```
pip install fastbook
```

```
from fastai.vision.all import *
from fastbook import *
matplotlib.rc('image', cmap='Greys')
```

DataSet MNIST

contains data type -> handwritten images of number .png

```
path = untar_data(URLs.MNIST_SAMPLE)
```

MNIST dataset follows a common layout for machine learning datasets: separate folders for the training set and the validation set (and/or test set)

```
Path.BASE_PATH=path
path.ls()
```

```
(#3) [Path('valid'),Path('train'),Path('labels.csv')]
```

```
#inside the training set
print((path/'train').ls())
```

```
[Path('train/7'), Path('train/3')]
```

Notes

- 3s and 7s folders - in ML we say
- '3' and '7' are labels/ targets

```
#in folders
three=(path/'train'/'3').ls().sorted() #return lists of asked dataset [.png]
seven=(path/'train'/'7').ls().sorted()
print("Length of 3s, 7s : ",len(three),len(seven))
```

```
Length of 3s, 7s :  6131 6265
```

HOW DATA LOOKS

Notes

- Image class from PIL Python Image Library used

```
image3_path=three[1]
image3=Image.open(image3_path)
image3
```

Represent in an array **Notes**

- Array (numpy) works on cpu
- tensor works on gpu
- So preferably use tensor for deep learning tasks

```
tensor(image3)[4:10,4:10]
```

```
tensor([[  0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,  29],
        [  0,   0,   0,  48, 166, 224],
        [  0,  93, 244, 249, 253, 187],
        [  0, 107, 253, 253, 230,  48],
        [  0,   3,  20,  20,  15,   0]], dtype=torch.uint8)
```

Notes

- *The 4:10 indicates we requested the rows from index 4 (included) to 10 (not included) and the same for the columns. NumPy indexes from top to bottom and left to right, so this section is located in the top-left corner of the image.*
- *We can slice the array to pick just the part with the top of the digit in it, and then use a Pandas DataFrame to color-code the values using a gradient, which shows us clearly how the image is created from the pixel values:*

```
image3_tensor=tensor(image3)
df=pd.DataFrame(image3_tensor[4:50,4:22])
df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	29	150	195	254	255	254	176	193	150	96	0	0
2	0	0	0	48	166	224	253	253	234	196	253	253	253	253	233	0	0
3	0	93	244	249	253	187	46	10	8	4	10	194	253	253	233	0	0
4	0	107	253	253	230	48	0	0	0	0	0	192	253	253	156	0	0
5	0	3	20	20	15	0	0	0	0	0	43	224	253	245	74	0	0
6	0	0	0	0	0	0	0	0	0	0	249	253	245	126	0	0	0
7	0	0	0	0	0	0	0	14	101	223	253	248	124	0	0	0	0
8	0	0	0	0	0	11	166	239	253	253	253	187	30	0	0	0	0
9	0	0	0	0	0	16	248	250	253	253	253	253	232	213	111	2	0
10	0	0	0	0	0	0	0	43	98	98	208	253	253	253	253	187	22
11	0	0	0	0	0	0	0	0	0	0	9	51	119	253	253	253	76
12	0	0	0	0	0	0	0	0	0	0	0	0	1	183	253	253	139
13	0	0	0	0	0	0	0	0	0	0	0	0	0	182	253	253	104
14	0	0	0	0	0	0	0	0	0	0	0	0	85	249	253	253	36
15	0	0	0	0	0	0	0	0	0	0	0	60	214	253	253	173	11
16	0	0	0	0	0	0	0	0	0	0	98	247	253	253	226	9	0
17	0	0	0	0	0	0	0	0	42	150	252	253	253	233	53	0	0
18	0	0	42	115	42	60	115	159	240	253	253	250	175	25	0	0	0
19	0	0	187	253	253	253	253	253	253	253	197	86	0	0	0	0	0
20	0	0	103	253	253	253	253	253	232	67	1	0	0	0	0	0	0

Color Codes for Better Understanding

