Data Science 101 Workshop

Winter 2024

Ryan Vaz

01

Statistical Learning - Intro

What’s the difference between Statistical Learning and Machine Learning?

Machine learning turns things (data) into numbers and finds patterns in those numbers

Statistical learning is the theoretical foundation for a machine learning frameworks.

Connects statistics, linear algebra, functional analysis and programming

While it is important to know what job is performed by each cog, it is not necessary to have the skills to construct the machine inside the box!

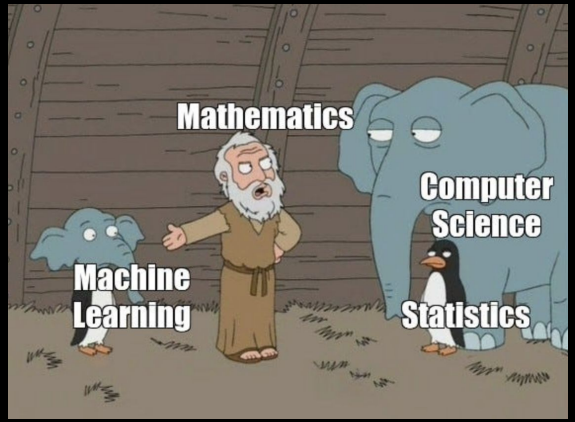
However statistical learning should not be viewed as a series of black boxes!

$ pip install ISLP

At the end of the day, we're talking about the same subject

Statistical Learning aims to understand and be able to explain the process or model
ML is focused on the best possible result no matter how complex and intractable the model.

ML and SL have so much overlap, label yourself with the more fitting position for what you apply to!



Some problems addressed with ML

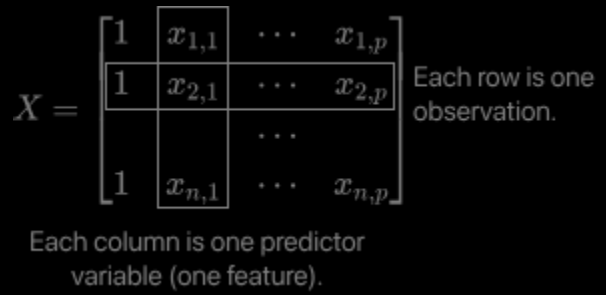
- Identify the risk factors for some type of cancer
- Spam mail detection
- Classification of tissue samples into different cancer classes
- Establish the relationship between salary and demographic in survey data
- Predict whether someone will have a heart attack on the basis of demographic, diet, and clinical measurements

Notation:

n = number of observations (rows)
p = number of features/variables (columns)

$\in$ = is an element of / in
$\mathbb{R}$ = the set of real numbers

$x \in \mathbb{R}$ is read "the element x is in the set of real numbers"
i.e. x is a real number



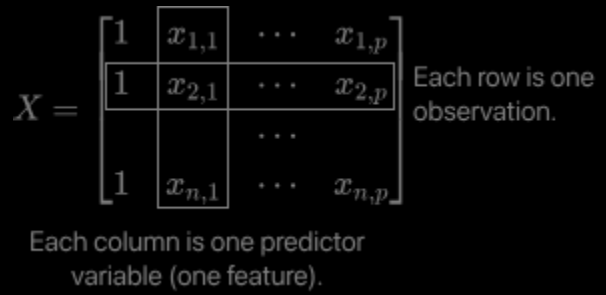
We will be using mathematical notation.
It's a new language!

Notation:

n = number of observations (rows)
p = number of features/variables (columns)

$\in$ = is an element of / in
$\mathbb{R}$ = the set of real numbers

$x \in \mathbb{R}$ is read "the element x is in the set of real numbers"
i.e. x is a real number



We will be using mathematical notation.
It's a new language!

Our Model

Input Variables = $\{X_1, X_2, \ldots, X_p\}$
Predictors / Independent Variables

Output Variable = Y
Response / Dependant Variable

We can assume that there is some relation between the input and output represented by:

$$Y = f(X) + \varepsilon$$

ε is a random error term with mean 0.
f represents the systematic information that X provides about Y.

In essence, statistical learning deals with different approaches to estimate *f*

A estimate would suffice

$$\hat{Y} = \hat{f}(x)$$

f hat is our *estimate* for f
Y hat is our *prediction* for Y

The accuracy of Ŷ depends on:

- Reducibility error
  - Due to f hat not being a perfect estimate
  - Can be reduced with the proper techniques
- Irreducible error
  - Due to **ε** and its variability
  - **ε** is independent from X, so no matter how well we estimate *f*, we can't reduce this error

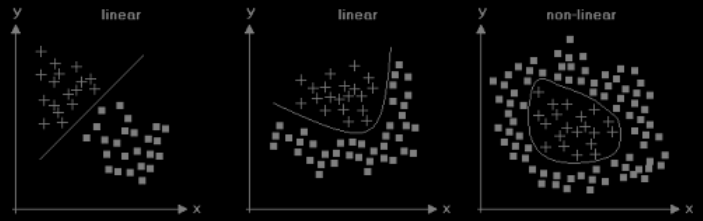
Inference

Here we are interested in **understanding the relation** between X and Y

Questions we should be asking:

- Which predictors are most associated with response?
- What is the relationship between the response and each predictor?
- Can such relationship be summarized via a linear equation, or is it more complex?

Linear models allow for easier interpretability, but can lack in prediction accuracy; while, non-linear models can be more accurate, but less interpretable



Yes I know this is getting old

Notation:

n: Number of observations
x_{ij}: Value of the j^{th} predictor, for the i^{th} observation
y_i: Response variable for the i^{th} observation

Training data:

- Set of observations
- Used to estimate f
- $\{(x_1, y_1), \cdots, (x_n, y_n)\}$ where $x_i = (x_{i1}, \cdots, x_{ip})^T$

Example of the training data:

{({“Angry”, “Wants to go to Hunter Halal”, “Rumbling Tummy”}, “Hungry”),
({“Wearing cap and gown”, “Happy”, “Never saw them again”}, “Graduated”)}

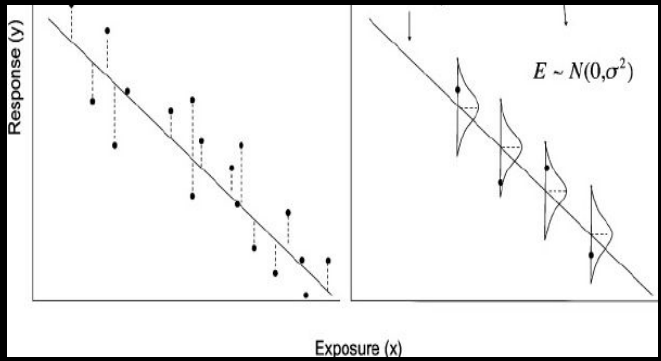
In this case f maps our *tokens* to the corresponding output

Parametric Modelling

1. Make an assumption about the **form** of f
   a. It could be linear: $f(x) = \beta_0 + \beta_1 X_1 + . + \beta_p X_p$
2. Select a procedure to **fit** the model using the training data
   a. The most common of such fitting procedures is called **ordinary least squares**

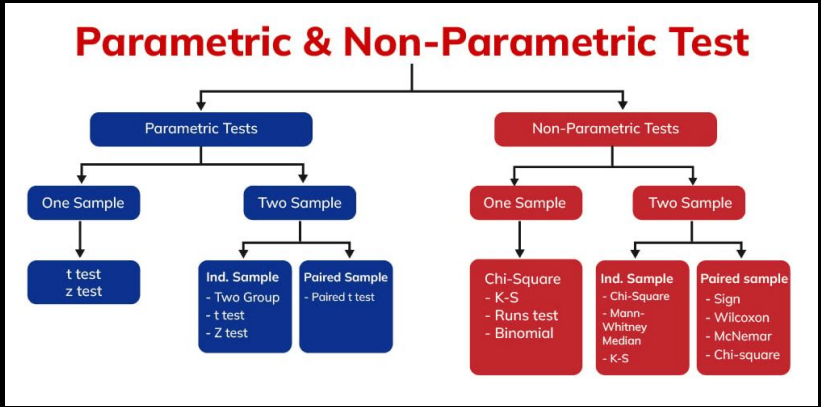
Via these steps, the problem of estimating f has been reduced to a problem of estimating a set of parameters β.

We can make the model more **flexible** via considering a greater number of parameters, but this can lead to **overfitting the data** i.e. if it follows the errors/noise too closely, it will not give accurate estimates for responses outside the original training data set.



Non-parametric methods

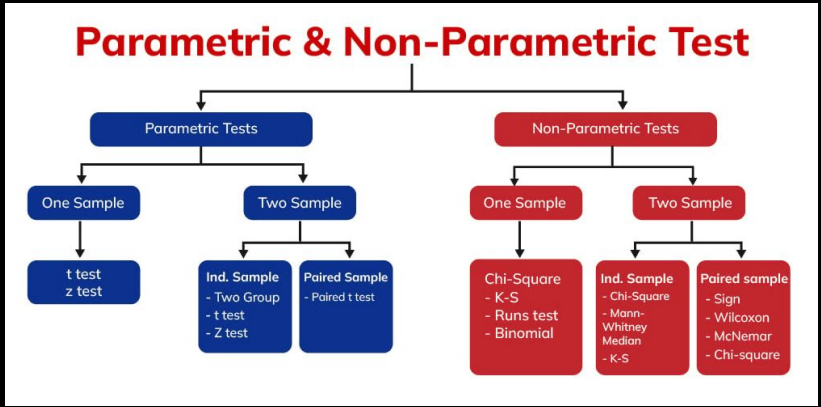
- No assumptions about the form of f are made
- Instead we seek an estimate of f which gets as close to the data point as possible
- Has the potential to fit a wider range of forms for f
- Typically requires a large and **fixed** amount of observations



What about a neural network?

