11.20.2018 ~ 12.07.2018.

# 2018 Pytorch FB Challenge

## Lesson 4

(very) Rough Note

by engiecat.

## What we'll learn.

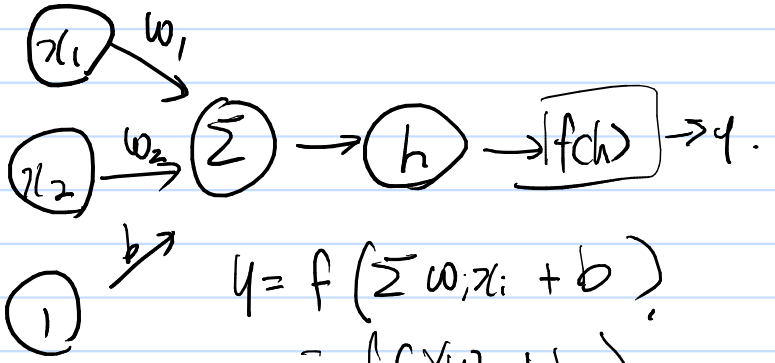
- Tensors - main ds.
- Autograds - calc. gradients
- Validations to check (methods)
- Transfer Learning

## Need

→ Notebook (on github)

→ PyTorch 0.4+

# ☆ Single Layer NNs.




$$y = f\left(\sum w_i x_i + b\right)$$
$$= f(XW + b)$$

$X: [\ldots]$ $W: \begin{bmatrix} \\ \vdots \\ \end{bmatrix}$

Tensors: generalized vectors. (3+D)

vectors → matrix → tensor.

☆ How to do it? ⇒ just do it like numpy!!!

e.g. torch.randn((size)), torch.exp.
rand_like(tensor)

quiz: [ activation( torch.dot(features, weights) + bias) ]

or maybe « ( (torch.mm(1)).sum() + bias. ]

strict — may have to change one?

ans: activation( (features * weights).sum() + bias))

↳ lenient.

↳ unexp. behav.

## NOTE

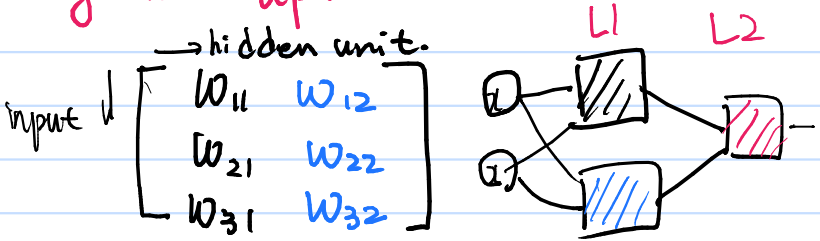
☆☆ weights.view : gives new tensor (shares data)

weights.resize_ : modifies tensor in-place

weights.reshape : usually works as view() but sometimes clone. (inefficient!!!)

◇ in this case, we can use transpose(), permute().

# Stacking Them up! (Vid 4-2)




quiz (calc L1) $[b_1\ b_2]$

L1 = activation ((features * W1 ) + bias1 )

L2 = activation (( L1 * W2) + bias 2 )

may use torch.mm (beter cuz ✗unexpected result)

## Vid 5

Numpy → torch : b = torch.from_numpy(a)

numpy ← torch : a = b.numpy()

* data is shared!

∴ support cross in-place operation.

# Vid 6,7: Neural Networks in PyTorch.

## MNIST example. (@torchvision)

↳ Task: identify the digits!

(torchvision.datasets.MNIST)

data → DataLoader (can set batch size, shuffle)

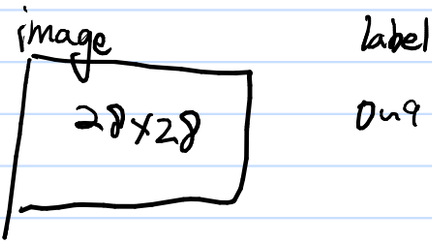
☆ Use How? for image, label in dataloader
blah blah blah.