```
df.style.set_properties(**{'font-size':'6pt'}).background_gradient('Greys')
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	29	150	195	254	255	254	176	193	150	96	0	0	0
2	0	0	0	48	166	224	253	253	234	196	253	253	253	253	233	0	0	0
3	0	93	244	249	253	187	46	10	8	4	10	194	253	253	233	0	0	0
4	0	107	253	253	230	48	0	0	0	0	0	192	253	253	156	0	0	0
5	0	3	20	20	15	0	0	0	0	0	43	224	253	245	74	0	0	0
6	0	0	0	0	0	0	0	0	0	0	249	253	245	126	0	0	0	0
7	0	0	0	0	0	0	0	14	101	223	253	248	124	0	0	0	0	0
8	0	0	0	0	0	11	166	239	253	253	253	187	30	0	0	0	0	0
9	0	0	0	0	0	16	248	250	253	253	253	253	232	213	111	2	0	0
10	0	0	0	0	0	0	0	43	98	98	208	253	253	253	253	187	22	0
11	0	0	0	0	0	0	0	0	0	0	9	51	119	253	253	253	76	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	183	253	253	139	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	182	253	253	104	0
14	0	0	0	0	0	0	0	0	0	0	0	0	85	249	253	253	36	0
15	0	0	0	0	0	0	0	0	0	0	0	60	214	253	253	173	11	0
16	0	0	0	0	0	0	0	0	0	0	98	247	253	253	226	9	0	0
17	0	0	0	0	0	0	0	0	42	150	252	253	253	233	53	0	0	0
18	0	0	42	115	42	60	115	159	240	253	253	250	175	25	0	0	0	0

SEVEN

21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
image7_path=seven[0]
image7=Image.open(image7_path)
image7_tensor=tensor(image7)
df=pd.DataFrame(image7_tensor[4:50,4:23])
df.style.set_properties(**{'font-size':'6pt'}).background_gradient('Greys')
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	21	51	213	254	252	252	252	254	252	252	252	254	252	252	252	255	252	100	0
4	161	250	250	252	250	250	250	252	250	250	250	252	250	250	250	252	250	100	0
5	250	250	250	252	189	190	250	252	250	250	250	252	250	250	250	252	189	40	0
6	130	250	250	49	29	30	49	49	49	49	49	49	49	170	250	252	149	0	0
7	0	0	0	0	0	0	0	0	0	0	0	11	132	252	252	244	121	0	0
8	0	0	0	0	0	0	0	0	0	0	0	51	250	250	250	202	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	172	250	250	250	80	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	252	250	250	250	0	0	0	0
11	0	0	0	0	0	0	0	0	0	31	213	254	252	252	49	0	0	0	0
12	0	0	0	0	0	0	0	0	0	151	250	252	250	250	49	0	0	0	0
13	0	0	0	0	0	0	0	0	0	151	250	252	250	159	20	0	0	0	0
14	0	0	0	0	0	0	0	0	0	151	250	252	250	100	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	152	252	254	252	100	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	151	250	252	250	100	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	151	250	252	250	100	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	151	250	252	250	221	40	0	0	0	0
19	0	0	0	0	0	0	0	0	0	153	252	255	252	252	49	0	0	0	0
20	0	0	0	0	0	0	0	0	0	151	250	252	250	250	49	0	0	0	0

First Method: Pixel Similarity

find the average pixel value for every pixel of the 3s, then do the same for the 7s. This will give us two group averages, defining what we might call the "ideal" 3 and 7. Then, to classify an image as one digit or the other, we see which of these two ideal digits the image is most similar to

[ ] ↳ *52 cells hidden*

Second Method : Wriitng Loss Funtion

```
three_tensorsx=[tensor(Image.open(img)) for img in three]
```

```
three_tensorsx=[tensor(Image.open(img)) for img in three]
seven_tensorsx=[tensor(Image.open(img)) for img in seven]
len(three_tensorsx),len(seven_tensorsx)
```

```
(6131, 6265)
```

```
show_image(three_tensorsx[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff16d3c6510>
```

```
stacked_threex=torch.stack(three_tensorsx).float()/255
stacked_sevenx=torch.stack(seven_tensorsx).float()/255
stacked_threex.shape
```

```
torch.Size([6131, 28, 28])
```

```
show_image(stacked_threex[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff1710e2b90>
```

