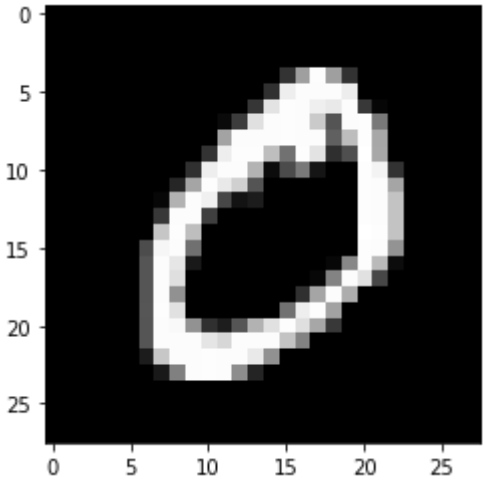
```
%matplotlib inline
from fastai.basics import *


from google.colab import drive
drive.mount('/content/drive')


path = '/content/drive/My Drive/mnist.pkl.gz'


with gzip.open(path, 'rb') as f:
    ((x_train, y_train), (x_valid, y_valid), _) = pickle.load(f, encoding='latin-1')


plt.imshow(x_train[1].reshape((28,28)), cmap="gray")
x_train.shape
```

```
(50000, 784)
```




```
x_train,y_train,x_valid,y_valid = map(torch.tensor, (x_train,y_train,x_valid,y_valid))
n,c = x_train.shape
x_train.shape, y_train.min(), y_train.max()
```

```
(torch.Size([50000, 784]), tensor(0), tensor(9))
```

```
bs=64
train_ds = TensorDataset(x_train, y_train)
valid_ds = TensorDataset(x_valid, y_valid)
data = DataBunch.create(train_ds, valid_ds, bs=bs)


x,y = next(iter(data.train_dl))
x.shape,y.shape
```

```
(torch.Size([64, 784]), torch.Size([64]))
```

```
class Mnist_Logistic(nn.Module):
    def __init__(self):
        super().__init__()
        self.lin = nn.Linear(784, 10, bias=True)
```

```
    def forward(self, xb): return self.lin(xb)
```

```
model = Mnist_Logistic().cuda()
```

```
model
```

```
Mnist_Logistic(
  (lin): Linear(in_features=784, out_features=10, bias=True)
)
```

```
model.lin
```

```
Linear(in_features=784, out_features=10, bias=True)
```

```
model(x).shape
```

```
torch.Size([64, 10])
```

```
[p.shape for p in model.parameters()]
```

```
[torch.Size([10, 784]), torch.Size([10])]
```

```
[p for p in model.parameters()]
```

```
[Parameter containing:
 tensor([[-0.0224,  0.0233, -0.0335,  ..., -0.0170,  0.0299,  0.0211],
         [-0.0217,  0.0193,  0.0330,  ...,  0.0268, -0.0142, -0.0356],
         [-0.0147,  0.0313,  0.0260,  ...,  0.0307, -0.0114,  0.0069],
         ...,
         [-0.0031,  0.0231,  0.0201,  ..., -0.0124, -0.0177,  0.0201],
         [ 0.0292,  0.0031,  0.0321,  ..., -0.0219,  0.0002, -0.0167],
         [-0.0156, -0.0226,  0.0337,  ..., -0.0225, -0.0301, -0.0275]],
        device='cuda:0', requires_grad=True), Parameter containing:
 tensor([-0.0040,  0.0310,  0.0130, -0.0303, -0.0130, -0.0085, -0.0290, -0.0061,
          0.0203,  0.0268], device='cuda:0', requires_grad=True)]
```

