Lesson 4

fastai/fastbook

Create dataset

```
train_x = torch.cat([stacked_threes, stacked_sevens]).view(-1, 28*28)
train_y = tensor([1] * len(threes) + [0] * len(sevens)).unsqueeze(1)
print(train_x.shape, train_y.shape)
CONSOLE: (torch.Size([12396, 784]), torch.Size([12396, 1]))
dset = list(zip(train_x, train_y))
x, y = dset[0]
print(x.shape, y)
PRINT: (torch.Size([784]), tensor([1]))
```

Create weights

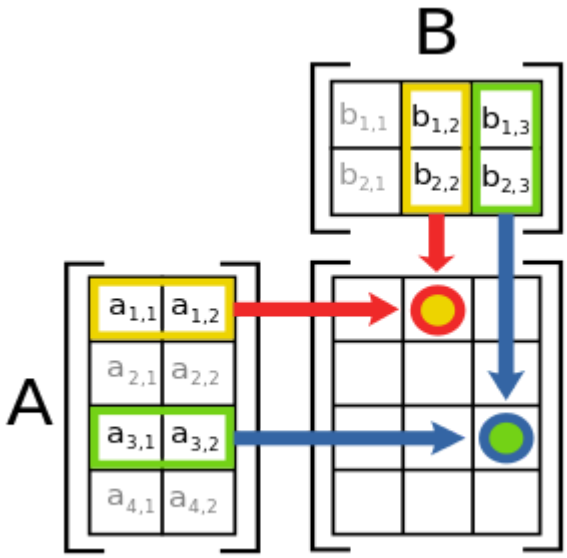
```
def init_params(size, variance=1.0):
    return (torch.randn(size)*variance).requires_grad_()

weights = init_params((28*28,1))
bias = init_params(1)
```

`requires_grad` tells that we want to gradients for these weights. `_` in the end means that it's in place operation.

`weights * pixel` is always zero if pixels are zero and that is why we need to add some number called `bias`

Matrix Multiplication



The first (red and yellow) circle is calculated:

$a_{1,1} \cdot b_{1,2} + a_{1,2} \cdot b_{2,2}$

In Python `@` represent matrix multiplication. `A@B`

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

threshold = 0.0
accuracy = ((preds > threshold).float() == train_y).float().mean().item()
```

If we change one weight a little bit it doesn't necessary affect the accuracy.

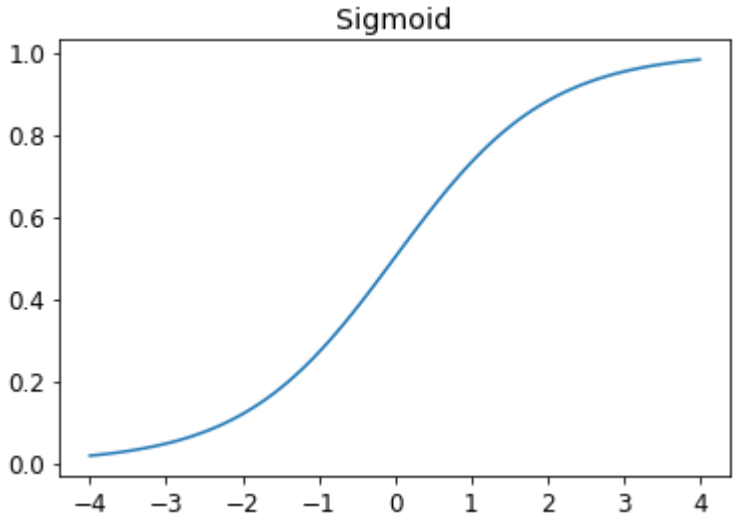
```
weights[0] *= 1.0001
preds = linear1(train_x)
accuracy2 = ((preds>0.0).float() == train_y).float().mean().item()
```

Now `accuracy == accuracy2`

This shows why accuracy is not a good loss function because small changes doesn't necessary affect it.

```
def sigmoid(x) : return 1 / (1 + torch.exp(-x))
```

Sigmoid function changes any numbers into numbers between zero and one.



Sigmoid is used in cases where the real values are between zero and one. It doesn't make sense that the model can predict something really far from that range.

Sigmoid can be modified to match any range. For example in movie review model the predictions might be between one and five. The function in that case `(5 - 1) * sigmoid(output) + 1`

So far we have seen gradient descent but what we look the next is called stochastic gradient descent. It is the same thing except done in batches.

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

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

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

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

This is stochastic gradient descent SGD because it's using batches.