Notes :

- Concatenating Both Stacked Lists
- into a single vecotor (3rank to 2rank tensor)
- view method -> transforms

```
train_x=torch.cat([stacked_threex,stacked_sevenx]).view(-1,28*28)
train_x
```

```
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]])
```

Lebel Selection

- 1 -> 3s
- 0 -> 7s

-> method unsqueeze will add a colum -> making it a 2d array / matrix

```
train_y=tensor([1]*len(three) + [0]*len(seven)).unsqueeze(1)
train_x.shape,train_y.shape
```

```
(torch.Size([12396, 784]), torch.Size([12396, 1]))
```

Making A DataSet

- Tuple (independent , dependent)
- 1 image , 1 label

```
train_dset= list(zip(train_x,train_y))
```

```
x,y=train_dset[0]#returns a tuple
x.shape,y
```

```
(torch.Size([784]), tensor([1]))
```

For Validation Set

```
valid_x=torch.cat([valid_three_tensors,valid_seven_tensors]).view(-1,28*28)
valid_y=tensor( [1]*len(valid_three_tensors) + [0]*len(valid_seven_tensors) ).unsqueez
valid_dset=list(zip(valid_x,valid_y))
```

Notes

- Now I have train dataset set with labelled data (img,label)
- And validation dataset with labelled data
- all in 2 vectors [2 rank vectors]

Step :1 Initialize with Random Weights

```
def initialize_params(size,std=1.0):
```

```
  return (torch.randn(size)*std).requires_grad_()
```

```
weights=initialize_params(28*28,1)
weights[0]
```

```
tensor(-0.0022, grad_fn=<SelectBackward0>)
```

Notes

- cant just use weights*pixels for weight formula
- as if pixel is 0 so weight become zero
- **so we using y=w*x+b - this c here helps not to be zero**
- w-> weight
- b-> bias

```
bias=initialize_params(1)
bias
```

```
tensor([0.5515], requires_grad=True)
```

Calculate Prediction for One Image

```
(train_x[0]*weights.T).sum()+bias # T -> Transpose to rose and cols
```

```
tensor([-11.9296], grad_fn=<AddBackward0>)
```

Use - > Matrix multiplication saves gpu and computation below method not up

```
def linear1(xb):
  return xb@weights + bias
```

```
preds=linear1(train_x)
preds
```

```
tensor([-11.9296, -17.2066, -11.8965,  ...,  11.3228,  -8.6819,  -8.0114],
grad_fn=<AddBackward0>)
```

Accuracy

```
corrects= (preds>0.0).float()==train_y
```

```
corrects
```

```
tensor([[False, False, False,  ...,  True, False, False],
        [False, False, False,  ...,  True, False, False],
        [False, False, False,  ...,  True, False, False],
        ...,
        [ True,  True,  True,  ..., False,  True,  True],
        [ True,  True,  True,  ..., False,  True,  True],
        [ True,  True,  True,  ..., False,  True,  True]])
```

```
corrects.float().mean().item() # item uncovers value from 1-rank tensor
```

```
0.5037140250205994
```

*A loss funtion *

- measures distance between predictions and targets
- not has zero gradient problem
- **Problem : only works when prediction betwwen 0,1**
- **Solution: Using Sigmoid Function**

```
def mnist_loss(pred,targets):
  pred=pred.sigmoid()
  return torch.where(targets==1,1-pred,pred).mean()
```

SDG AND MINI BATCHES

```
# PRACTICE MINI BATCH CREATION
coll = range(20)
dl=DataLoader(coll,batch_size=5,shuffle=True)
list(dl)
```

```
[tensor([ 1,  4,  5, 18,  9]),
 tensor([ 2, 11,  6, 19, 12]),
 tensor([10,  7, 16, 15, 13]),
 tensor([17,  3,  0, 14,  8])]
```

```
ds=L(enumerate(string.ascii_lowercase))
ds
```

```
(#26) [(0, 'a'),(1, 'b'),(2, 'c'),(3, 'd'),(4, 'e'),(5, 'f'),(6, 'g'),(7, 'h'),
(8, 'i'),(9, 'j')...]
```