Non-parametric methods

- No assumptions about the form of f are made
- Instead we seek an estimate of f which gets as close to the data point as possible
- Has the potential to fit a wider range of forms for f
- Typically requires a large and **fixed** amount of observations

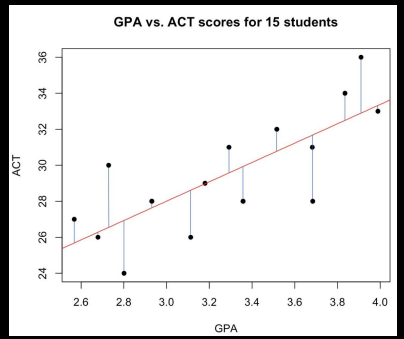


Don't fall for the interview trap! Most neural networks are parametric!
We will discuss its form later

So when do I use each model?

Parametric

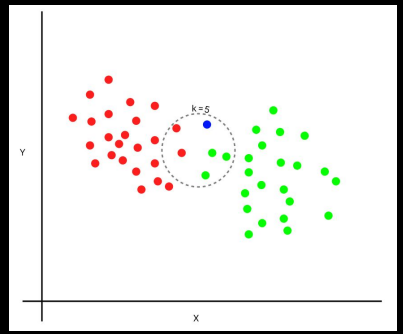
- These models are "**restrictive**"
- Restrictive models are more **interpretable**, so they are useful for inference



Line go uppy!

Non-Parametric

- These models are "**flexible**"
- Flexible models can be difficult to interpret due to the complexity



KNN

uhhhh

Despite this, we will often obtain more accurate predictions using a less flexible method due to the potential for *overfitting* in highly flexible models

Regression or Classification

This is the first question you should ask about any study.

If the **response** is...

- **Quantitative**, then, it's a regression problem
- **Categorical**, then, it's a classification problem

Measuring the quality of fit

How can we identify how good our model is?
We can use a value called **mean squared error**.

Instead of giving the formula, let me explain it this time...

Suppose we have 100 predictions, and 100 actual values
we take the difference between the predictions and actual values, then square them
lastly, we divide that value by 100.

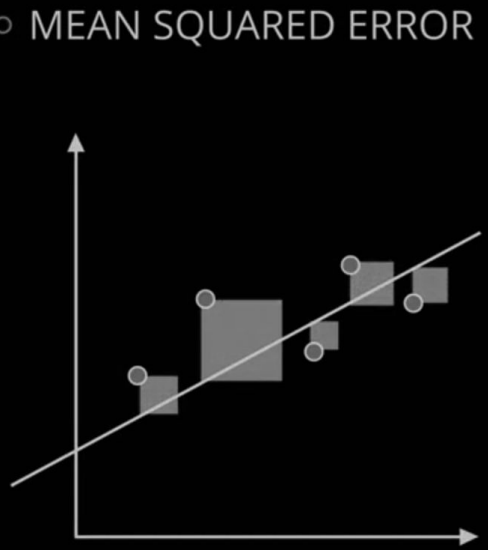
The closer this number is to 0, the more accurate our model was

Measuring the quality of fit

We want our model to accurately predict **unseen data** (testing data), not so much the training data, where the response is already known.

The *best* model will be those with the **lowest test MSE**, not necessarily the lowest training MSE

It's **not true** that the model with the lowest training MSE will also have the lowest test MSE

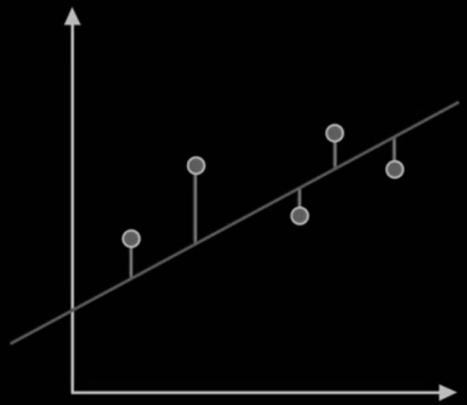


Measuring the quality of fit

Alternatively, instead of squaring the value, we can take the absolute value of the difference.

This value is called the **MAE** (mean absolute error)

- MEAN ABSOLUTE ERROR



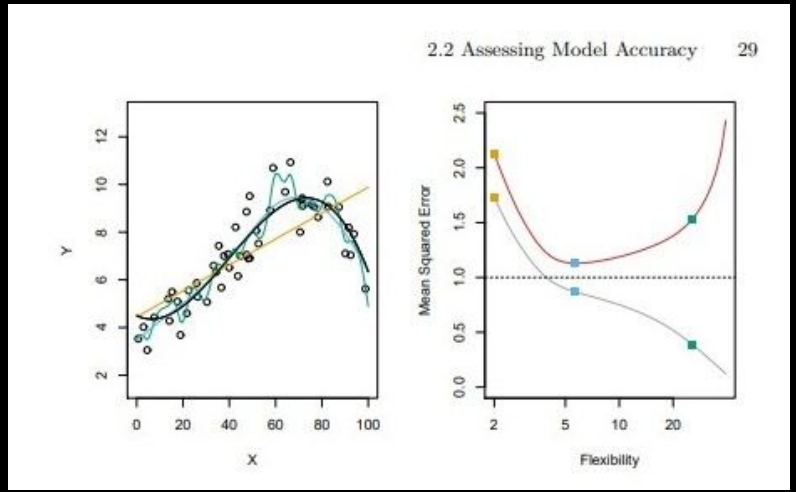
Overfitting

Fundamental property: For any data set and any statistical learning method used, as the flexibility if the statistical learning method increases:

- The training MSE decreases monotonically
- The test MSE graph has a U shape

Small training MSE but big test MSE implies that we overfitted on our data.

Regardless of overfitting, we always expect training MSE < testing MSE because most statistical learning methods seek to minimize the training MSE



also STDEV^2

The Bias-Variance Trade-off

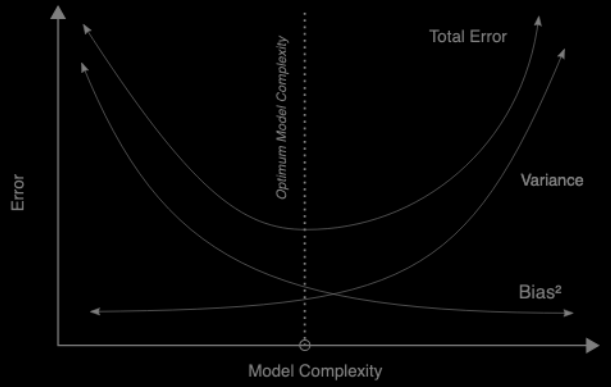
Variance: The average MSE after repeatedly estimating f using a large number of training sets

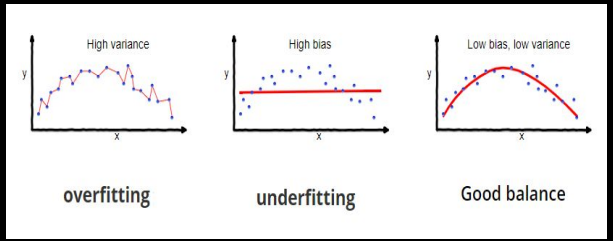
Bias: The error generated by approximating a complex model by a much simpler one

As a general rule of thumb, the **more flexible** a statistical method is, the **higher its variance,** and lower its bias

Tradeoff:

- *Extremely low bias, but high variance*: Draw a line that passes over every single point
- *Extremely low variance, but high bias*: Draw a horizontal line to fit the entire dataset





The Bias-Variance Trade-off

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + Bias(\hat{f}(x_0))^2 + Var(\epsilon)$$

The expected difference between actual and predictions

Leave the proofs to the math nerds

Due to variance and squared bias being non negative, this equation implies that if we want to **minimize the expected test error**, we require a statistical learning method with **low variance** and **low bias**

Supervised vs Unsupervised Learning

In **supervised learning**, we wish to fit a model that relates inputs/predictors to some output

In **unsupervised learning**, we lack a response/variable to predict. Instead we wish to understand the relationships between the variables or between the observations

There are instances where a mix of such methods are required (**semi-supervised learning problems**), but we won't have time to cover this topic

Categorical prediction

The most common approach for quantifying the accuracy of our estimate is the **training error rate**, the proportion of mistakes that our estimate makes:

$$\frac{1}{n}\sum_{i=1}^{n} 1_{y_i \neq \hat{y}_i}$$

one of my favorite functions: indicator function = 1 when the condition is true, else 0

The **test error rate** is defined similarly but uses our estimate on the testing set instead of the training set and comparing it with the true response y