☆ how to extract batch?

dataiter = iter(dataloader) ← contains trainset.

images, labels = dataiter.next()

(64, 1, 28, 28) (mini batch!) (64)




☆ Then, how to?

① Flatten image (28x28 ⇒ 784)

images_flat = (64, 784)

Why? we only learned dense net. (FC) ⇒ 1D data.

② Define output

⇒ 10 classes = 10 output units.

Quiz N layer

# Flatten

images.view(images.shape[0], -1) ← batch size, auto-set.

# init.

w1 = torch.randn(784, 256) ← Why? 2^n 승 조금 더 좋음

b1 = torch.randn(256) ← stricter, better!

h = activation(torch.mm(inputs, w1) + b1)

↳ REPEAT N times. (last should have (64, 10))

The output: 10 vals of one!

need to normalize by softmax.

⇒ For Proper Probability distribution.

Quiz

# softmax

torch.exp(x) / torch.sum(torch.exp(x), dim=1) ← sum across col.

64x10 64x64.

so we add .view(-1, 1) → 64x1

Vid 8 : Building NN with PyTorch (with Class)

☆ class Network(nn.Module) ☆

① __init__
- need super.init
- can add layer like

```
self.someLayer = nn.Linear(256, 10)
```

- can add activation like.

```
self.sigmoid = nn.Sigmoid()
```

② forward → where things are done

e.g.)
```
x = self.hidden(x)
x = self.softmax(x)
```

☆ Functional description available.

```
import torch.nn.functional as F
```

then apply it at forward

```
x = F.softmax(self.output(x), dim=1)
```

why? activation is fixed (no weights!)

Tip: use activation that are non-linear (e.g. ReLU)

# Vid 9 - How to train in PyTorch?

Universal Function Approximators.

= maps input to output. (e.g. MNIST classifier)

= deep learning approximates it.

① Loss Function = How bad is it?

( Mean Sq. Err, Cross entropy )

② Gradient Descent ⇒ ↓ Loss function.

How? Back prop. (go to Lesson 2!)

⇒ get $\frac{\partial L}{\partial w_{(s)}}$ or $\frac{\partial L}{\partial b_{(s)}}$ for grad. desc.

---

## How to in PyTorch?

Convention) L assigned to 'criterion'

+ in nn.CrossEntropyLoss

input is EXPECTED to be raw scores, not softmax normalized!

Tip: nn.Sequential ( List of layers ) are best for simple net.

### Code

```
model = nn.Sequential( ~~~ )
Criterion = nn.CrossEn~Loss()
logits = model(images)
loss = criterion(logits, labels)
```

Tip: nn.LogSoftMax is popular, convenient.

(real prob ⇒ torch.exp(output))

with this, use nn.NLLLoss

# Vid 10 Autograd.

How to do actual backprop in PyTorch?
autograd does it automatically!

To enable it (just to be sure)
`x.requires_grad_(True)`

To use it
`x.backward()` ⇒ calc. `x.grad`.

To disable it (∵ forwardprop only spd ↑)
`with torch.no_grad():`

Then, after autograd?
`loss.bacl` ()⇒ calc loss.grad
cby pa f network

* the loss ed into optimizers.

`optimizer = optim.SGD(model.parameters(), ...)`

each step [
`optimizer.zero_grad()` → loss accumulated!
calc loss - fwdprop, `loss = criterion(...)`
`loss.backward()`
`optimizer.step()` ⇒ changes params.
]

EPOCH: one pass of training set.

# Vid 12 - Fashion MNIST exercise.

⇒ My soln.
ReLU, n784 → 256 → 128 → 32 → 10 (NLLLoss)
Loss after 5 epochs: 0.296.

# Vid 14 - Inference and Validation

To prevent overfitting.

**Inference** → making predictions (feedforward)

**Validation** → for testing.
(aka test set) Not in training set, can check overfitting.

## How to do inference?

LogSoftmax → Softmax : `ps = torch.exp(out)`

① Find highest val class

`top_p, top_class = ps.topk(1, dim=1)`
(top-k; column)

② Get whether prediction == label

`equals = top_class == labels.view(*top_class.s`

64 × 1 → NOTEBOOK HAS ERROR!!