Notes

- Creating Tuples - Data, Labels

```
dl=DataLoader(ds,batch_size=6,shuffle=True)
list(dl)
```

```
[(tensor([ 1, 23,  9,  8, 24,  2]), ('b', 'x', 'j', 'i', 'y', 'c')),
 (tensor([14, 25, 13, 11, 19,  5]), ('o', 'z', 'n', 'l', 't', 'f')),
 (tensor([ 0, 10,  4,  7, 18, 12]), ('a', 'k', 'e', 'h', 's', 'm')),
 (tensor([ 6, 21, 15, 16, 22,  3]), ('g', 'v', 'p', 'q', 'w', 'd')),
 (tensor([20, 17]), ('u', 'r'))]
```

Writing Proper Training Loop for SDG

```
weights=initialize_params((28*28,1))
```

```
bias=initialize_params(1)
```

```
dl=DataLoader(train_dset,batch_size=256)
xb,yb=first(dl) #first - describes first thing in arbitrator
xb.shape,yb.shape
```

```
(torch.Size([256, 784]), torch.Size([256, 1]))
```

```
valid_dl=DataLoader(valid_dset,batch_size=256)
```

creating mini batch of size 4

```
minibatch=train_x[:4]
minibatch.shape
```

```
torch.Size([4, 784])
```

```
preds=linear1(minibatch)
preds
```

```
tensor([[ 2.3260],
        [ 0.7517],
        [-0.0286],
        [ 4.0277]], grad_fn=<AddBackward0>)
```

```
loss=mnist_loss(preds,train_y[:4])
```

```
loss
```

```
tensor(0.2335, grad_fn=<MeanBackward0>)
```

calculation gradients

```
loss.backward() # calculates gradients and adds to existing gradeingts
```

```
weights.grad.shape,weights.grad.mean(),bias.grad
```

```
(torch.Size([784, 1]), tensor(-0.0204), tensor([-0.1415]))
```

```
#writing funtion for it
def cal_gradient(xb,yb,model):
  preds=model(xb)
  loss=mnist_loss(preds,yb)
  loss.backward()
```

testing

```
cal_gradient(minibatch,train_y[:4],linear1)
weights.grad.mean(),bias.grad
```

```
(tensor(-0.0407), tensor([-0.2830]))
```

```
cal_gradient(minibatch,train_y[:4],linear1)
weights.grad.mean(),bias.grad
```

```
(tensor(-0.0611), tensor([-0.4245]))
```

```
cal_gradient(minibatch,train_y[:4],linear1)
weights.grad.mean(),bias.grad
```

```
(tensor(-0.0815), tensor([-0.5660]))
```

```
#gradients updated to we have to set them to zero
weights.grad.zero_()
bias.grad.zero_()
```

```
tensor([0.])
```

TRAINING LOOP

```
def train_epoch(model) :
  for xb,yb in dl:
    cal_gradient(xb,yb,model)
    opt.update()
    opt.zero_grad()

def batch_accuracy(xb,yb):
  preds=xb.sigmoid()
  correct=(preds>0.5)==yb
  return correct.float().mean()
```

```
batch_accuracy(linear1(minibatch),train_y[:4])
```

```
tensor(0.7500)
```

Doing For Every Batch in Validation Set

```
def validate_epoch(model):
  accs=[batch_accuracy(model(xb),yb) for xb,yb in valid_dl]
  return round(torch.stack(accs).mean().item(),4)
```

```
validate_epoch(linear1)
```

```
0.3896
```

Train for 1 epoch

```
lr=1
params=weights,bias
train_epoch(linear1,lr,params)
validate_epoch(step)
```

Loop

```
for i in range(10):
  train_epoch(linear1,lr,params)
  print(validate_epoch(step))
```

```
0.977
0.978
0.9789
0.9789
0.9789
0.9799
```

```
0.9804
0.9804
0.9804
0.9804
```