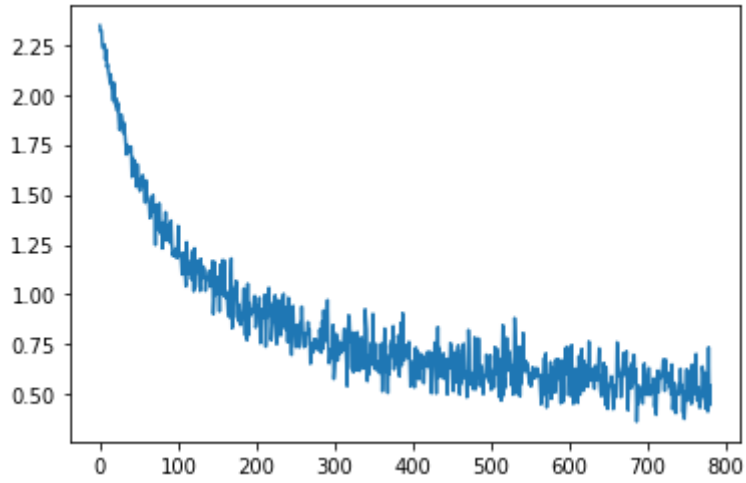
```
lr=2e-2
```

```
loss_func = nn.CrossEntropyLoss()
```

```
def update(x,y,lr):
    wd = 1e-5
    y_hat = model(x)
    w2 = 0.
    for p in model.parameters(): w2 += (p**2).sum()
    loss = loss_func(y_hat, y) + w2*wd
    loss.backward()
    with torch.no_grad():
        for p in model.parameters():
            p.sub_(lr * p.grad)
            p.grad.zero_()
    return loss.item()
```

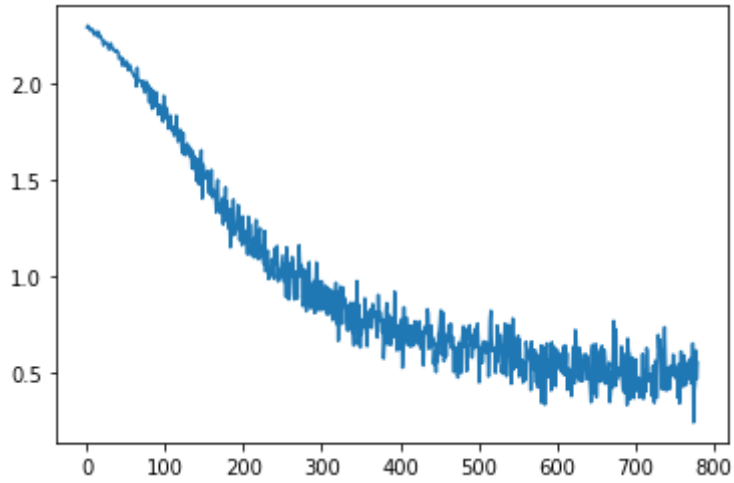
```
losses = [update(x,y,lr) for x,y in data.train_dl]

plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa2c551fcf8>]
```




```
class Mnist_NN(nn.Module):
    def __init__(self):
        super().__init__()
        self.lin1 = nn.Linear(784, 50, bias=True)
        self.lin2 = nn.Linear(50, 10, bias=True)

    def forward(self, xb):
        x = self.lin1(xb)
        x = F.relu(x)
        return self.lin2(x)

model = Mnist_NN().cuda()

losses = [update(x,y,lr) for x,y in data.train_dl]

plt.plot(losses)
```

