch 5, Iteration 3700:
Training data: loss = 0.0086, accuracy = 100.00
Validation data: loss = 0.0415, accuracy = 98.72

-----------
```

```
Epoch 6, Iteration 3800:
Training data: loss = 0.0232, accuracy = 100.00
Validation data: loss = 0.0421, accuracy = 98.67

Epoch 6, Iteration 3900:
Training data: loss = 0.0014, accuracy = 100.00
Validation data: loss = 0.0550, accuracy = 98.23

Epoch 6, Iteration 4000:
Training data: loss = 0.0515, accuracy = 98.44
Validation data: loss = 0.0437, accuracy = 98.68

Epoch 6, Iteration 4100:
Training data: loss = 0.0045, accuracy = 100.00
Validation data: loss = 0.0389, accuracy = 98.72

Epoch 6, Iteration 4200:
Training data: loss = 0.0368, accuracy = 98.44
Validation data: loss = 0.0434, accuracy = 98.70

Epoch 6, Iteration 4300:
Training data: loss = 0.0498, accuracy = 98.44
Validation data: loss = 0.0537, accuracy = 98.20

Epoch 6, Iteration 4400:
Training data: loss = 0.0175, accuracy = 100.00
Validation data: loss = 0.0525, accuracy = 98.39

-----------
Epoch 7, Iteration 4500:
Training data: loss = 0.0336, accuracy = 98.44
Validation data: loss = 0.0429, accuracy = 98.77

Epoch 7, Iteration 4600:
Training data: loss = 0.0064, accuracy = 100.00
Validation data: loss = 0.0531, accuracy = 98.36

Epoch 7, Iteration 4700:
Training data: loss = 0.1892, accuracy = 95.31
Validation data: loss = 0.0733, accuracy = 97.85

Epoch 7, Iteration 4800:
Training data: loss = 0.0227, accuracy = 98.44
Validation data: loss = 0.0457, accuracy = 98.66

Epoch 7, Iteration 4900:
Training data: loss = 0.0314, accuracy = 98.44
Validation data: loss = 0.0423, accuracy = 98.76

Epoch 7, Iteration 5000:
Training data: loss = 0.0653, accuracy = 95.31
Validation data: loss = 0.0545, accuracy = 98.37

Epoch 7, Iteration 5100:
Training data: loss = 0.0129, accuracy = 100.00
Validation data: loss = 0.0417, accuracy = 98.74
```

```
Epoch 7, Iteration 5200:
Training data: loss = 0.0188, accuracy = 100.00
Validation data: loss = 0.0414, accuracy = 98.76

-----------
Epoch 8, Iteration 5300:
Training data: loss = 0.0282, accuracy = 100.00
Validation data: loss = 0.0428, accuracy = 98.71

Epoch 8, Iteration 5400:
Training data: loss = 0.0194, accuracy = 100.00
Validation data: loss = 0.0468, accuracy = 98.53

Epoch 8, Iteration 5500:
Training data: loss = 0.0520, accuracy = 98.44
Validation data: loss = 0.0487, accuracy = 98.49

Epoch 8, Iteration 5600:
Training data: loss = 0.0384, accuracy = 98.44
Validation data: loss = 0.0395, accuracy = 98.86

Epoch 8, Iteration 5700:
Training data: loss = 0.0240, accuracy = 98.44
Validation data: loss = 0.0457, accuracy = 98.62

Epoch 8, Iteration 5800:
Training data: loss = 0.0839, accuracy = 96.88
Validation data: loss = 0.0748, accuracy = 97.71

Epoch 8, Iteration 5900:
Training data: loss = 0.0295, accuracy = 100.00
Validation data: loss = 0.0453, accuracy = 98.63

-----------
Epoch 9, Iteration 6000:
Training data: loss = 0.0670, accuracy = 98.44
Validation data: loss = 0.0737, accuracy = 97.67

Epoch 9, Iteration 6100:
Training data: loss = 0.0366, accuracy = 98.44
Validation data: loss = 0.0516, accuracy = 98.34

Epoch 9, Iteration 6200:
Training data: loss = 0.0161, accuracy = 100.00
Validation data: loss = 0.0513, accuracy = 98.37

Epoch 9, Iteration 6300:
Training data: loss = 0.0285, accuracy = 98.44
Validation data: loss = 0.0415, accuracy = 98.79