Creating one optimizer

```
linear1??
```

Notes

- Using nn.Linear pytorch
- does same work as
- our initilizer() and step()
- nn.Linear contains both weights and bias in a single class

```
linear_model=nn.Linear(28*28,1)
```

```
w,b=linear_model.parameters()
w.shape,b.shape
```

```
(torch.Size([1, 784]), torch.Size([1]))
```

```
class BasicOptimizer:
  def __init__(self,params,lr):
    self.params,self.lr=list(params),lr

  def update(self,*args,**kwargs):
    for p in self.params:
      p.data = p.data-p.grad.data * self.lr
      # p.data -= p.grad.data * self.lr

  def zero_grad(self,*args,**kwargs):
    for p in self.params: p.grad = None
```

```
opt = BasicOptimizer(linear_model.parameters(),lr)
```

```
def train_model(model,epochs):
  for i in range(epochs):
    train_epoch(model)
    print(validate_epoch(model))
```

```
optx = BasicOptimizer(linear_model.parameters(),lr)


def train_modelx(model,epochs):
  for i in range(epochs):
    train_epochx(model)
    print(validate_epoch(model))


def train_epochx(model) :
  for xb,yb in dl:
    cal_gradient(xb,yb,model)
    optx.update()
    optx.zero_grad()
```

Testing

```
train_model(linear_model,20)
```

```
0.4932
0.8193
0.8471
0.915
0.934
0.9487
0.956
0.9628
0.9658
0.9677
0.9692
0.9716
0.9731
0.9751
0.9755
0.9765
0.9775
0.978
0.9785
0.9785
```

WE ALSO NOT NEED TO WRITE CLASS OPTIMIZER

- Fastai method -> SDG

```
linear_model=nn.Linear(28*28,1)
opt=SGD(linear_model.parameters(),lr)
train_modelx(linear_model,20)
```

```
pip install pytorch
```

```
dls= DataLoaders(dl,valid_dl)
```

```
learn=Learner(dls,nn.Linear(28*28,1),opt_func=SGD,loss_func=mnist_loss,metrics=batch_a
```

```
learn.fit(15,lr=lr)
```

epoch	train_loss	valid_loss	batch_accuracy	time
0	0.637089	0.503549	0.495584	00:00
1	0.555877	0.154397	0.878803	00:00
2	0.202395	0.194094	0.821394	00:00
3	0.087831	0.110828	0.909715	00:00
4	0.045691	0.079877	0.932287	00:00
5	0.029299	0.063613	0.946025	00:00
6	0.022605	0.053556	0.954858	00:00
7	0.019664	0.046893	0.962709	00:00
8	0.018199	0.042236	0.965653	00:00
9	0.017337	0.038823	0.967615	00:00
10	0.016740	0.036214	0.969087	00:00
11	0.016275	0.034146	0.971050	00:00
12	0.015888	0.032457	0.973013	00:00
13	0.015557	0.031047	0.974485	00:00
14	0.015272	0.029852	0.975466	00:00

Adding A Non Linearity

Notes

- Creating our NN
- Simply writing two linear funtions

```
def simple_NN(xb):
  res= xb@w1 + b1
  #implemetnt universal approximation funtion# turn all
  #-ives with zero- without this line it will be a stay a simple linear
  #called activation funtion
  # non- linearity
  res=res.max(tensor(0.0))
  res=res@w2+ b2
  return res
```

randomly initializing weigjts and biasness

```
w1=initialize_params(28*28,30)
b1=initialize_params(30)
w2=initialize_params(30,1)
b2=initialize_params(1)
```

- nn.Sequential creates a module which call each of layers or funtions in turn
- F.relu is a funtion module version - needed by nn.Sequesntial
-

```
simple_net= nn.Sequential(
    nn.Linear(28*28,30),
    nn.ReLU(),
    nn.Linear(30,1)
)
```

```
learn=Learner(dls,simple_net,opt_func=SGD,loss_func=mnist_loss,metrics=batch_accuracy)
```

```
learn.fit(20,0.1)
```