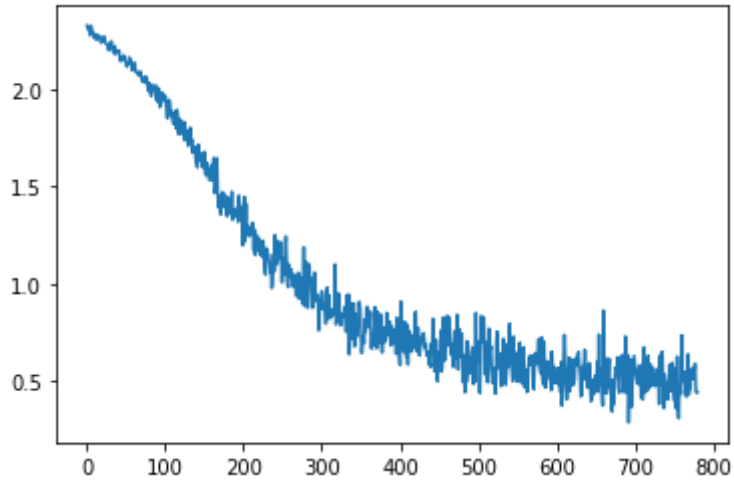
```
[<matplotlib.lines.Line2D at 0x7fa2c6668710>]
```




```
model = Mnist_NN().cuda()
```

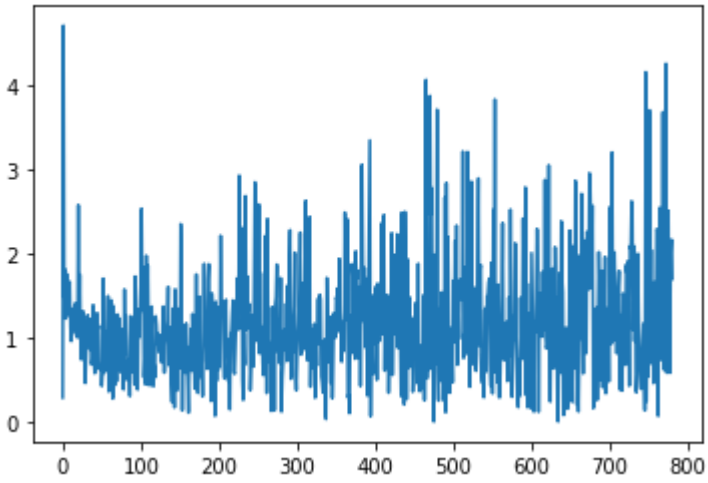
```
def update(x,y,lr):
    opt = optim.SGD(model.parameters(), lr)
    y_hat = model(x)
    loss = loss_func(y_hat, y)
    loss.backward()
    opt.step()
    opt.zero_grad()
    return loss.item()


losses = [update(x,y,lr) for x,y in data.train_dl]


plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa2c6632630>]
```



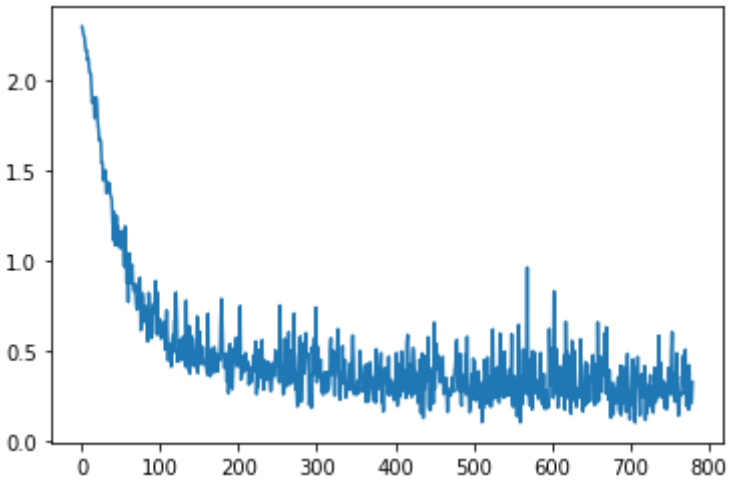

```
def update(x,y,lr):
    opt = optim.Adam(model.parameters(), lr)
    y_hat = model(x)
    loss = loss_func(y_hat, y)
    loss.backward()
    opt.step()
    opt.zero_grad()
    return loss.item()


losses = [update(x,y,lr) for x,y in data.train_dl]


plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa2c65e55c0>]
```




```
model = Mnist_NN().cuda()

def update(x,y,lr):
    opt = optim.Adam(model.parameters(), lr)
    y_hat = model(x)
    loss = loss_func(y_hat, y)
    loss.backward()
    opt.step()
    opt.zero_grad()
    return loss.item()

losses = [update(x,y,1e-3) for x,y in data.train_dl]

plt.plot(losses)
```