Epoch 9, Iteration 6400:
Training data: loss = 0.0291, accuracy = 98.44
Validation data: loss = 0.0471, accuracy = 98.65

Epoch 9, Iteration 6500:
Training data: loss = 0.0666, accuracy = 96.88
Validation data: loss = 0.0739, accuracy = 97.58
```

```
Epoch 9, Iteration 6600:
Training data: loss = 0.0164, accuracy = 100.00
Validation data: loss = 0.0505, accuracy = 98.55

Epoch 9, Iteration 6700:
Training data: loss = 0.0186, accuracy = 100.00
Validation data: loss = 0.0392, accuracy = 98.85

-----------
Epoch 10, Iteration 6800:
Training data: loss = 0.0164, accuracy = 100.00
Validation data: loss = 0.0478, accuracy = 98.58

Epoch 10, Iteration 6900:
Training data: loss = 0.0343, accuracy = 98.44
Validation data: loss = 0.0486, accuracy = 98.41

Epoch 10, Iteration 7000:
Training data: loss = 0.0120, accuracy = 100.00
Validation data: loss = 0.0399, accuracy = 98.78

Epoch 10, Iteration 7100:
Training data: loss = 0.0074, accuracy = 100.00
Validation data: loss = 0.0620, accuracy = 98.20

Epoch 10, Iteration 7200:
Training data: loss = 0.0043, accuracy = 100.00
Validation data: loss = 0.0414, accuracy = 98.83

Epoch 10, Iteration 7300:
Training data: loss = 0.0038, accuracy = 100.00
Validation data: loss = 0.0446, accuracy = 98.63

Epoch 10, Iteration 7400:
Training data: loss = 0.0087, accuracy = 100.00
Validation data: loss = 0.0518, accuracy = 98.44

-----------
```

Evaluating the model

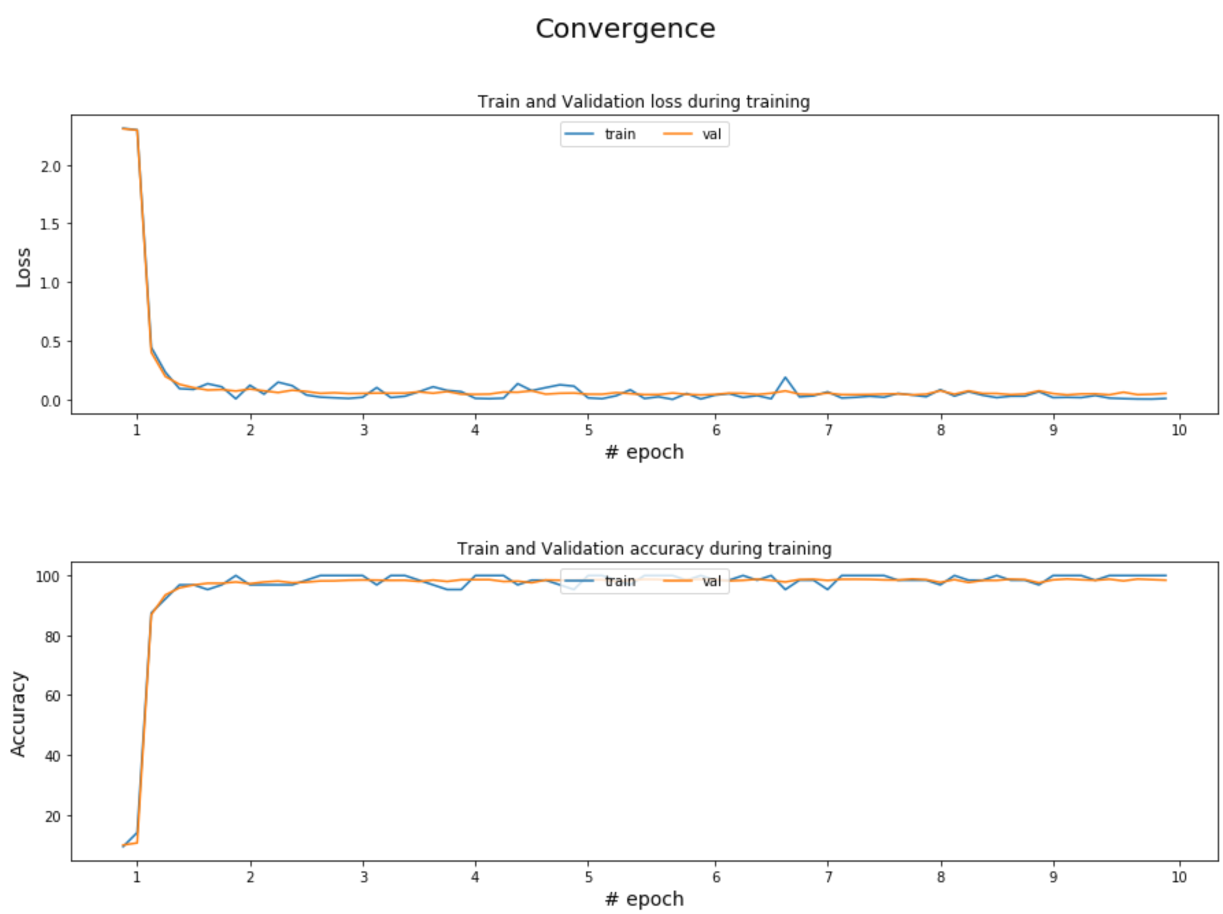
1. Inference Accuracy

In [12]:
```
from utils.training_utils import evaluation
train_acc, _ = evaluation(model, loader_train, None, device)
val_acc, _ = evaluation(model, loader_val, None, device)
test_acc, _ = evaluation(model, loader_test, None, device)
print('Train Accuracy: %.2f%%' % train_acc)
print('Validation Accuracy: %.2f%%' % val_acc)
print('Test Accuracy: %.2f%%' % test_acc)
```