```
def calc_grad(xb, yb, model):
    preds = model(xb)
    loss = mnist_loss(preds, yb)
    loss.backward()
```

```
def train_epoch(model, lr, params):
    for xb,yb in dl:
        calc_grad(xb, yb, model)
        for p in params:
            p.data -= p.grad*lr
            p.grad.zero_()
```

Using PyTorch's `nn.Linear`

```
linear_model = nn.Linear(28*28, 1)
# nn.Linear(num of weights, num of biases)

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

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

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

opt = BasicOptim(linear_model.parameters(), lr)

def train_epoch(model):
    for xb,yb in dl:
        calc_grad(xb, yb, model)
        opt.step()
        opt.zero_grad()

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

train_model(linear_model, 10)
```

Using fastai's `SGD`

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

def train_epoch(model):
    for xb,yb in dl:
        calc_grad(xb, yb, model)
        opt.step()
        opt.zero_grad()

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

train_model(linear_model, 10)
```

Using fastai's `Learner`

```
# Previously we used DataLoader not DataLoader**s**
dls = DataLoaders(dl, valid_dl)
learn = Learner(dls, nn.Linear(28*28,1), opt_func=SGD,
                loss_func=mnist_loss, metrics=batch_accuracy)
learn.fit(10)
```

Learner = `optimizer` + `train_model` + `train_epoch`

Optimizer = `initialize_parameters` + `step` + `zero_gradients`

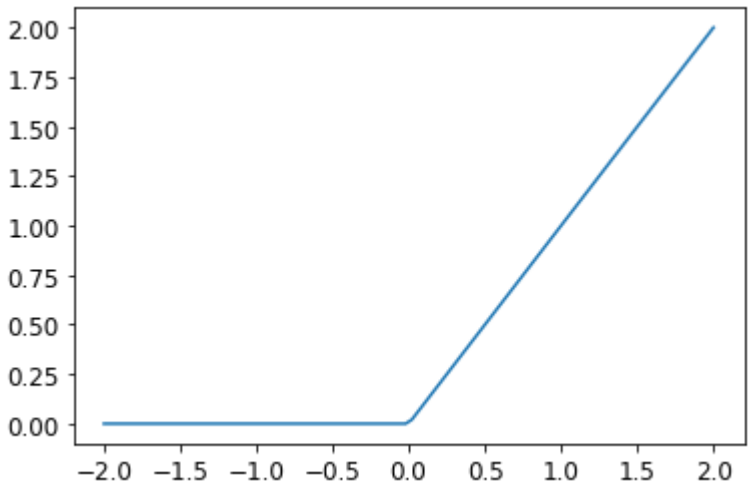
Train Model = for loop around `train_epoch`

Train Epoch = take one batch at a time and calculate gradient for that batch and take a step

---

Right now our model has been a linear function. To turn it into neural network it needs another linear function and non-linear function between the two linear functions.

```
def simple_net(xb):
    res = xb@w1 + b1
    res = res.max(tensor(0.0))
    res = res@w2 + b2
    return res
```



`simple_net` uses ReLU non-linear model

```
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(40, 0.1)
```

