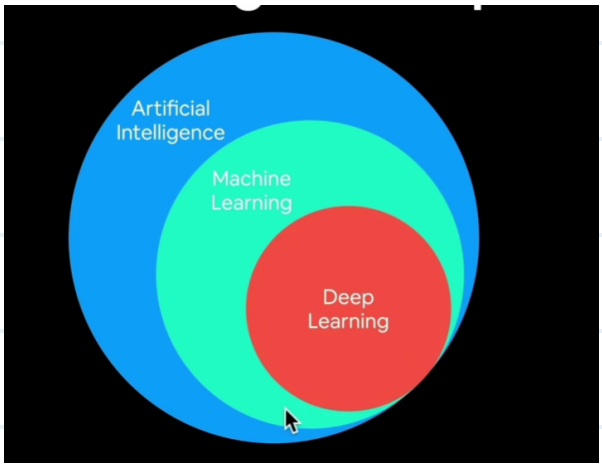
What is ML

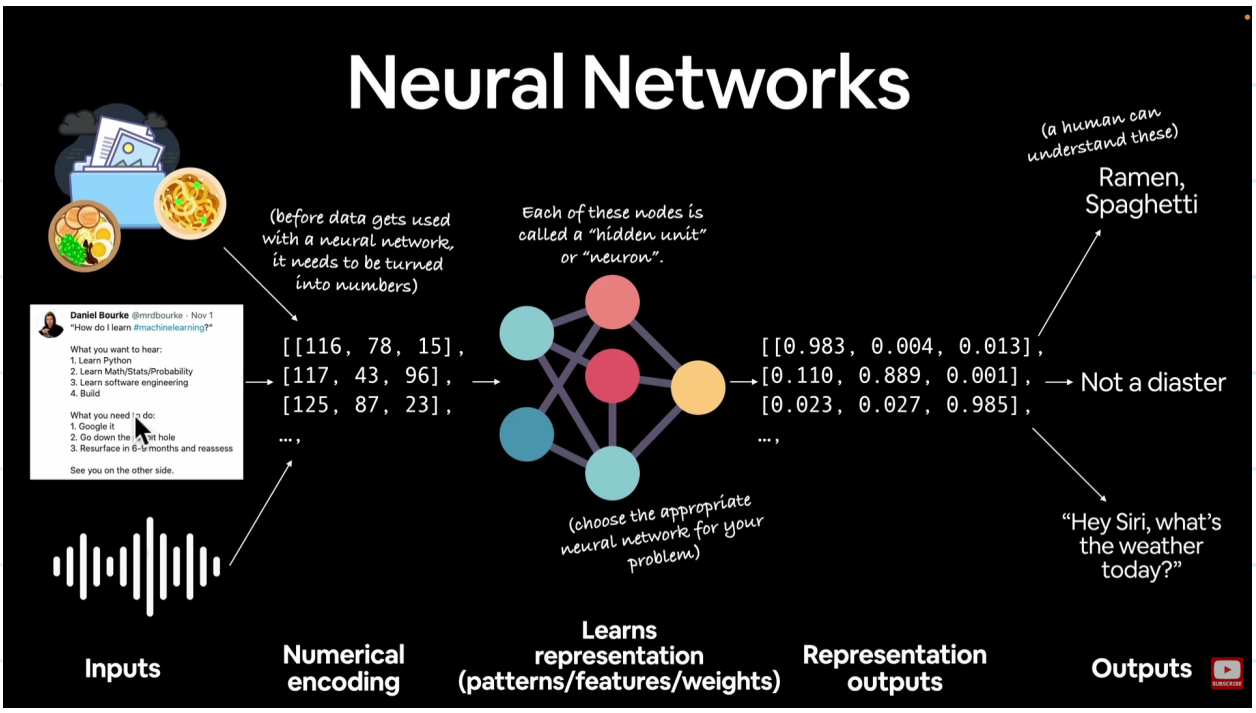
Machine learning is turning things (data) into numbers and finding patterns in those numbers.