```
Train Accuracy: 98.66%
Validation Accuracy: 98.21%
Test Accuracy: 98.47%
```

2. Convergence

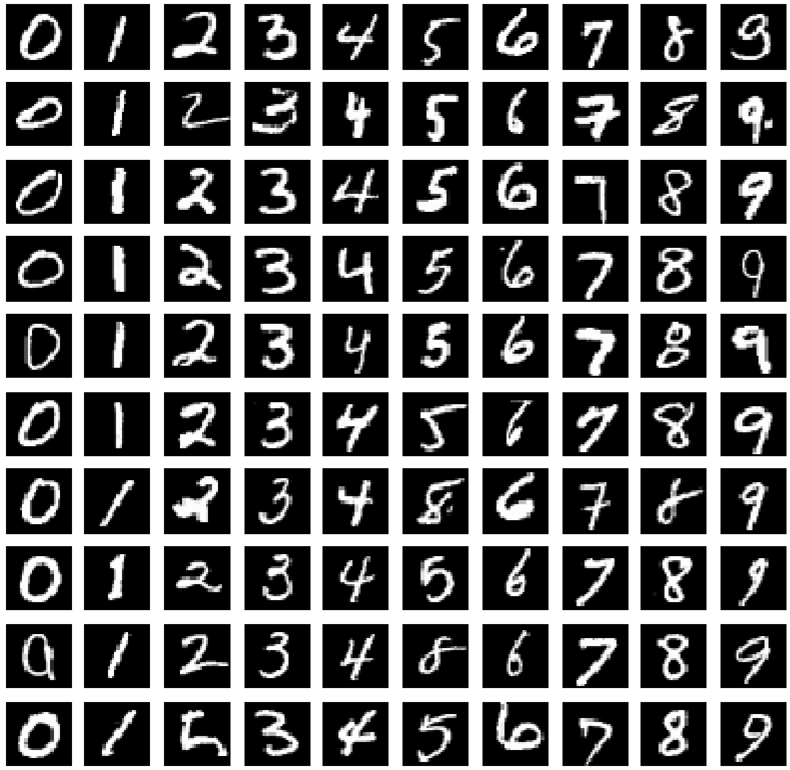
In [13]:
```
fig, axes = plt.subplots(2, 1)
for i, measure in enumerate(['loss', 'acc']):
    loss_figures = {'train':train_history['train_'+measure+'_hist'],
                    'val':train_history['val_'+measure+'_hist']}
    ax = axes[i]
    for loss_name, loss_history in list(loss_figures.items()):
        ax.plot(loss_history, label = loss_name, rasterized=True)
        ax.set_xlabel('# epoch', size=14)
        ax.legend(loc='upper center', ncol=2)
        if measure == 'loss':
            ax.set_ylabel('Loss', size=14)
        else:
            ax.set_ylabel('Accuracy', size=14)
        ax.xaxis.set_ticks(np.linspace(1, len(loss_history), 10).astype(int))
        ax.xaxis.set_ticklabels(np.linspace(1, 10, 10).astype(int))