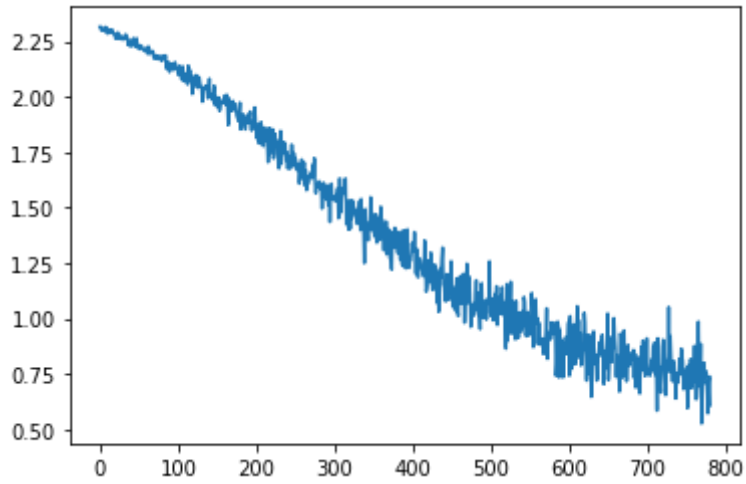
```
[<matplotlib.lines.Line2D at 0x7fa2c65b8400>]
```




```
model = Mnist_NN().cuda()

def update(x,y,lr):
    opt = optim.SGD(model.parameters(), lr, momentum=0.9)
    y_hat = model(x)
    loss = loss_func(y_hat, y)
```

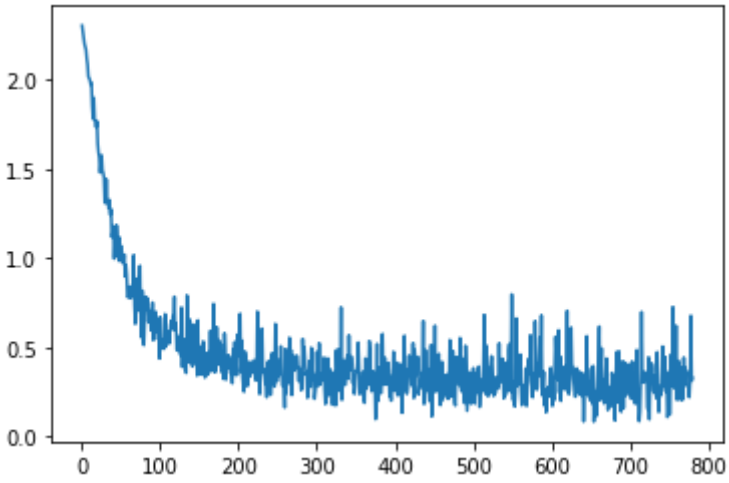
```
    loss.backward()
    opt.step()
    opt.zero_grad()
    return loss.item()


losses = [update(x,y,1e-2) for x,y in data.train_dl]


plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa2c144af98>]
```




```
model = Mnist_NN().cuda()


def update(x,y,lr):
    opt = optim.RMSprop(model.parameters(), lr)
    y_hat = model(x)
    loss = loss_func(y_hat, y)
    loss.backward()
    opt.step()
    opt.zero_grad()
    return loss.item()


losses = [update(x,y,1e-4) for x,y in data.train_dl]


plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa2c18dec18>]
```