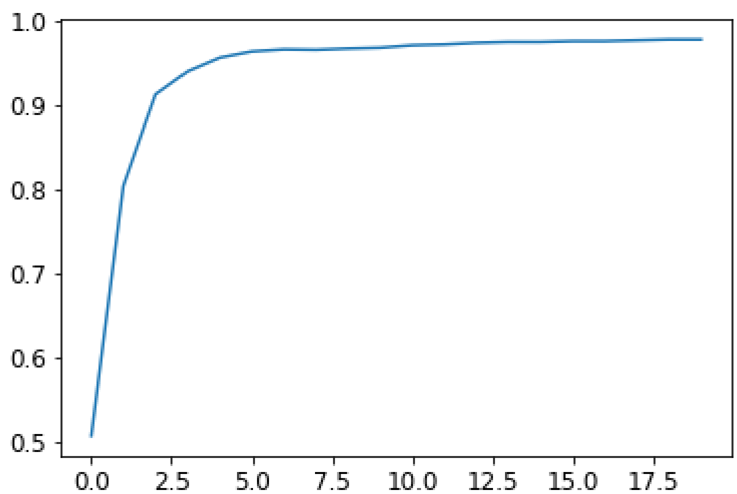
epoch	train_loss	valid_loss	batch_accuracy	time
0	0.309497	0.411356	0.506869	00:00
1	0.145424	0.229308	0.803729	00:00
2	0.081266	0.116420	0.912169	00:00
3	0.053824	0.078722	0.939647	00:00
4	0.041018	0.061564	0.955839	00:00
5	0.034424	0.051944	0.963199	00:00
6	0.030595	0.045890	0.965653	00:00
7	0.028080	0.041744	0.965162	00:00
8	0.026250	0.038714	0.966634	00:00
9	0.024821	0.036393	0.967615	00:00
10	0.023658	0.034541	0.970559	00:00
11	0.022685	0.033025	0.971541	00:00

How our Training Looks after fit

14	0.020517	0.029697	0.974485	00:00

```
plt.plot(L(learn.recorder.values).itemgot(2))
```

```
[<matplotlib.lines.Line2D at 0x7ff16cdebf50>]
```




```
learn.fine_tune(10,0.1)
```

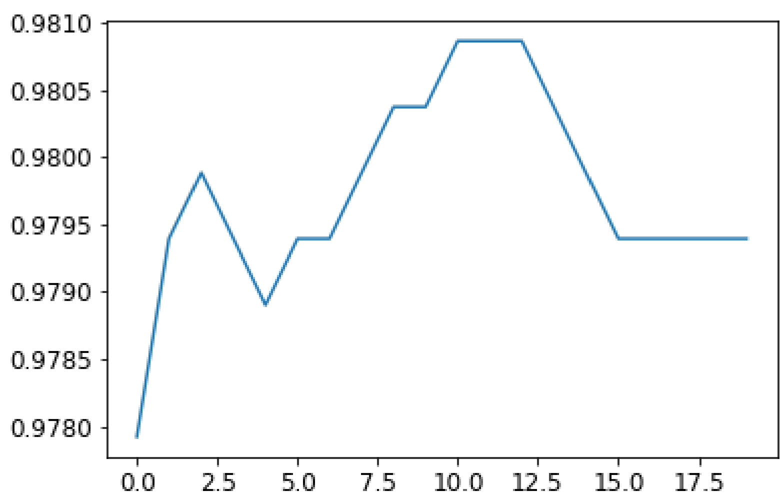
epoch	train_loss	valid_loss	batch_accuracy	time
0	0.018876	0.027572	0.976448	00:00

epoch	train_loss	valid_loss	batch_accuracy	time
0	0.020500	0.026418	0.976938	00:00
1	0.019711	0.025408	0.978410	00:00
2	0.019044	0.024806	0.979882	00:00
3	0.018673	0.024508	0.978410	00:00
4	0.018494	0.024339	0.978901	00:00
5	0.018441	0.024270	0.980373	00:00
6	0.018439	0.024310	0.979882	00:00
7	[illegible]	[illegible]	[illegible]	[illegible]

How Our Training Looks Like after fine tune

```
plt.plot(L(learn.recorder.values).itemgot(2))
```

```
[<matplotlib.lines.Line2D at 0x7ff16cf5c5d0>]
```



Due to excessive training accuracy dropped - overfitting condition wow

Final Accuracy

```
learn.recorder.values[-1][2]
```

```
0.9793915748596191
```

```
learn.model
```

```
Sequential(
  (0): Linear(in_features=784, out_features=30, bias=True)
  (1): ReLU()
  (2): Linear(in_features=30, out_features=1, bias=True)
)
```

