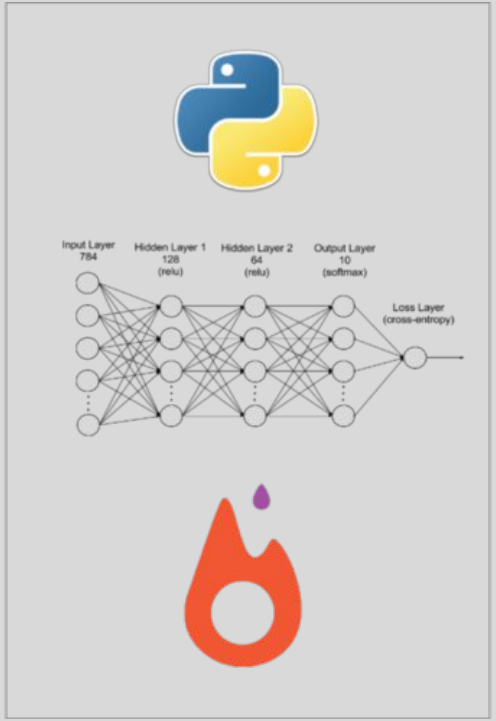
Pytorch intro

By Alex Torres

Starting off

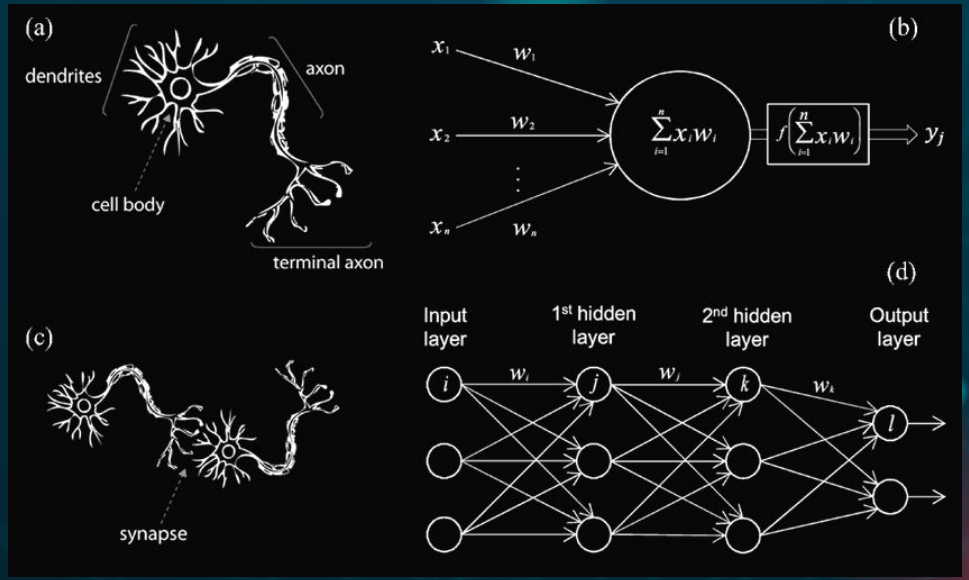
1. I am not qualified, just a curious student who wants to show others.
2. Will not be getting into the math behind it all view sources for more info
3. You can view live editing using this link: https://colab.research.google.com/drive/1OhDp_z_Lha92ilbuoxfg0uyErZ5gzm7I?usp=sharing
4. Q/A at the end of presentation.
5. It's ok if you get confused. Takes time.
6. **Thank you.**


Input Layer
784
Hidden Layer 1
128
(relu)
Hidden Layer 2
64
(relu)
Output Layer
10
(softmax)
Loss Layer
(cross-entropy)

2

What is Deep Learning

Deep learning was created as a subset of machine learning in which the computer mimics the human brain using neural networks.



Types

Analyzes labeled training data and produces an output, which can be used for mapping new examples.