A **good classifier** is one for which the test error is the smallest

Simple Linear Regression

- Supervised learning approach
- Linear regression = Linear model

Simple linear regression is a *very* straightforward approach to predicting response Y on predictor X. We may make the assumption that there is approximately a linear relationship between X and Y (i.e. a straight line)

$$Y \approx \beta_0 + \beta_1 X$$

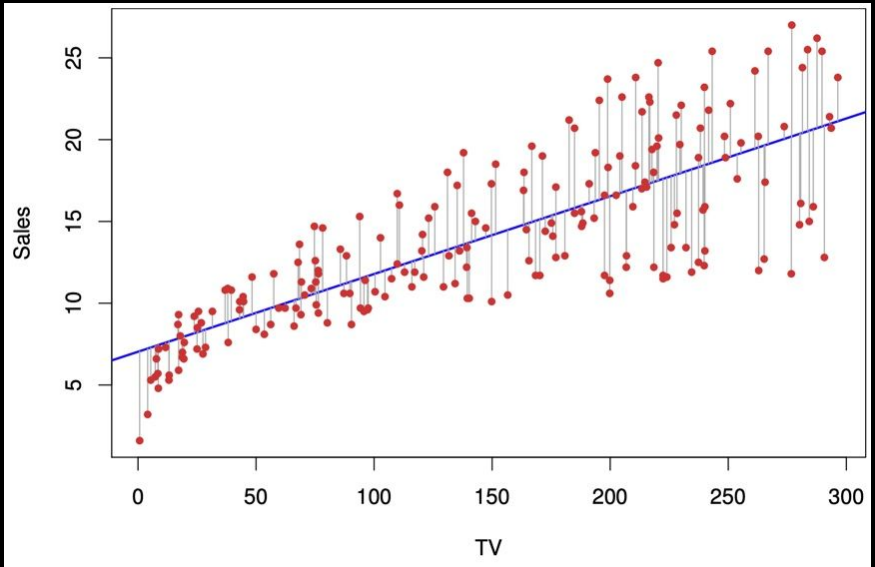
$$\beta_0 = \text{intercept}$$

$$\beta_1 = \text{slope}$$

≈ is read as "is approximately modeled by"
or "regressing Y on X"

y=mx + b
y=🙂x + 😢

Simple Linear Regression



Is this a good fit (MSE close to 0)?

Too much bias or too much variance?

How interpretable is this model?

Multiple Linear Regression

Multiple linear regression extends simple linear regression for *p* predictors:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon_i$$

$$Y \approx \beta_0 + \beta_1 X$$

$\beta_0 =$ intercept

$\beta_1 =$ slope

≈ is read as “is approximately modeled by”
or “regressing Y on X”

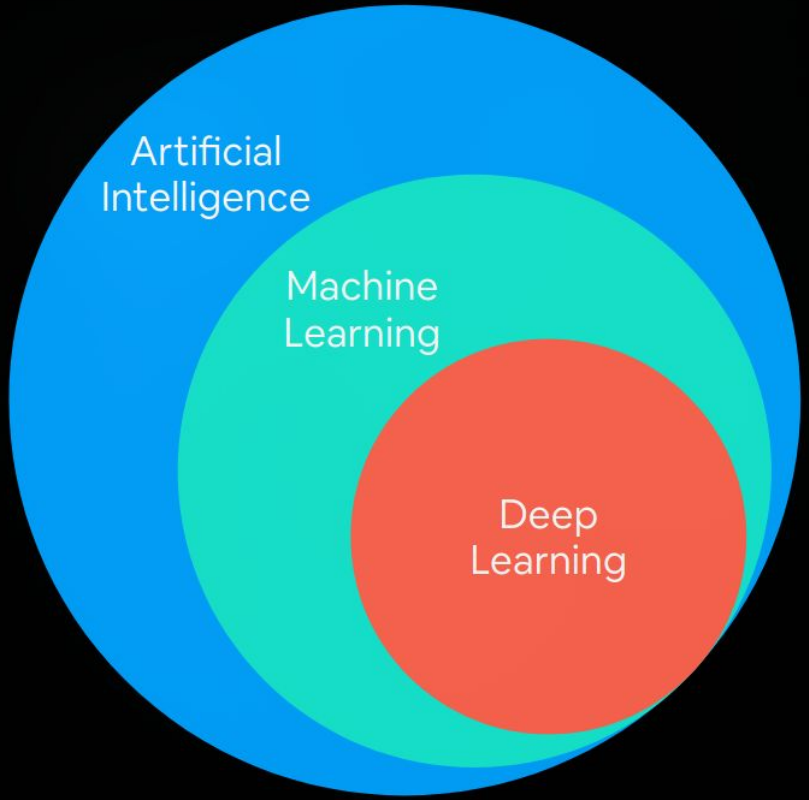
y=mx + b
y=🙂x + 😢

Let’s answer some
questions on the notebook

02

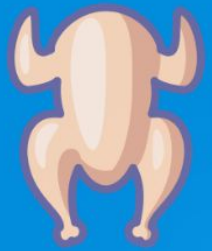
Deep Learning - Intro

Machine Learning vs. Deep Learning



Traditional programming

Inputs



Rules

1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

Output

Starts with **Makes**

Machine learning algorithm

Inputs

Output

Rules

1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

Starts with **Figures out**

Why do we use Deep Learning?

Why do we use Deep Learning?
Answer: Why Not?

Why do we use Deep Learning?

Answer: ~~Why Not?~~

We can't parameterize every problem

“If you can build a simple rule-based system that doesn’t require machine learning, do that.” - Google

https://developers.google.com/machine-learning/guides/rules-of-ml/

“If you can build a simple rule-based system that doesn’t require machine learning, do that.” - Google

https://developers.google.com/machine-learning/guides/rules-of-ml/

What deep learning is good for

- **Problems with long lists of rules** — when the traditional approach fails, machine learning/deep learning may help.
- **Continually changing environments** — deep learning can adapt ('learn') to new scenarios.
- **Discovering insights within large collections of data** — can you imagine trying to hand-craft rules for what 101 different kinds of food look like?

What deep learning is not good for

- **When you need explainability** — the patterns learned by a deep learning model are typically uninterpretable by a human.
- **When the traditional approach is a better option** — if you can accomplish what you need with a simple rule-based system.
- **When errors are unacceptable** — since the outputs of deep learning model aren't always predictable.
- **When you don't have much data —** deep learning models usually require a fairly large amount of data to produce great results.

Machine Learning vs. Deep Learning

- Random forest
- Gradient boosted models
- Naive Bayes
- Nearest neighbours
- Support vector machine
- ...many more

- Neural networks
- Fully connected neural network
- Convolutional neural network
- Recurrent neural network
- Transformer
- ...many many more

(since the advent of deep learning, these are often called shallow algorithms)

Machine Learning vs. Deep Learning

- Random forest
- Gradient boosted models
- Naive Bayes
- Nearest neighbours
- Support vector machine
- ...many more

- Neural networks
- Fully connected neural network
- Convolutional neural network
- Recurrent neural network
- Transformer
- ...many many more

(since the advent of deep learning, these are often called shallow algorithms)

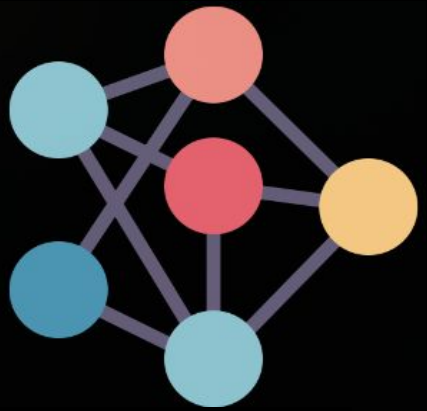
What is a neural network?

Neural Network

Physics is fun!

Before data can be fed into the network, it needs to be turned into numbers

[[32, 49, 183],
[231, 84, 18],
[96, 84, 122]]



[[0.2, 0.3, 0.9],
[0.6, 0.3, 0.4],
[0.1, 0.8, 0.7]]

Cat Image

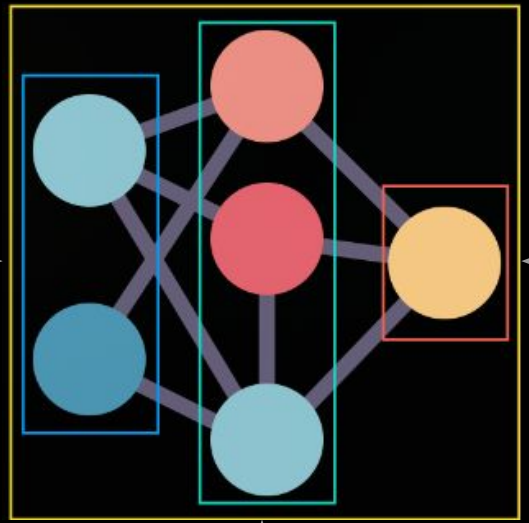
💀

Hey Siri

Input **Numerical Encoding** **Neural Network** **Representation Output** **Outputs**

Neural Network

Overall Architecture



Input layer
(data goes in here)
units/neurons = 2

Output layer
(outputs learned representation or prediction probabilities)
units/neurons = 1

Hidden layer(s)
(learns patterns in data)
units/neurons = 3

Note: There are numerous names for patterns such as: embeddings, weights, feature representation, feature vectors.