plt.gcf().set_size_inches(15, 10)
plt.subplots_adjust(hspace=0.5)
fig.suptitle('Convergence', size=20)
axes[0].title.set_text('Train and Validation loss during training')
axes[1].title.set_text('Train and Validation accuracy during training')
plt.show()
```



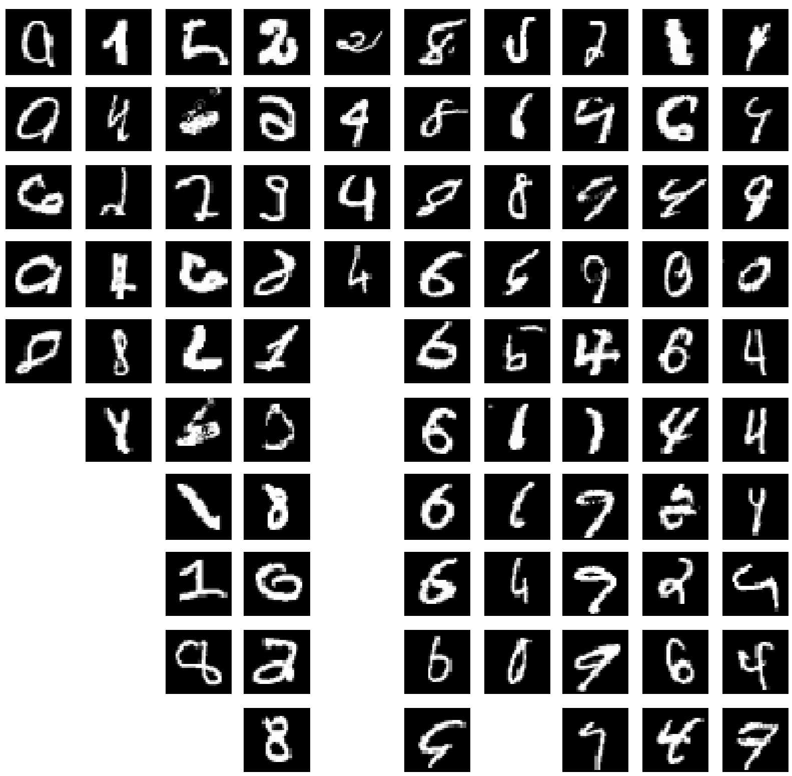
3. Visualizing Predictions for Validation Samples

In [14]:
```
categorized_data = {0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[]}
misclassified_data = {0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[]}

category_lens = 0
model.eval()  # change model mode to eval
with torch.no_grad():  # temporarily set all requires_grad flags to False
    for X, y in loader_val:
        # Move inputs to specified device
        X = X.to(device=device, dtype=torch.float32)
        y = y.to(device=device, dtype=torch.long)

        # Compute scores (Forward pass)
        scores = model(X)
        _, preds = scores.max(dim=1)

        # Convert tensor to numpy array
        data = X.cpu().numpy()
        preds = preds.cpu().numpy()
        y = y.cpu().numpy()

        # Fill dictionary of predictions
        if np.min(category_lens)<10:
            for num, category in enumerate(preds):
                categorized_data[category].append([data[num]])

        # Fill dictionary of incorrect predictions
        for num, category in enumerate(preds):
            if category != y[num]:
                misclassified_data[category].append([data[num]])

        # Break out of loop when we have 10 samples per category
        category_lens = np.array([len(samples) for category,samples in categorize
        misclassified_lens = np.array([len(samples) for category,samples in miscl
        if (np.min(category_lens)>10) and (np.min(misclassified_lens)>10):
            break

pltClasses(categorized_data, 'Randomly chosen model predictions for validation se
pltClasses(misclassified_data, 'Incorrect predictions for validation set')
```

Randomly chosen model predictions for validation set



Incorrect predictions for validation set



```
In [ ]:
```