③ Get accuracy : div. by size of minibatch.

`acc = torch.mean(equals.type(torch.FloatTensor))`

## How to Get Rid of Overfit?

**Regularization e.g. Dropout.**

→ Recap) turn OFF some nodes!

use) `self.dropout = nn.Dropout(p=...)`

`x = self.dropout(F.relu(...))`

(Output layer : X Dropout!

when turning on/off dropout (e.g. eval)

`model.eval()` ↔ `model.train()`

# Vid 14 - Load / Save Models

PyTorch Network's param

⇒ Saved in state_dict. (in dictionary)

like model.state_dict.

(weight, bias)

To Save.

```
torch.save(model.state_dict, '~.pth')
```

To Load

```
state_dict = torch.load('~.pth')
model.load_state_dict(state_dict)
```

↳ must be created/init in advance

REQ. SAME ARCHITECTURE.

how to solve this? SAVE architecture (custom pth)

```
checkpoint = {'input_size': ~,
              'output_size': ~,
              'hidden_layers':
              [each.out_features for each in model.hidden_layers],
              'state_dict': model.state_dict()}
torch.save( ~, ~.pth)
```

list of the layers

no. of features

#loading.

```
checkpoint = torch.load(filepath)
model = Network(checkpoint['input_size'],
                ''  ['output_size'],
                ''  ['hidden_layers'])
model.load_state_dict(.. ['state_dict'])
```

custom defined.

# Vid 18 - Load Image data.

e.g. smartphone cam.

① torchvision.datasets.ImageFolder

dataset = datasets.ImF~('path', transform=~)

path : to file folder

e.g. root / dog / xxx.png
root / cat / yyy.png

② image → Tensor : what transform do!

transform : ensemble of processes
(e.g. crop, resize, toTensor)

transforms = transforms.Compose (
[tvs.Resize(255),
tvs.CenterCrop(224),
tvs.ToTensor(),])

③ Pass to DataLoader (generator)

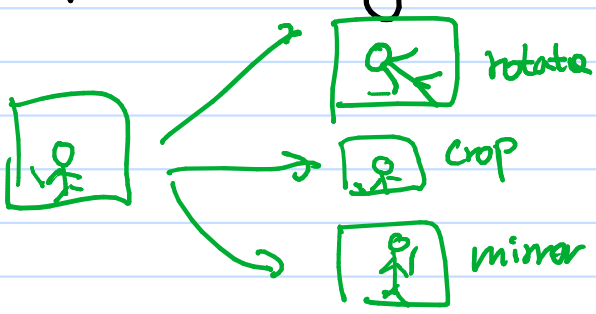
define batch size, shuffle,

dataloader = torch.utils.DataLoader
(dataset, batch_size=16, shuffle=True)

↳ generator ⇒ put it in forloop or
make it an iterator using iter()
(e.g. next(iter(data)))

# Data Augmentation.

→ Introduce 'Randomness' in data.
Help. network 'generalize' (↑perf.)




how?

train_tr = transforms.Compose([
tvs.RandomRotation(30),
tvs.RandomResizeCrop(100),
tvs.RandomHorizontalFlip(),
tvs.ToTensor
tvs.Normalize([0.5,0.5,0.5],
[0.5,0.5,0.5])
)]

in test_tr, no rotation, no flip.
just resize, center crop.
(as this is Validation)

no randomness (∵ consistent result)

# Vid 20 - Pre-trained Network

(Transfer Learning)

Ex. Dog Cat classifier.

ImageNet-trained net. (1M+ pics)
(torchvision.models)

e.g. VGG, AlexNet, Inception

size vs performance tradeoff.

Why it works?

→ such pretrained net is
very effective feature detectors.

∴ ImageNet trained info
TRANSFERED to our dataset.