What library do we use?

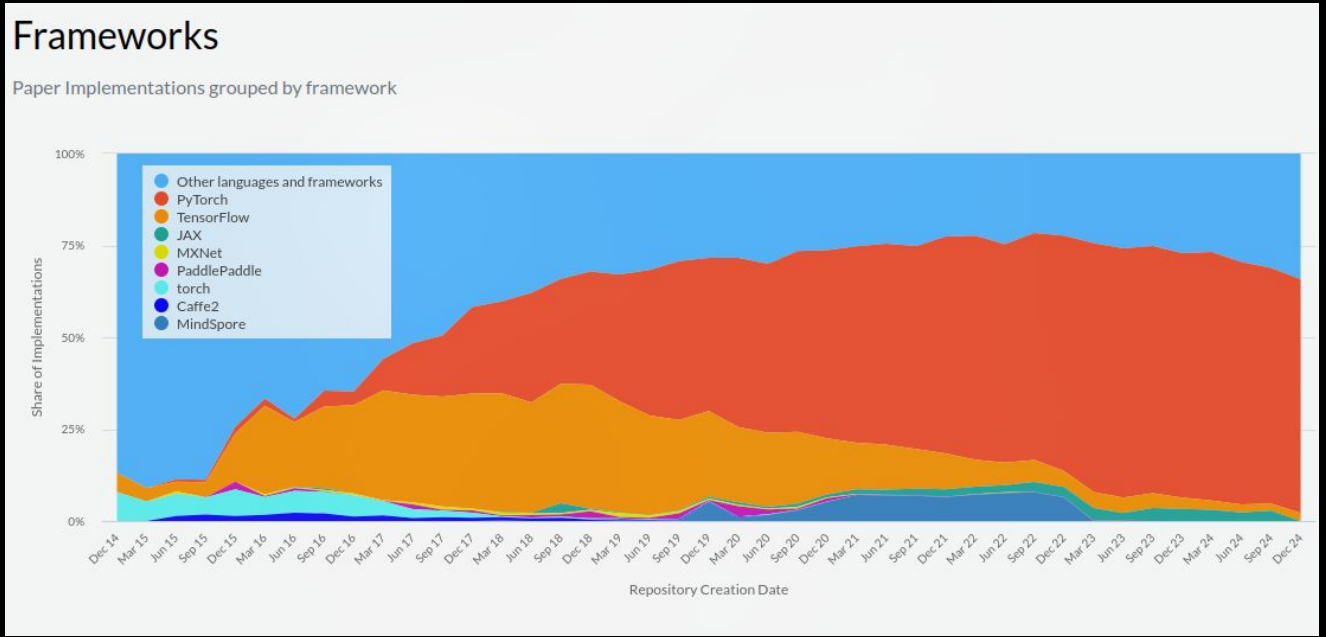
Tensorflow

by Google

Pytorch

by ~~Facebook~~
Meta

You may say there's a bias

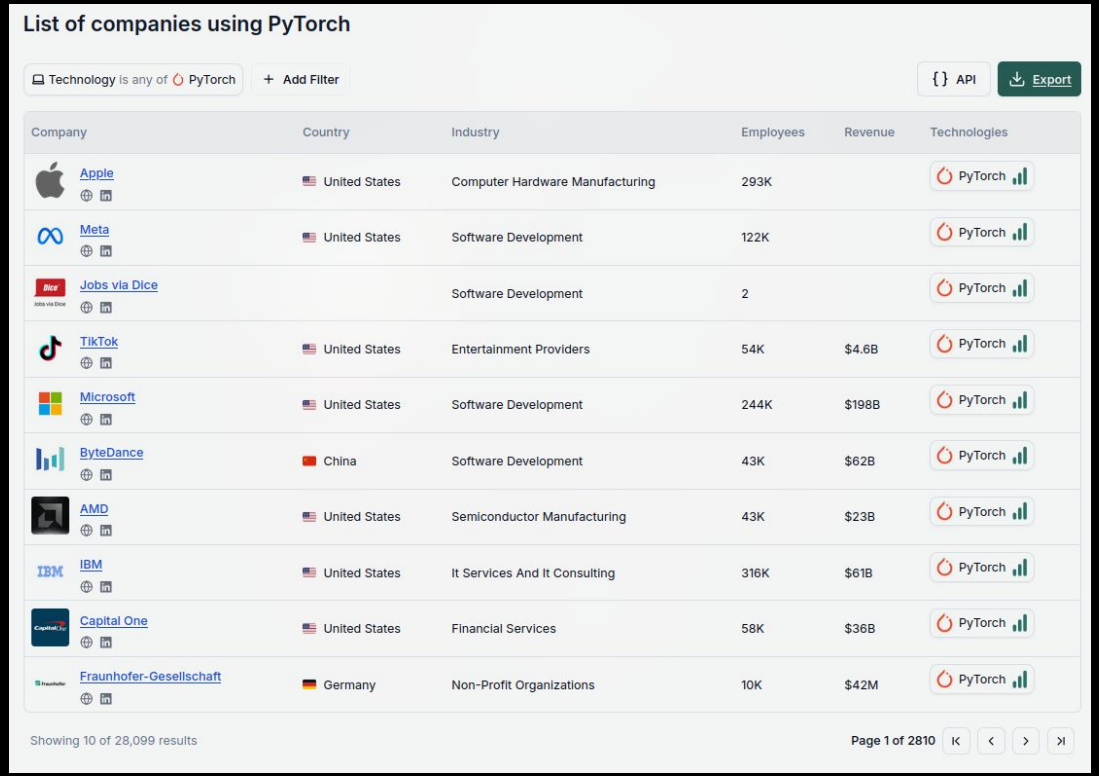


57% vs 2%
Today

Research Leader

https://paperswithcode.com/trends

Why Pytorch?



List of companies using PyTorch

Technology is any of PyTorch | + Add Filter | { } API | Export

Company	Country	Industry	Employees	Revenue	Technologies
Apple	United States	Computer Hardware Manufacturing	293K		PyTorch
Meta	United States	Software Development	122K		PyTorch
Jobs via Dice		Software Development	2		PyTorch
TikTok	United States	Entertainment Providers	54K	$4.6B	PyTorch
Microsoft	United States	Software Development	244K	$198B	PyTorch
ByteDance	China	Software Development	43K	$62B	PyTorch
AMD	United States	Semiconductor Manufacturing	43K	$23B	PyTorch
IBM	United States	It Services And It Consulting	316K	$61B	PyTorch
Capital One	United States	Financial Services	58K	$36B	PyTorch
Fraunhofer-Gesellschaft	Germany	Non-Profit Organizations	10K	$42M	PyTorch

Showing 10 of 28,099 results

Page 1 of 2810

Market Leader

https://theirstack.com/en/technology/pytorch

Why doesn't everyone use Deep Learning?

Where can I run it?

Well...bad news

CPU

Instructions that the CPU are capable of:

- Addition
- Subtraction
- Multiplication
- Division

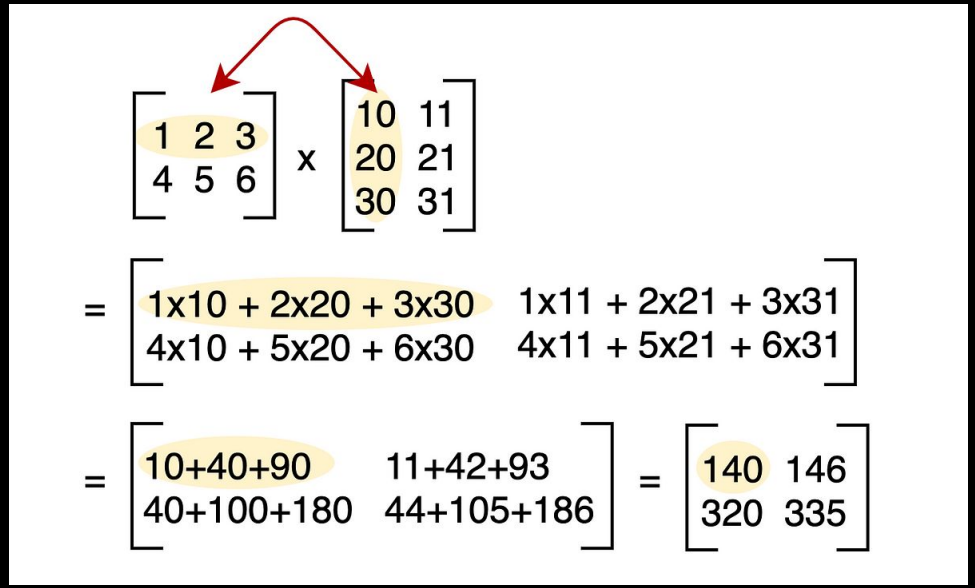
Scalar: A number, numerical quantity, or element in a field.

All these CPU operations are referred to as scalar operators.

If you've taken computer architecture shush...

Problem

We are working with matrices in Deep Learning



but it's just pluses and times...
why can't we do it anymore?

Threading

A CPU **thread** is a virtual core that allows a processor to handle multiple tasks simultaneously.