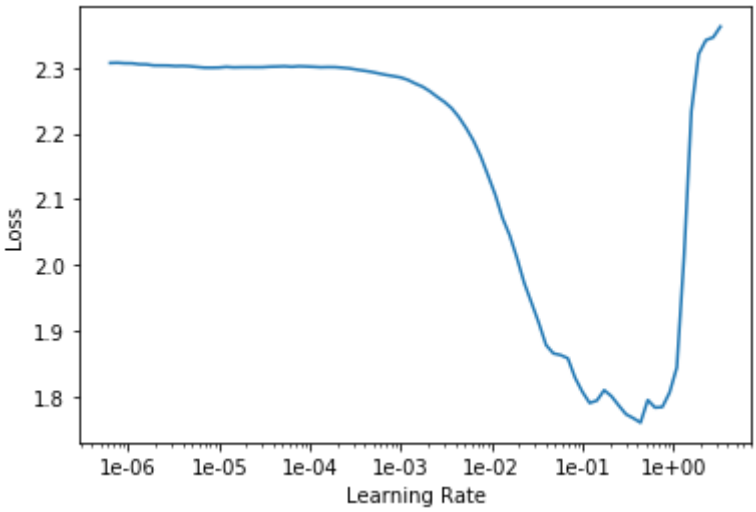

```
learn = Learner(data, Mnist_NN(), loss_func=loss_func, metrics=error_rate)

learn.lr_find()
learn.recorder.plot()
```

0.00% [0/1 00:00<00:00]

epoch	train_loss	valid_loss	error_rate	time

10.37% [81/781 00:00<00:03 1.7862]

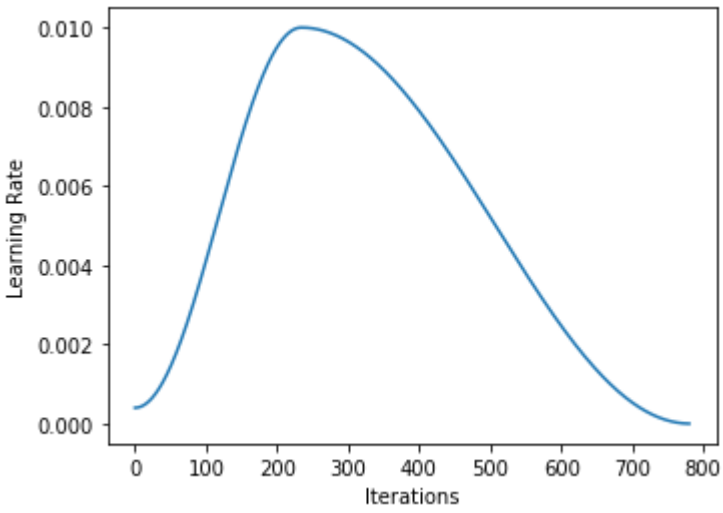
```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```

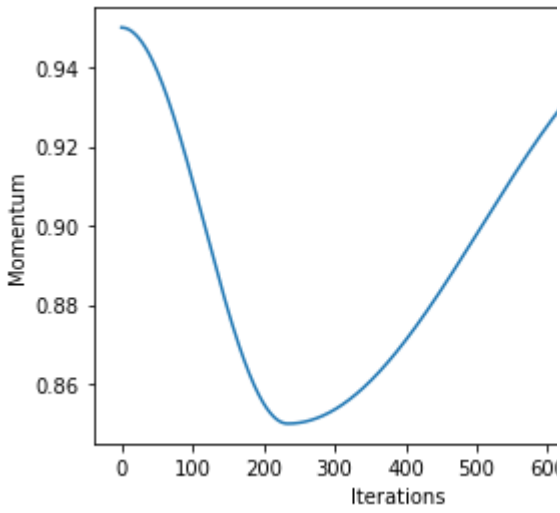



```
learn.fit_one_cycle(1, 1e-2)
```

epoch	train_loss	valid_loss	error_rate	time
0	0.152018	0.127762	0.037700	00:03

```
learn.recorder.plot_lr(show_moms=True)
```






```
learn.recorder.plot_losses()
```