algorithm the things that we're going to

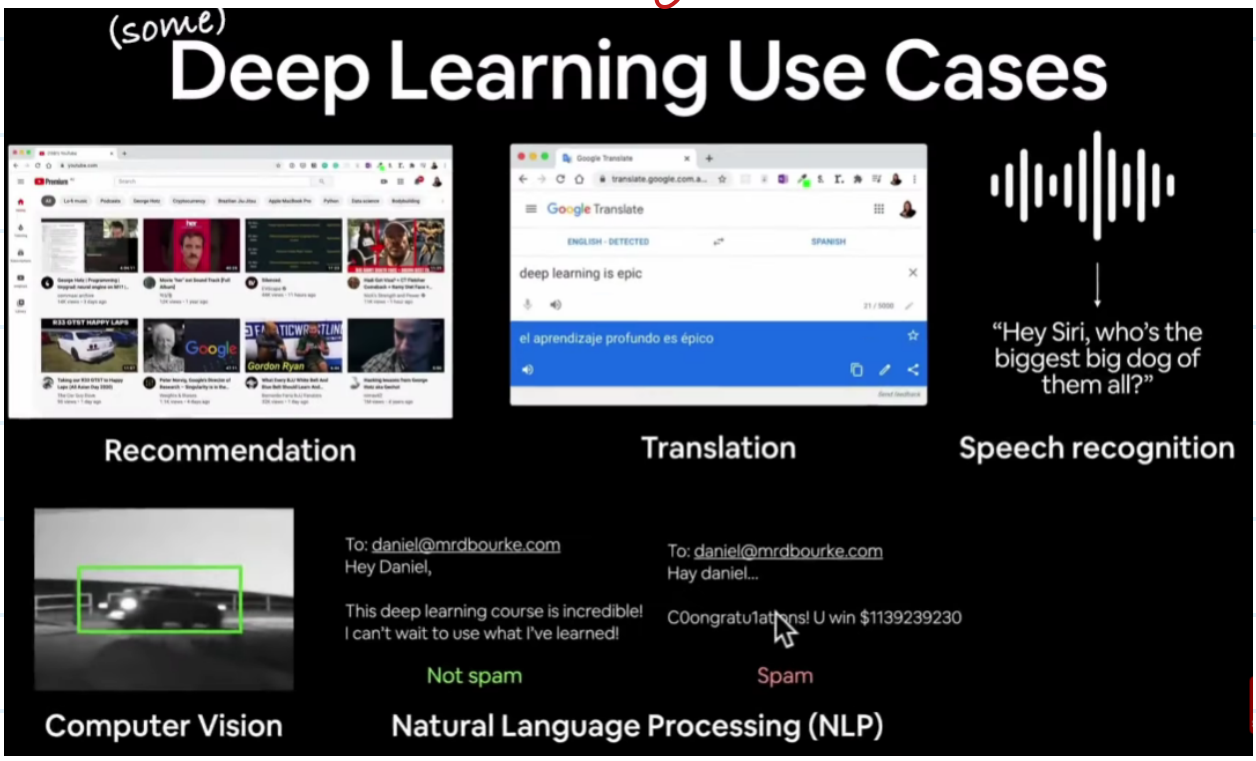
The computer does this part.
How?
Cod & math.
We're going to be writing the code.



What are NNs?



Cool Deep Learning Applications



Why pytorch?

- Researcher Friendly
- Production Ready
- Pythonic

> **Note:** When you run into issues in PyTorch, it's very often one to do with one of the three attributes above. So when the error messages show up, sing yourself a little song called "what, what, where":
>
> - *"what shape are my tensors? what datatype are they and where are they stored? what shape, what datatype, where where where"*

What is a tensor?

a **tensor** is a multi-dimensional array, the fundamental data structure for representing and manipulating data like numbers, vectors, matrices, and higher-order structures, serving as the core of frameworks like TensorFlow and PyTorch for tasks such as image, audio, and text processing, and holding model weights and biases in neural networks.

What is a vector?