Most modern consumer CPUs have less than 32 threads which means they can perform at most 32 calculations at once

Our matrices will normally be massive, which means we'll see a dramatic slowdown if we stick to using a CPU

While we can do the calculation on CPUs, it will be very slow.

Parallel Processing

We need a device with even more threads than a CPU, luckily the computation of graphics calls for lots of matrix operations (and especially the computation of light rays)

Most GPUs (Graphics Processing Units) are designed with this computation in mind, where NVIDIA and AMD support parallel processing APIs, CUDA (Compute Unified Device Architecture) and ROCm (Radeon Open Compute platform) respectively.

For the sake of computation, note the fact that 1 CPU Core ~= 2 Threads ~= 2 GPU Cores
(due to hyper threading)

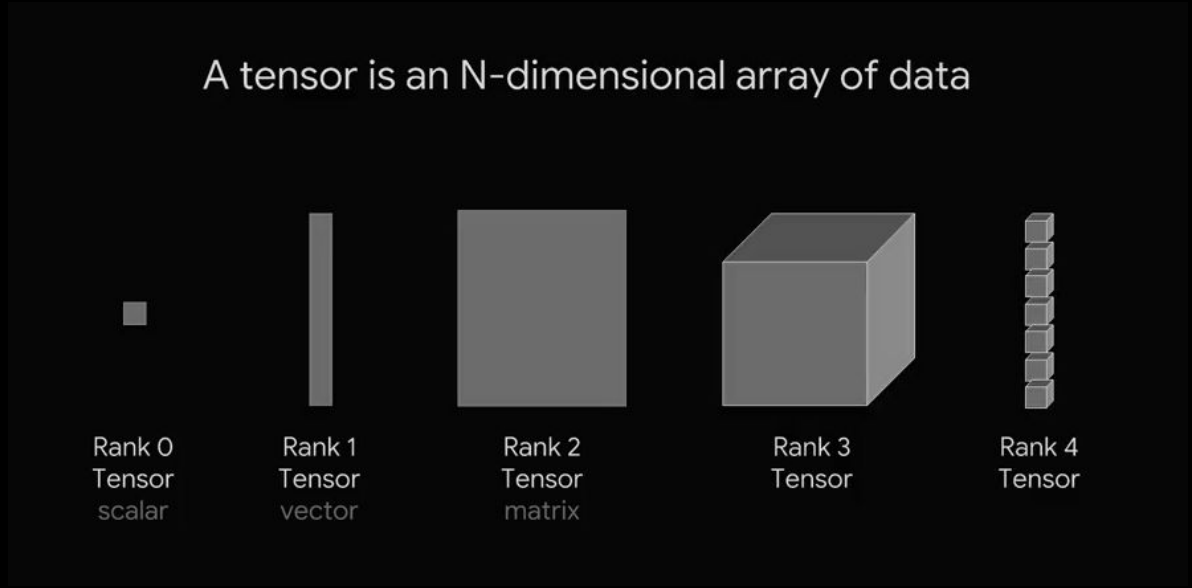
Alternatively, TPUs are even faster and hyper specified to this purpose, but are not available to the average consumer

DLSS / MFG

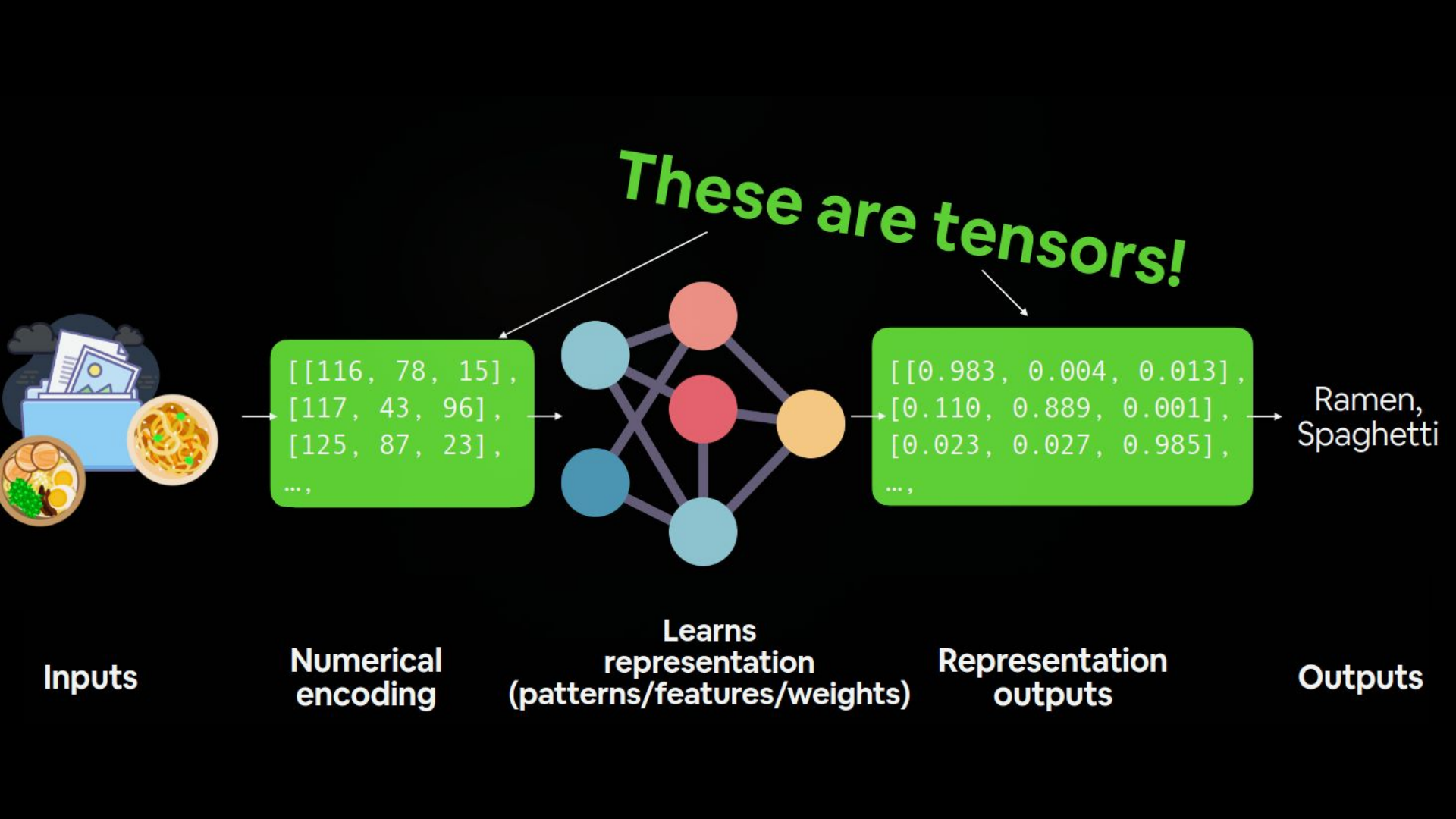
DLSS (or Deep Learning Super Sampling) is an example of a deep learning applied in the real world. Nvidia GPUs generate extra "interpolated" frames that are fed between two frames to generate a more smooth picture. MFG, the most modern version of DLSS supports up to 4 frames generated from any given frame, causing up to an 8x increase in frames.
Despite the benefits of this, as we've seen with the bias-variance trade off, it's impossible to make a model perfect. These errors in frame generation are called *artifacting*.