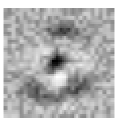
```
m=learn.model
w,b=m[0].parameters()
w[0].view(28,28)
```

```
tensor([[ 8.4111e-03,  3.3304e-02, -7.6458e-03,  1.2225e-02, -1.1078e-02,
-1.0224e-02, -9.1012e-03,  2.8117e-02, -3.2428e-02,  3.1413e-02,  7.3132e-03,
-2.8838e-02, -9.3962e-04, -1.4130e-02,
          8.5116e-03,  2.4588e-02,  8.3867e-03,  3.1530e-02,  1.7041e-02,
2.9750e-02,  1.0144e-02, -3.8949e-03,  1.3641e-02, -3.0581e-02,  9.5395e-03,
-1.6134e-02,  2.5186e-02,  3.0892e-02],
        [ 1.9285e-02,  2.8970e-02,  1.5249e-02, -1.4921e-02, -1.6701e-02,
1.8175e-02, -2.3327e-02, -9.3992e-03,  3.0889e-03,  1.3917e-02, -1.1852e-02,
-2.3780e-02, -1.1761e-02, -1.3369e-02,
         -7.1780e-03,  2.7980e-02,  7.0839e-03,  3.2441e-02,  1.5102e-02,
1.2221e-02, -3.5624e-02, -2.9662e-02,  1.3773e-02, -2.7223e-02,  1.8015e-02,
-1.8850e-02,  2.6321e-02, -2.6609e-03],
        [ 1.8536e-02,  7.4296e-04, -2.7726e-02,  2.8434e-02, -1.4306e-02,
2.1847e-02,  6.1863e-03, -8.9122e-03, -2.7433e-02,  1.8597e-02, -3.3840e-02,
3.2745e-02,  3.5639e-03,  3.1051e-02,
         -2.9791e-03, -4.8937e-03, -2.0237e-02, -2.9642e-02,  1.5716e-02,
-3.5538e-02,  2.4344e-02, -2.8577e-02,  3.7063e-03,  2.3488e-03,  2.6233e-02,
-2.7778e-02, -1.2733e-03, -1.9562e-02],
        [-1.7774e-02, -1.5739e-02, -9.1247e-03,  3.4774e-02, -7.2054e-04,
-5.3050e-03,  2.3231e-02,  1.0064e-02,  3.3946e-02,  5.1869e-03,  8.1435e-03,
3.1757e-02,  5.4580e-03, -2.3930e-03,
          4.0100e-02,  2.8005e-02,  3.8270e-02,  3.5021e-02, -1.4749e-02,
3.7014e-02, -8.6486e-03,  1.8845e-02, -1.1422e-03, -5.5164e-03, -7.2863e-03,
2.6802e-02,  1.6707e-02,  3.1098e-02],
        [ 2.7164e-02,  2.3788e-02, -2.2741e-02, -1.1017e-02,  1.9809e-02,
-1.4584e-02, -4.8811e-03, -2.5603e-02,  2.6815e-02, -6.7600e-03, -4.8739e-03,
1.1368e-02,  2.7654e-02,  3.5284e-02,
          5.7429e-02,  5.3861e-02,  4.0302e-02,  5.4914e-02, -1.1186e-02,
-3.6638e-03, -5.0248e-03,  2.5299e-02,  3.0410e-02,  1.8880e-02,  2.3433e-02,
-8.4125e-03, -2.2691e-02,  1.9615e-02],
        [ 1.5261e-02, -2.1302e-02, -2.5557e-02,  3.2784e-02,  1.5592e-02,
-1.1776e-02,  6.9683e-03,  1.9139e-02, -7.7244e-03,  1.1505e-02,  3.3937e-02,
7.9336e-02,  3.1054e-02,  5.7853e-02,
          1.0374e-01,  8.3106e-02,  1.0150e-01,  4.0097e-02,  5.9117e-02,
9.7890e-03,  3.8057e-02, -4.8346e-03,  5.2431e-03,  2.2737e-02, -2.0711e-02,
-7.5971e-03, -1.2095e-02,  1.9394e-02],
        [ 2.6046e-02,  3.5441e-02, -2.6356e-02,  1.6974e-02,  3.6421e-02,
-2.0327e-02, -2.6408e-02, -2.5062e-02,  3.1392e-02,  1.4916e-02, -1.3305e-02,
3.4333e-02,  7.7861e-02,  8.3313e-02,
          8.4014e-02,  9.6823e-02,  5.4469e-02,  4.3268e-02,  7.5027e-02,
2.5648e-02,  8.3297e-04,  3.5031e-03,  2.2287e-02,  1.6912e-02,  2.8993e-03,
```

```
-1.1278e-02, -1.5500e-02, -1.2082e-02],
        [-2.0424e-02, -1.6013e-03, -2.4120e-02, -2.0560e-04, -1.2666e-02,
2.8112e-02,  1.0534e-02,  8.5265e-03, -3.2864e-02,  1.6680e-03,  1.4991e-02,
-6.9410e-03,  4.1371e-02,  2.2614e-02,
          3.9483e-02, -1.4887e-03,  4.9206e-02,  4.5181e-02,  9.9295e-03,
2.0616e-02,  3.6073e-02,  5.0295e-02,  1.2882e-02, -2.5647e-02, -8.6326e-03,
2.8361e-02, -1.9754e-02, -1.2316e-02],
        [-3.2936e-02,  3.3486e-02,  2.6017e-02,  2.1669e-02, -3.7261e-02,
-9.8220e-03,  2.1863e-02,  9.5615e-03, -2.1744e-02, -1.3038e-02,  1.9122e-02,
1.5309e-02,  1.9344e-02, -2.2465e-02,
         -1.7685e-02, -2.0811e-02, -1.7803e-02,  1.7530e-02,  1.0280e-02,
1.0762e-02, -2.3090e-03,  4.1128e-02, -8.2738e-03, -1.7589e-02, -8.2207e-03,
2.8510e-02,  2.5453e-02,  2.4187e-02],
        [-2.6777e-02,  9.4454e-03, -3.3574e-02, -3.2809e-03,  2.8676e-02,
-2.7424e-03, -6.1350e-03,  1.5263e-02,  1.9119e-02, -3.7989e-02, -9.5632e-03,
1.0072e-03,  1.0839e-02,  1.0112e-02,
          9.7889e-03,  3.0017e-03,  1.6449e-02,  1.9433e-02,  2.8274e-02
```

```
show_image(w[0].view(28,28))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff16d04dd90>
```