## How to?

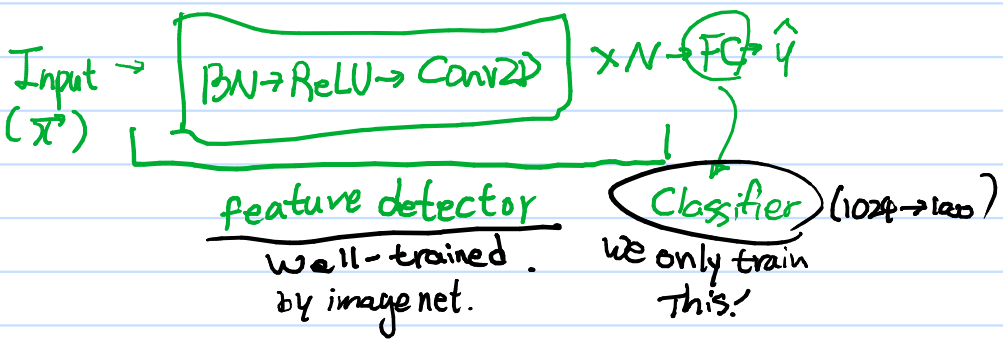
Note: most pretrained needs 224×224 pic.
& need to match color ch. distribution

ImageNet - Mean [0.485, 0.456, 0.406]
SD [0.229, 0.224, 0.225]

① Download model

```
from torchvision import models.
model = models.densenet121(pretrained=True)
```

② Look at structure!




③ Freeze feature detector part!

```
for param in model.parameters()
    param.requires_grad = False
```

① ×update params
② speed up the operation!

④ Build our own classifier.

```
classifier = nn.Sequential(...)
```

(e.g. FC(1024, 500) → ReLU
→ FC(500, 2) → ReLU → LogSoftmax)

⑤ Train only classifier.

use `optimizer = optim.Adam(model.classifier.parameters(), ...)`

(update only classifier)

## Tip. how to run in GPU?

① `model.cuda()` ↔ `model.cpu()`

*move other tensors USED by model too.!!!
(e.g. images, labels)

② `model.to('cuda')`.
vs. `model.to('cpu')`

check whether available by
`torch.cuda.is_available()`

## Tip. how to format train msg?

```
print(f"Test loss: {blah blah:.3f}.."
      f"..."
      ....)
```

## ★Tip. what you should do? (in loop)★

① forward prop
(don't forget `optimizer.zero_grad()`.)

② backprop
(don't forget to store running loss)

③ Validation (for some interval)

0. `model.eval()`
1. fwdprop (log-ps ⇒ ps)
2. find topclass (`ps.topk(1, dim=1)`)
3. calc. accuracy
`torch.mean(equality.type(...).item())`
convert to FloatTensor
4. calc. loss.
(running_loss / count.)
5. `model.train()`

④ Print CONCISELY.

### Watch those shapes

In general, you'll want to check that the tensors going through your model and other code are the correct shapes. Make use of the

```
.shape
```

method during debugging and development.

### A few things to check if your network isn't training appropriately

Make sure you're clearing the gradients in the training loop with

```
optimizer.zero_grad()
```

→ accumulating gradient.

. If you're doing a validation loop, be sure to set the network to evaluation mode with

```
model.eval()
```

→ dropout OFF

, then back to training mode with

```
model.train()
```

→ dropout ON

.

### CUDA errors

Sometimes you'll see this error:

```
RuntimeError: Expected object of type torch.FloatTensor but found type torch.cuda.FloatTensor
for argument #1 'mat1'
```

You'll notice the second type is

```
torch.cuda.FloatTensor
```

, this means it's a tensor that has been moved to the GPU. It's expecting a tensor with type

```
torch.FloatTensor
```

, no

```
.cuda
```

there, which means the tensor should be on the CPU. PyTorch can only perform operations on tensors that are on the same device, so either both CPU or both GPU. If you're trying to run your network on the GPU, check to make sure you've moved the model and all necessary tensors to the GPU with

```
.to(device)
```

where

```
device
```

is either

```
"cuda"
```

or

```
"cpu"
```