Real Frame | Interpolated Frame | Real Frame

Tensors



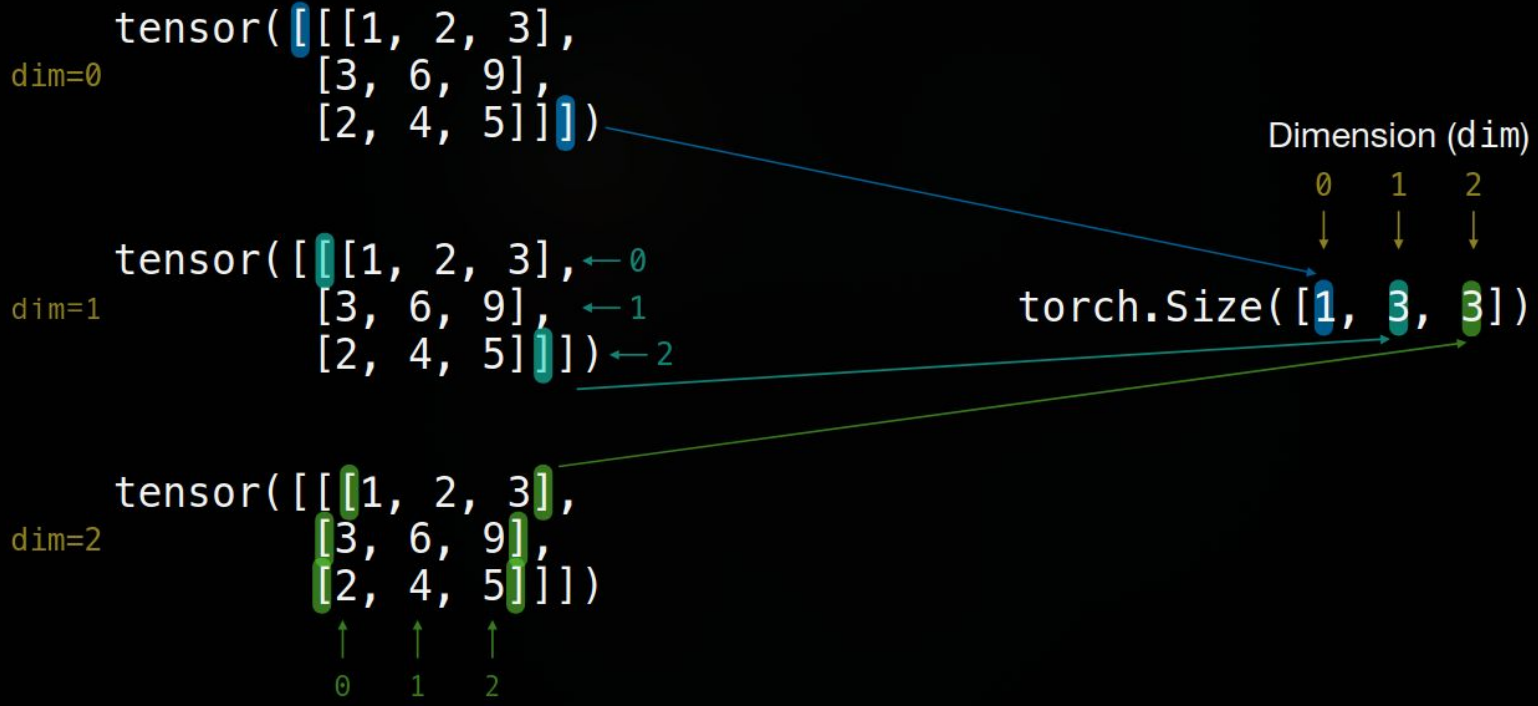
Tensors and dimensions are analogous


These are tensors!
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
…,
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,
Ramen,
Spaghetti
Inputs
Numerical encoding
Learns representation (patterns/features/weights)
Representation outputs
Outputs

Deep Learning Steps

1. Get the data ready (turn it into a tensor)
2. Pick an appropriate model (or build one!)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your model

Tensor dimensions

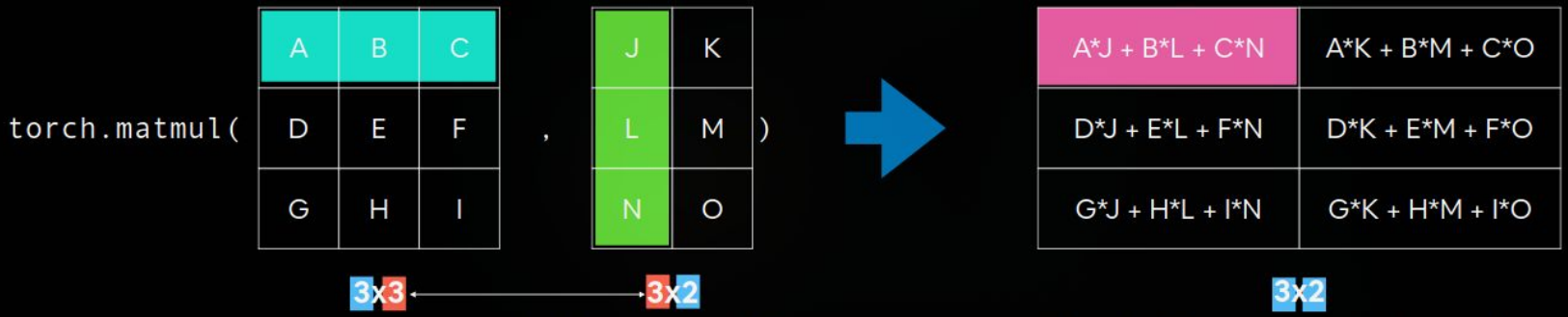


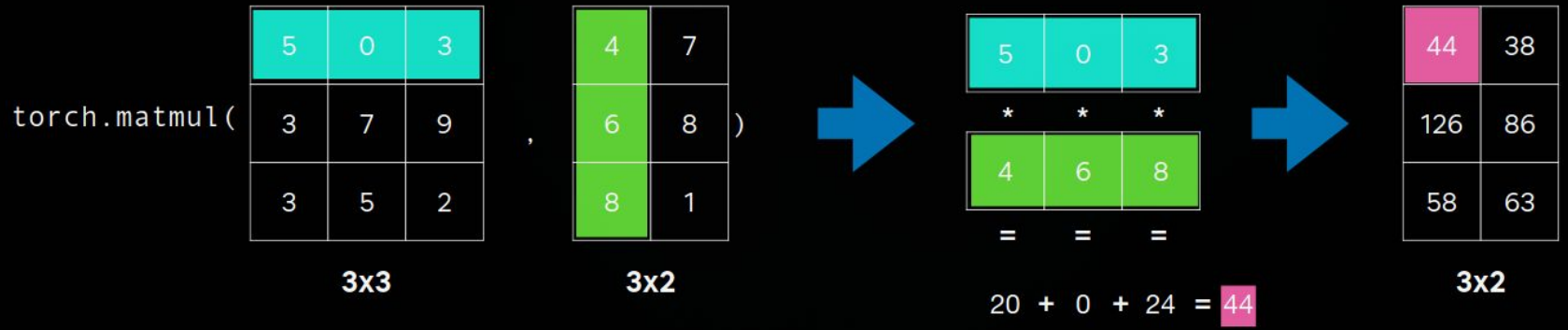
Tensor attributes

Attribute	Meaning	Code
Shape	The length (number of elements) of each of the dimensions of a tensor.	`tensor.shape`
Rank/dimensions	The total number of tensor dimensions. A scalar has rank 0, a vector has rank 1, a matrix is rank 2, a tensor has rank n.	`tensor.ndim or tensor.size()`
Specific axis or dimension (e.g. “1st axis” or “0th dimension”)	A particular dimension of a tensor.	`tensor[0], tensor[:, 1]…`

sounds familiar? it is! numpy has similar functionality

Dot product


torch.matmul(
A B C
D E F
G H I
,
J K
L M
N O
)
A*J + B*L + C*N
A*K + B*M + C*O
D*J + E*L + F*N
D*K + E*M + F*O
G*J + H*L + I*N
G*K + H*M + I*O
3x3
3x2
Numbers on the inside must match
3x2
New size is same as outside numbers

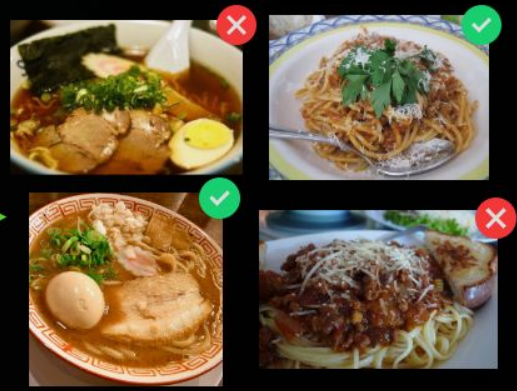

torch.matmul(
5 0 3
3 7 9
3 5 2
,
4 7
6 8
8 1
)
3x3
3x2
5 0 3
* * *
4 6 8
= = =
20 + 0 + 24 = 44
44 38
126 86
58 63
3x2


Credit fast.ai
class LSTMCell(Module):
def __init__(self, ni, nh):
self.forget_gate = nn.Linear(ni + nh, nh)
self.input_gate = nn.Linear(ni + nh, nh)
self.cell_gate = nn.Linear(ni + nh, nh)
self.output_gate = nn.Linear(ni + nh, nh)
def forward(self, input, state):
h,c = state
h = torch.stack([h, input], dim=1)
forget = torch.sigmoid(self.forget_gate(h))
c = c * forget
inp = torch.sigmoid(self.input_gate(h))
cell = torch.tanh(self.cell_gate(h))
c = c + inp * cell
out = torch.sigmoid(self.output_gate(h))
h = outgate * torch.tanh(c)
return h, (h,c)
ALWAYS HAS BEEN
WAIT SO YOU'RE TELLING ME
IT'S ALL MATRIX MULTIPLICATIONS?
imgflip.com

Supervised learning

1. Initialize our model with random weights

```
[[0.092, 0.210, 0.415],
[0.778, 0.929, 0.030],
[0.019, 0.182, 0.555],
…,
```



2. Make predictions on the data

```
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
…,
```

```
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,
```

Ramen, Spaghetti

3. Update the model

4. Run the model again

Inputs	Numerical encoding	Learns representation (patterns/features/weights)	Representation outputs	Outputs

03

Linear Regression

Three Datasets

Model learns from this

Training set
(Course Material)

Tune the parameters

Validation Set
(Practice Exam)

Training Data

Model gets tested here

Testing set
(Final exam)

Testing Data

Linear Regression Returns

```python
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.weights = nn.Parameter(torch.randn(1,
                                                dtype=torch.float),
                                    requires_grad=True)

        self.bias = nn.Parameter(torch.randn(1,
                                             dtype=torch.float),
                                 requires_grad=True)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.weights * x + self.bias
```

Subclass nn.Module
We inherit from the nn.Module class to gain (steal) pytorch's baseline for a neural network