Deeper Network - Using 18 layers Now

Notes

- More Performance
- Not Many params required
- Quick Training
- less memory
- See Register for detqiled explaination

```
dls=ImageDataLoaders.from_folder(path)
learn=cnn_learner(dls,resnet18,pretrained=False,
                  loss_func=F.cross_entropy,metrics=accuracy)
```

```
/usr/local/lib/python3.7/dist-packages/fastai/vision/learner.py:287: UserWarning
  warn("`cnn_learner` has been renamed to `vision_learner` -- please update your
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:136: UserWar
  f"Using {sequence_to_str(tuple(keyword_only_kwargs.keys()), separate_last='and
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserWar
  warnings.warn(msg)
```

```
learn.fit_one_cycle(1,0.1)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.096796	0.015008	0.996075	04:21

On 34 Layers ?

```
dls=ImageDataLoaders.from_folder(path)
learn=cnn_learner(dls,resnet34,pretrained=False,
                  loss_func=F.cross_entropy,metrics=accuracy)
```

```
/usr/local/lib/python3.7/dist-packages/fastai/vision/learner.py:287: UserWarning
  warn("`cnn_learner` has been renamed to `vision_learner` -- please update your
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:136: UserWarn
  f"Using {sequence_to_str(tuple(keyword_only_kwargs.keys()), separate_last='and
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserWarn
  warnings.warn(msg)
```

```
learn.fit_one_cycle(1,0.1)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.063652	0.021577	0.996075	06:02

```
kind of overfitting
```

kind of overfitting

✓ 6m 2s completed at 01:06