Supervised

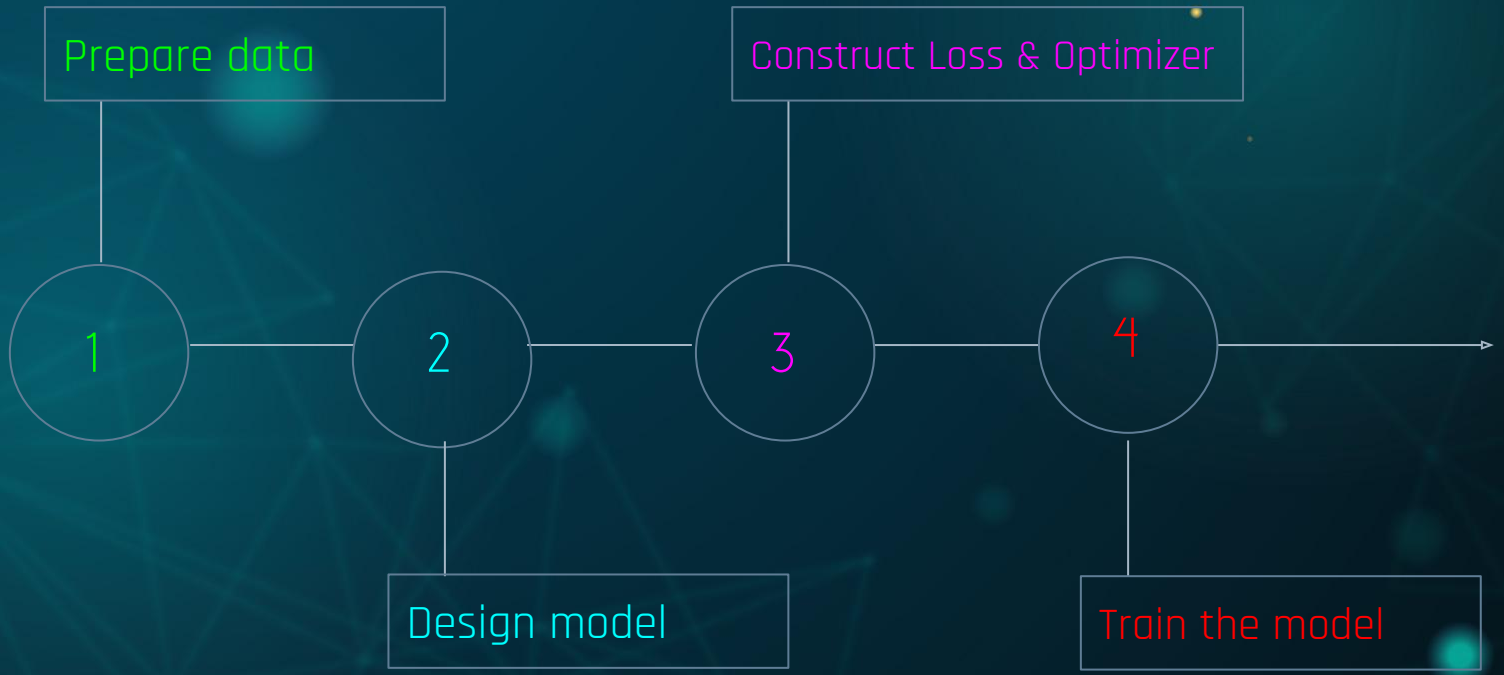
Combines a small amount of labeled data with a large amount of unlabeled data during training.

Semi-Supervised

Through mimicry, the machine learns patterns from untagged data.

Unsupervised

The Process


Prepare data
Construct Loss & Optimizer
1
2
3
4
Design model
Train the model

Full code overview

```
train = datasets.MNIST('', train=True, download=True,
                       transform=transforms.Compose([
                           transforms.ToTensor()
                       ]))

test = datasets.MNIST('', train=False, download=True,
                       transform=transforms.Compose([
                           transforms.ToTensor()
                       ]))

trainset = torch.utils.data.DataLoader(train, batch_size=10, shuffle=True)
testset = torch.utils.data.DataLoader(test, batch_size=10, shuffle=False)
```

Prepare data

1

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 32)
        self.fc2 = nn.Linear(32, 32)
        self.fc3 = nn.Linear(32, 10)

    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)

net = Net()
```

2

Design model

Construct Loss & Optimizer

```
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

3

```
for epoch in range(3): # 3 full passes over the data
    for data in trainset:  # `data` is a batch of data
        X, y = data  # X is the batch of features, y is the batch of targets.
        net.zero_grad()  # sets gradients to 0 before loss calc. You will do this likely every step.
        output = net(X.view(-1,784))  # pass in the reshaped batch (recall they are 28x28 atm)
        loss = F.nll_loss(output, y)  # calc and grab the loss value
        loss.backward()  # apply this loss backwards thru the network's parameters
        optimizer.step()  # attempt to optimize weights to account for loss/gradients
    print(loss)  # print loss. We hope loss (a measure of wrong-ness) declines!
```

4

Train the model

Breath

Prepare Data

Most Time consuming part.