We initialize the model's weight and bias tensor with a rank 1 tensor (in reality land we use a rank 2 tensor)

requires_grad=True tells pytorch to keep track of the gradient for this parameter using torch.autograd

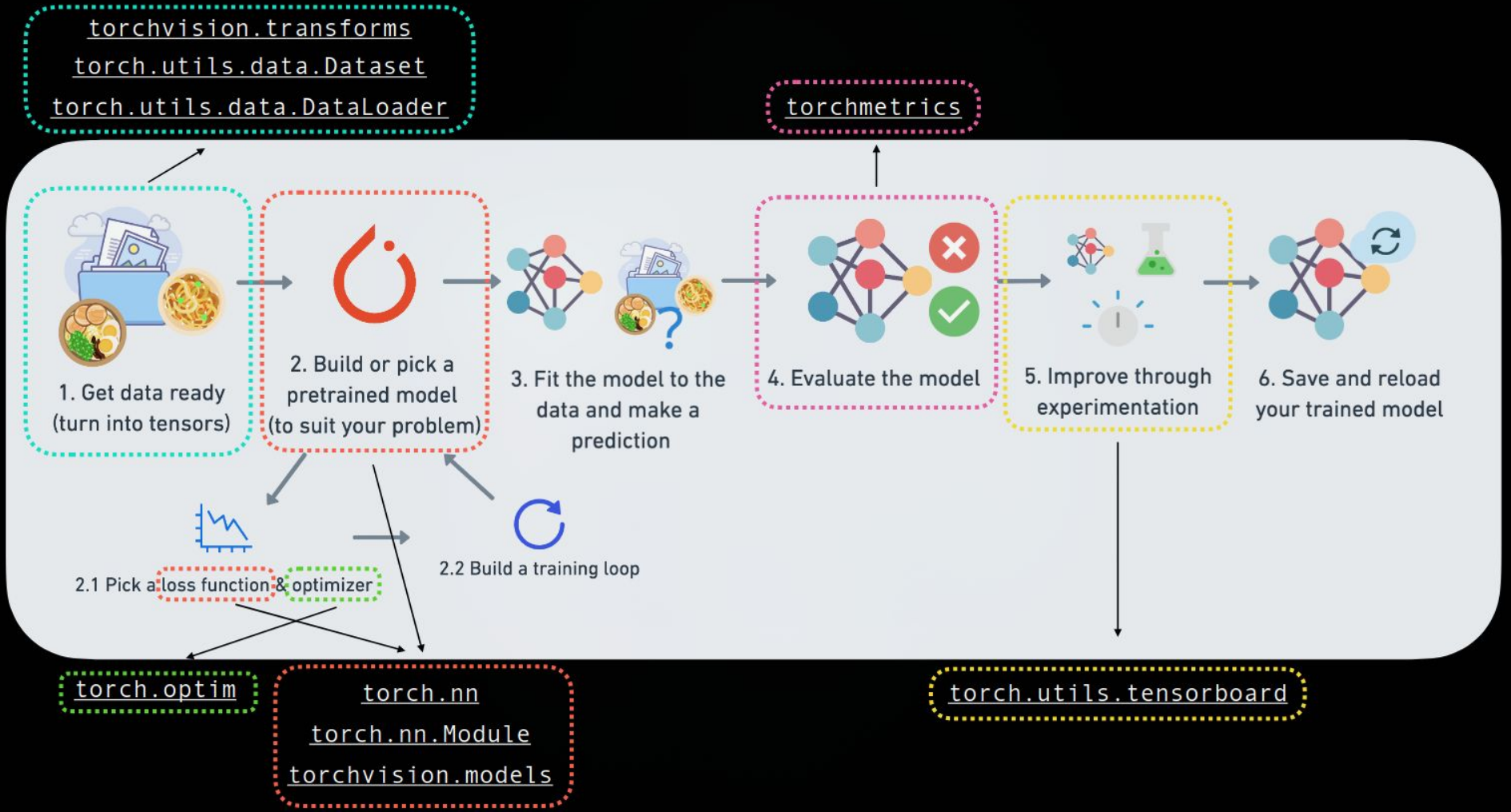
Any subclass of nn.Module needs to override forward() which defines the forward computation of the model.

Looks familiar? $Y = \beta_0 + \beta_1 X$

PyTorch essential modules

PyTorch module	What does it do?
`torch.nn`	Contains all of the building blocks for computational graphs (essentially a series of computations executed in a particular way).
`torch.nn.Module`	The base class for all neural network modules, all the building blocks for neural networks are subclasses. If you're building a neural network in PyTorch, your models should subclass `nn.Module`. Requires a `forward()` method be implemented.
`torch.optim`	Contains various optimization algorithms (these tell the model parameters stored in `nn.Parameter` how to best change to improve gradient descent and in turn reduce the loss).
`torch.utils.data.Dataset`	Represents a map between key (label) and sample (features) pairs of your data. Such as images and their associated labels.
`torch.utils.data.DataLoader`	Creates a Python iterable over a torch Dataset (allows you to iterate over your data).

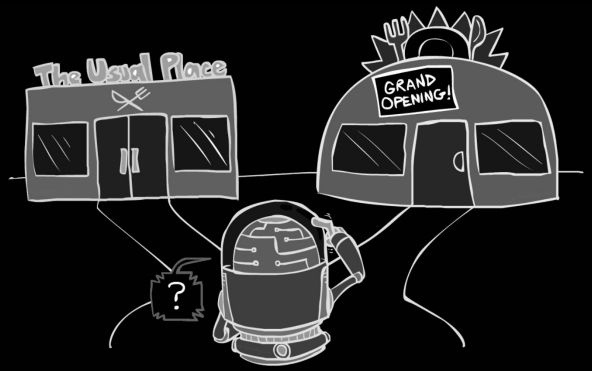
https://pytorch.org/tutorials/beginner/ptcheat.html


torchvision.transforms
torch.utils.data.Dataset
torch.utils.data.DataLoader
torchmetrics
1. Get data ready (turn into tensors)
2. Build or pick a pretrained model (to suit your problem)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your trained model
2.1 Pick a loss function & optimizer
2.2 Build a training loop
torch.optim
torch.nn
torch.nn.Module
torchvision.models
torch.utils.tensorboard

Inference Mode

When we work with Deep Learning models, by default they have autograd enabled which means the model is trying to minimize loss (we'll talk more about gradient descent later). In Artificial Intelligence, this is also called exploration mode, as the model is trying to explore the loss function.

When autograd is turned off, the model switches to Inference mode, where it tries to make predictions. Artificial Intelligence calls this mode exploitation mode.



older versions of pytorch
call this torch.no_grad()

MSE vs MAE loss function

Mean Squared Error has the advantage of finding a solution faster. The squared term magnifies the impact of the error which accelerates minimization. This isn't always a positive as the impact of outliers will now also be greatly magnified which can compromise the stability of the model.

Mean Absolute Error may take longer to converge than MSE, however it treats all terms equally which prevents outliers from being overweight in the model.

Pick the loss function that is appropriate for your data!

04

Classification

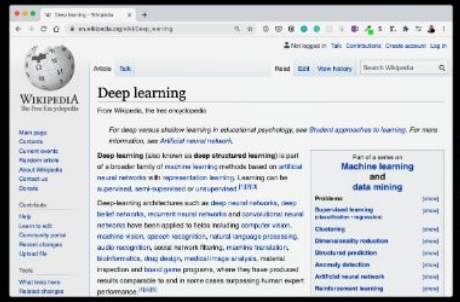
What is a classification problem?

Examples of classification problems:

"Is this a photo of sushi, steak or pizza?"

Multiclass classification

"What tags should this article have?"



Multilabel classification

"Is this email spam or not spam?"

To: rv846@hunter.cuny.edu
Hey Ryan,

My computer is broken, can you check it out when you have a chance?

Not spam

To: rv846@hunter.cuny.edu
hi,

try my game <3<3<3
attached: totally_not_a_virus.exe

Spam

Binary classification

Types of classification

- **Binary classification:** One thing or another (e.g. Cat or Dog)

- **Multiclass classification:** More than one thing or another (e.g. Cat, Dog, or Chicken)

- **Multilabel classification:** Multiple labels per sample

Classification inputs and outputs

W = 224
H = 224
C = 3

Sushi
Steak
Pizza

Actual output

```
[[0.31, 0.62, 0.44…],
[0.92, 0.03, 0.27…],
[0.25, 0.78, 0.07…],
…,
```