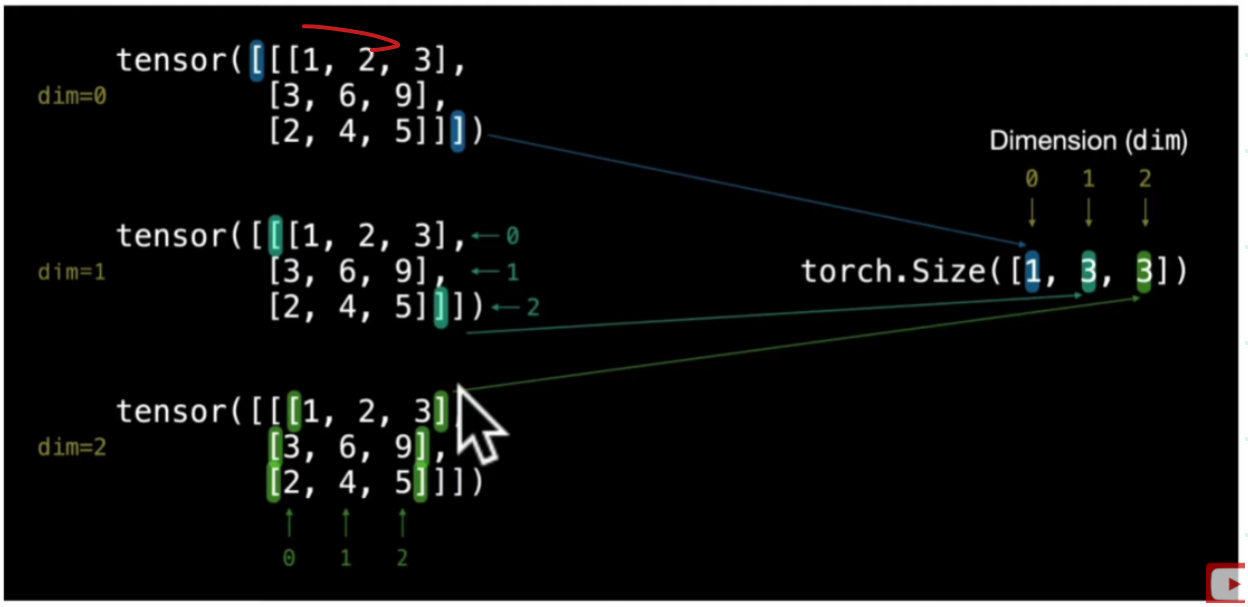
In machine learning, a vector is an ordered list (or array) of numbers that numerically represents a data point, with each number corresponding to a specific feature or attribute, allowing algorithms to process and understand complex information like images, text, or user preferences as mathematical objects. Essentially, it's how real-world data, with its diverse characteristics, gets converted into a format that computers can use for tasks like classification, recommendation, and pattern recognition.

-

-

Tensor Representation:

That means there's 1 dimension of 3 by 3.



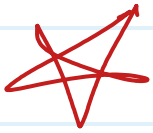
Naming Convections:

- Scalar / Vector → lower case
- Matrix / Tensor → Upper case

Why Random Tensor?

Random tensors are important because the way many neural networks learn is that they start with tensors full of random numbers and then
adjust those random numbers to better represent the data.

Start with random numbers -> look at data -> update random numbers -> look at data -> update random numbers



Note

Note: When you run into issues in PyTorch, it's very often one to do with one of the three attributes above. So when the error messages show up, sing yourself a little song called "what, what, where":

. "what shape are my tensors? what datatype are they and where are they stored? what shape, what datatype, where where where"

Reshaping, stacking, squeezing and unsqueezing

Often times you'll want to reshape or change the dimensions of your tensors without actually changing the values inside them.

To do so, some popular methods are:

Method	One-line description
`torch.reshape(input, shape)`	Reshapes `input` to `shape` (if compatible), can also use `torch.Tensor.reshape()`.
`Tensor.view(shape)`	Returns a view of the original tensor in a different `shape` but shares the same data as the original tensor.
`torch.stack(tensors, dim=0)`	Concatenates a sequence of `tensors` along a new dimension (`dim`), all `tensors` must be same size.
`torch.squeeze(input)`	Squeezes `input` to remove all the dimenions with value `1`.
`torch.unsqueeze(input, dim)`	Returns `input` with a dimension value of `1` added at `dim`.
`torch.permute(input, dims)`	Returns a *view* of the original `input` with its dimensions permuted (rearranged) to `dims`.

torch.view() → creates a new view of same tensor
changing view changes og. tensor too.

Reproducibility in Neural N/ws :

Reproducibility (trying to take the random out of random)

As you learn more about neural networks and machine learning, you'll start to discover how much randomness plays a part.

Well, pseudorandomness that is. Because after all, as they're designed, a computer is fundamentally deterministic (each step is predictable) so the randomness they create are simulated randomness (though there is debate on this too, but since I'm not a computer scientist, I'll let you find out more yourself).

How does this relate to neural networks and deep learning then?

We've discussed neural networks start with random numbers to describe patterns in data (these numbers are poor descriptions) and try to improve those random numbers using tensor operations (and a few other things we haven't discussed yet) to better describe patterns in data.

In short:

```
start with random numbers -> tensor operations -> try to make better (again and again and again)
```

Although randomness is nice and powerful, sometimes you'd like there to be a little less randomness.

Why?

So you can perform repeatable experiments.

For example, you create an algorithm capable of achieving X performance.

And then your friend tries it out to verify you're not crazy.

How could they do such a thing?

That's where **reproducibility** comes in.