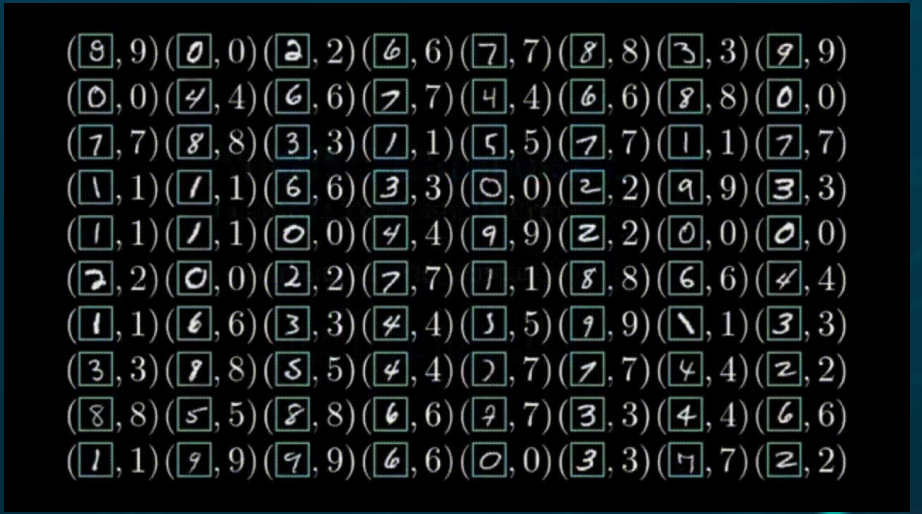
For this example we are using premade data from the torchvision datasets but this is not how it works in the real world.

Make a Train and a Test set. We will use the trainset for our model then test the model on the testset.

```
train = datasets.MNIST('', train=True, download=True,
                       transform=transforms.Compose([
                           transforms.ToTensor()
                       ]))

test = datasets.MNIST('', train=False, download=True,
                       transform=transforms.Compose([
                           transforms.ToTensor()
                       ]))

trainset = torch.utils.data.DataLoader(train, batch_size=10, shuffle=True)
testset = torch.utils.data.DataLoader(test, batch_size=10, shuffle=False)
```

The MNIST data set

More vision data sets can be explored using the following link:
https://pytorch.org/vision/0.8/datasets.html



Design Model

Our neural net called Net is a class the inherits from the parent class nn.Module. This is where the use of pytorch makes things very easy.

We define our layers with fc. Our first Layer takes an input of 784 because our image is 28 by 28 pixels and outputs 32 parameters.
Followed by other linear arrays with a given number of parameters and the last layer outputs 10 because there are 10 different classification, 0-9.