---

activations = numbers that are calculated

parameters = numbers that are randomly initialized and optimized

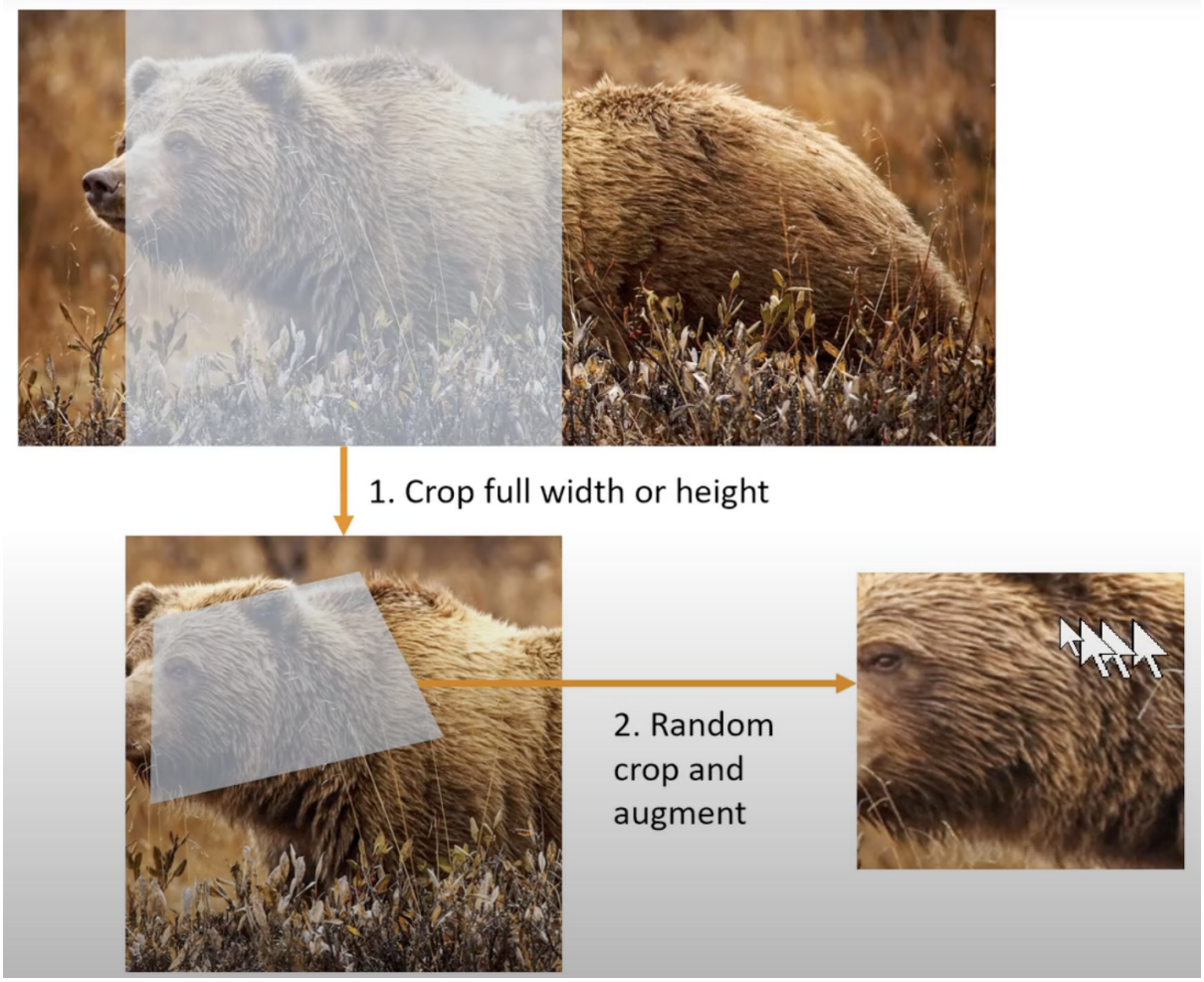
---

fastai/fastbook

`L` is an enhanced version of the default `list` in Python with added functionalities in fastai

---

- Use regex to get the breed of the cat/dog (the label) from the file name and create a datablock
- We are initially presizing to 460×460 from the portrait/landscape photo or higher resolution photo, and then the aug transform (happens on GPU) will grab a warped crop and augment it to 224×224




```
pets = DataBlock(blocks = (ImageBlock, CategoryBlock),
                 get_items=get_image_files,
                 splitter=RandomSplitter(seed=42),
                 get_y=using_attr(RegexLabeller(r'(.+)_\\d+.jpg$'), 'name'),
                 item_tfms=Resize(460),
                 batch_tfms=aug_transforms(size=224, min_scale=0.75))
dls = pets.dataloaders(path/"images")
```

Three ways to resize images into square:

1. Squish = makes objects in a image fatter, skinnier, taller, or shorter
2. Pad = adds black padding horizontally or vertically
3. Crop = crops area inside a image maybe leaving something important out

---

People often teach to start by collecting a lot of data but Jeremy advice to build the model as quickly as possible because it can teach something about the problem.

Cross Entropy Loss is similar to the `mnist_loss` function we defined in the previous chapter except:

- it works when dependent variables have more than two categories
- results in faster and more reliable training

In a binary classification case, when we look at activations:

- the sigmoid of two activations don't add up to 1, as they should since the combined porbability should be 1 in a binary classification problem
- We can take the difference of the two activations. Take its sigmoid. And keep the other value 1 - `diff.sigmoid()` (we take the sigmoid to make sure the `diff` is between 0 and 1)

Extending it to more than two columns:

```
def softmax(x): return exp(x) / exp(x).sum(dim=1, keepdim=True)
```

In the binary case, softmax is identical to `diff.sigmoid()` and `1 - diff.sigmoid()`.

The output values of Softmax add up into one.

	output	exp	softmax
teddy	0.02	1.02	0.22
grizzly	-2.49	0.08	0.02
brown	1.25	3.49	0.76
		4.60	1.00

e.g. 0.22 = 1.02/4.60

The idea in Softmax is that when using e to power something, even a small increment increases the result a lot. This means that a little bit bigger output will become a lot bigger softmax. This means that Softmax wants to pick one class.

```
#targ is the labels
targ = ([0,1,0,1,0,1])
#sm_acts is the softmax activations
#tensor indexing:
idx = range(6)
sm_acts(idx, targ)
#or through PyTorch
-F.nll_loss(sm_acts, targ, reduction='none')
```

But our probabilities are limited between 0 and 1, and the model won't see much difference between 0.99 and 0.999 even though the latter is ten times more precise. So we want to transform the numbers between 0 and 1 to between negative inf and positive inf. **Log**.

Cross Entropy Loss = Softmax + Log

`log(a * b) = log(a) + log(b)` This is used a lot in deep learning.

Two ways to use PyTorch

```
loss_func = nn.CrossEntropyLoss()
loss_func(acts, targ)
```

and

```
F.cross_entropy(acts, targ)
```