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 32)
        self.fc2 = nn.Linear(32, 32)
        self.fc3 = nn.Linear(32, 10)
```

Linear layer options

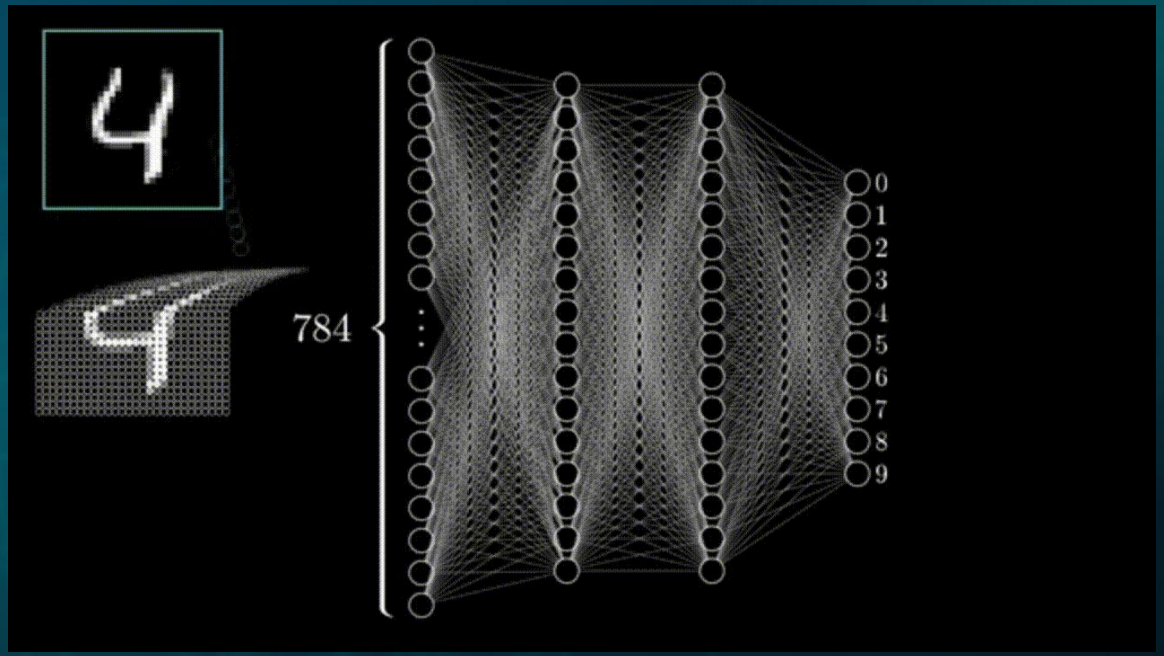
Linear Layers

`nn.Identity`	A placeholder identity operator that is argument-insensitive.
`nn.Linear`	Applies a linear transformation to the incoming data: $y = xA^T + b$
`nn.Bilinear`	Applies a bilinear transformation to the incoming data: $y = x_1^T A x_2 + b$
`nn.LazyLinear`	A `torch.nn.Linear` module with lazy initialization.

More info here

https://pytorch.org/docs/stable/nn.html#linear-layers

Forward function



Design Model

Our forward function will take a parameter which we stated was 784 values from our images. Then we make the values go through our first layer which makes x = 32 values. These values are then wrapped with a sigmoid function (activation function) to be within a range of 0-1.

Finally return the resulting 10 values and make them go through log_softmax to give probabilities on what the image might be.

Then initialize the object.

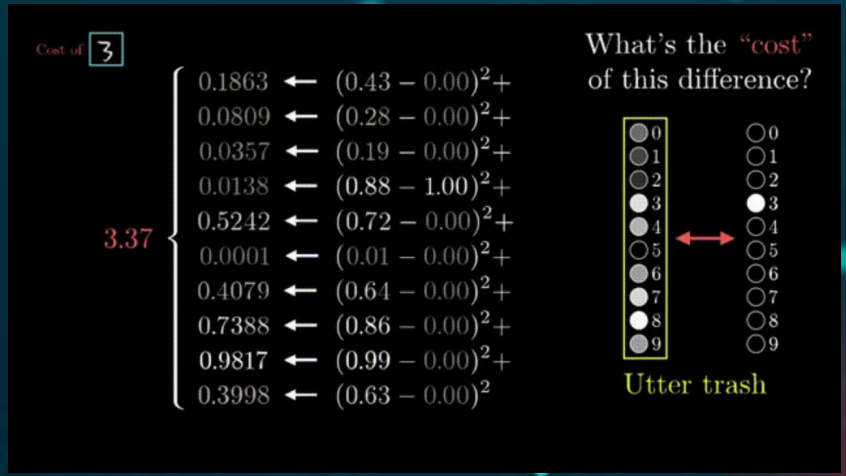
```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 32)
        self.fc2 = nn.Linear(32, 32)
        self.fc3 = nn.Linear(32, 10)

    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)

net = Net()
```

Construct Loss and Optimizer

We will be using CrossEntropyLoss as our loss function which is used for dictating how severely we should be punishing the model for getting the wrong answer.

The optimizer we will be using is adaptive momentum that will take in our model's "adjustable parameters" and a learning rate suitable for the best outcome.

```
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
```



Pytorch Loss Functions
https://pytorch.org/docs/stable/nn.html#loss-functions

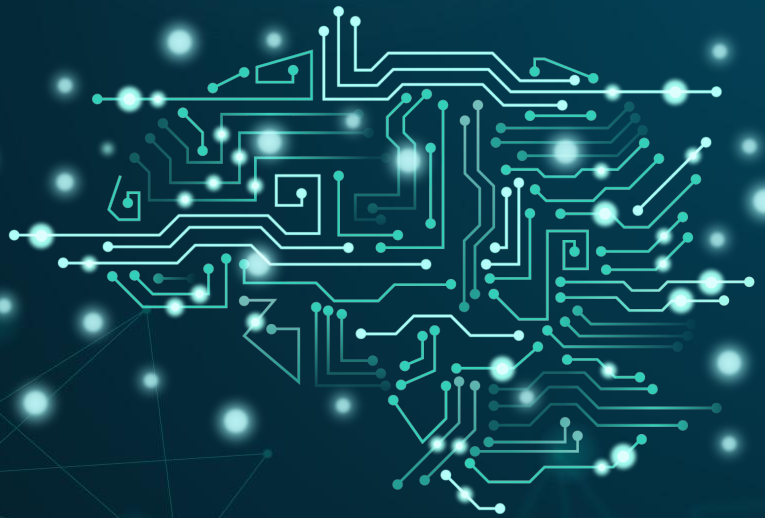
Pytorch Optimizers
https://pytorch.org/docs/stable/optim.html

Train the Model

In this section the model will go through three training sessions.
Each training session will iterate through a batch from our trainset which we specified 10.
X will have the image's and y will have the correct answer for that image (Labels).
Set the gradients to 0 so those value dont get added into the loss values.
Our output will call forward from our model to return some values for each image.
Loss will be our values compared to our labels and use that to backpropagate through our model.
Finally use optimizer.step() to adjust the weights for our model.

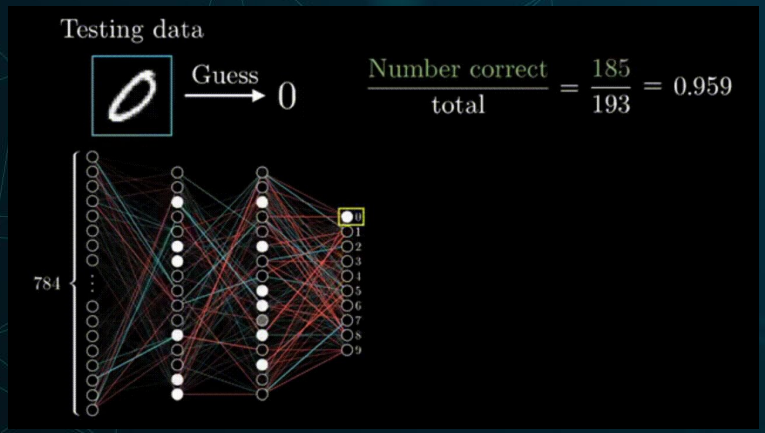
```
for epoch in range(3): # 3 full passes over the data
    for data in trainset:  # `data` is a batch of data
        X, y = data  # X is the batch of features, y is the batch of targets.
        net.zero_grad()  # sets gradients to 0 before loss calc. You will do this likely every step.
        output = net.forward(X.view(-1,784))  # pass in the reshaped batch (recall they are 28x28 atm)
        loss = loss_function(output, y)  # calc and grab the loss value
        loss.backward()  # apply this loss backwards thru the network's parameters
        optimizer.step()  # attempt to optimize weights to account for loss/gradients
    print(loss)  # print loss. We hope loss (a measure of wrong-ness) declines!
```

WOW! You made a brain!



Now let's test it

This will differ with other projects but essentially what is happening is we grab a batch of test images and computes a prediction. Then we keep track of the amount of images and the number of correct guesses. Lastly print the accuracy of our model, which isn't bad at all.




```
correct = 0
total = 0

with torch.no_grad():
    for data in testset:
        X, y = data
        output = net(X.view(-1,784))
        #print(output)
        for idx, i in enumerate(output):
            #print(torch.argmax(i), y[idx])
            if torch.argmax(i) == y[idx]:
                correct += 1
            total += 1

print("Accuracy: ", round(correct/total, 3))
```

```
Accuracy:  0.956
```

Q/A

Sources

Sentdex
https://pythonprogramming.net/introduction-deep-learning-neural-network-pytorch/

3Blue1Brown
https://www.youtube.com/watch?v=aircAruvnKk&t=6s