Numerical encoding

Inputs

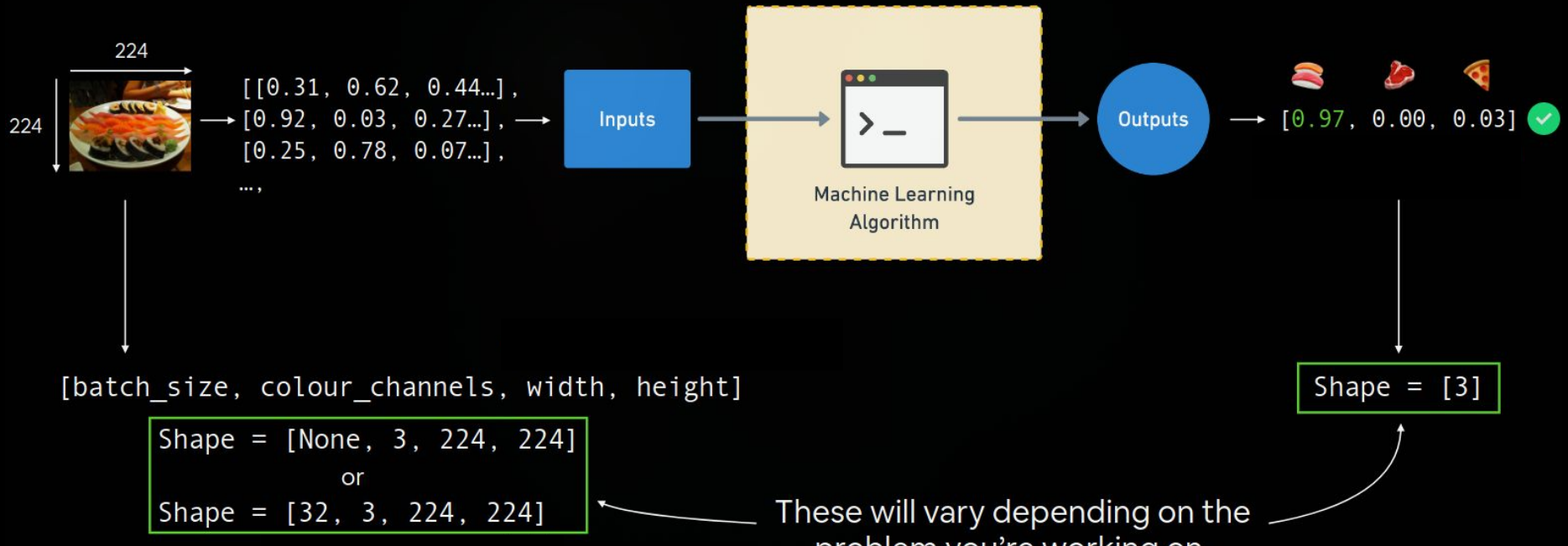
Machine Learning Algorithm

Outputs

```
[[0.97, 0.00, 0.03],
[0.81, 0.14, 0.05],
[0.03, 0.07, 0.90],
…,
```

Predicted output

Input and output shapes



Architecture of a classification model

Hyperparameter	Binary Classification	Multiclass classification
Input layer shape (`in_features`)	Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction)	Same as binary classification
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited	Same as binary classification
Neurons per hidden layer	Problem specific, generally 10 to 512	Same as binary classification
Output layer shape (`out_features`)	1 (one class or the other)	1 per class (e.g. 3 for food, person or dog photo)
Hidden layer activation	Usually ReLU (rectified linear unit) but can be many others	Same as binary classification
Output activation	Sigmoid (`torch.sigmoid` in PyTorch)	Softmax (`torch.softmax` in PyTorch)
Loss function	Binary crossentropy (`torch.nn.BCELoss` in PyTorch)	Cross entropy (`torch.nn.CrossEntropyLoss` in PyTorch)
Optimizer	SGD (stochastic gradient descent), Adam (see `torch.optim` for more options)	Same as binary classification

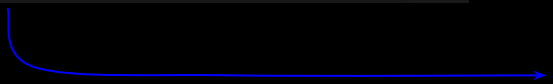
```
# Create a model
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=100),
    nn.linear(in_feature=100, out_feature=100),
    nn.ReLU(),
    nn.Linear(in_features=100, out_features=3)
)

# Set up the loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(params=model.parameters(),
                            lr = 0.001)
```

Common ways to improve a model

```python
# Create a model
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=100),
    nn.linear(in_feature=100, out_feature=100),
    nn.ReLU(),
    nn.Linear(in_features=100, out_features=3)
)

# Set up the loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(params=model.parameters(),
                            lr = 0.001)
```



```python
# Create a model
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=128),
    nn.ReLU(),
    nn.linear(in_feature=128, out_feature=256),
    nn.ReLU(),
    nn.linear(in_feature=256, out_feature=128),
    nn.ReLU(),
    nn.Linear(in_features=128, out_features=3)
)

# Set up the loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.Adam(params=model.parameters(),
                             lr = 0.0001)
```

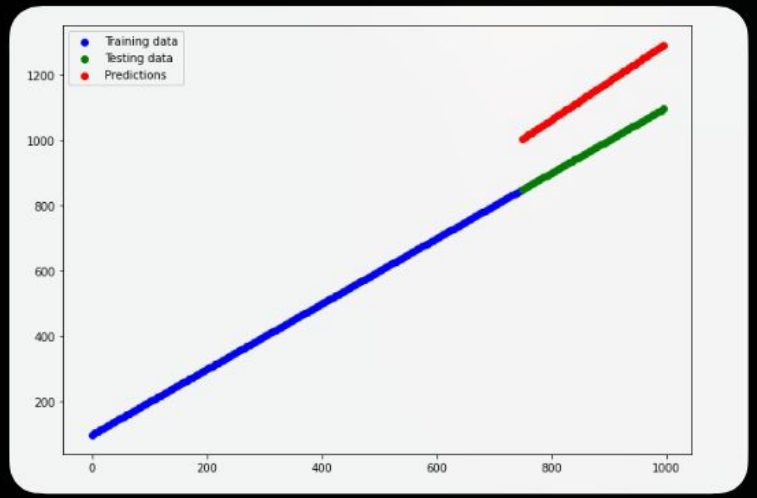
- Adding Layers
- Increase the number of hidden units
- Change/Add activation functions
- Change the optimization function
- Change the learning rate
- Fitting for longer

(these are known as hyperparameters)

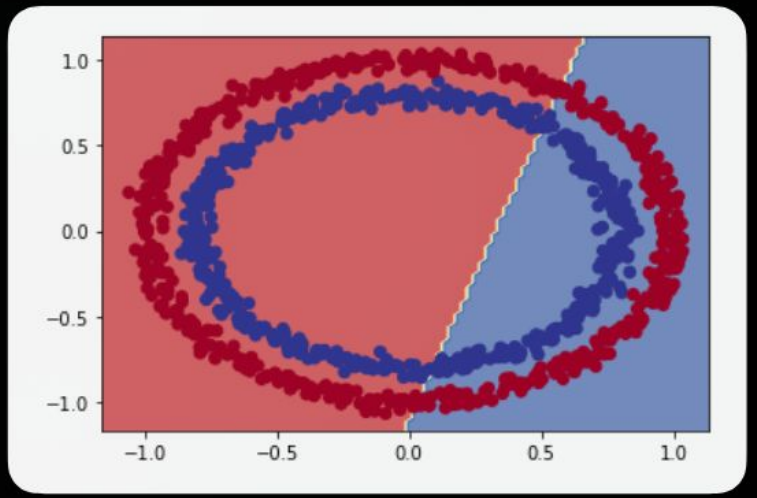
Why does my model still fail?

Non-linearity

Linear regression can fit any data that has a linear trend, but in the real world that is a joke of an assumption. We need a model that doesn't rely in extracting the linear pattern in data



Linear data

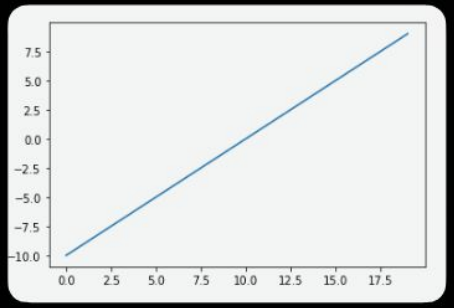


Non-linear data

The missing piece: Non-linearity

```
A = torch.arange(-10, 10)
```

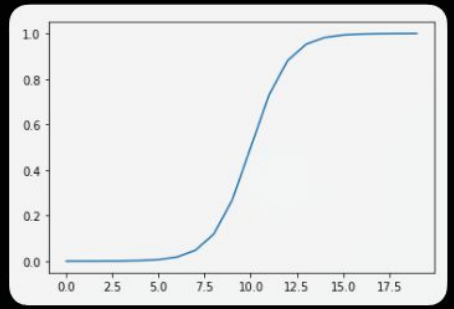
```
plt.plot(A)
```



Linear activation

(same as original values)

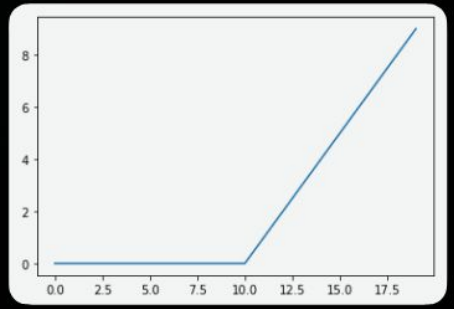
```
plt.plot(torch.sigmoid(A))
```



Sigmoid activation

(non-linear)

```
plt.plot(torch.relu(A))
```



ReLU activation

(non-linear)

Steps in modelling with PyTorch

```python
# Create a model
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=100),
    nn.linear(in_feature=100, out_feature=100),
    nn.ReLU(),
    nn.Linear(in_features=100, out_features=3)
)

# Set up the loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(params=model.parameters(),
                            lr = 0.001)
```

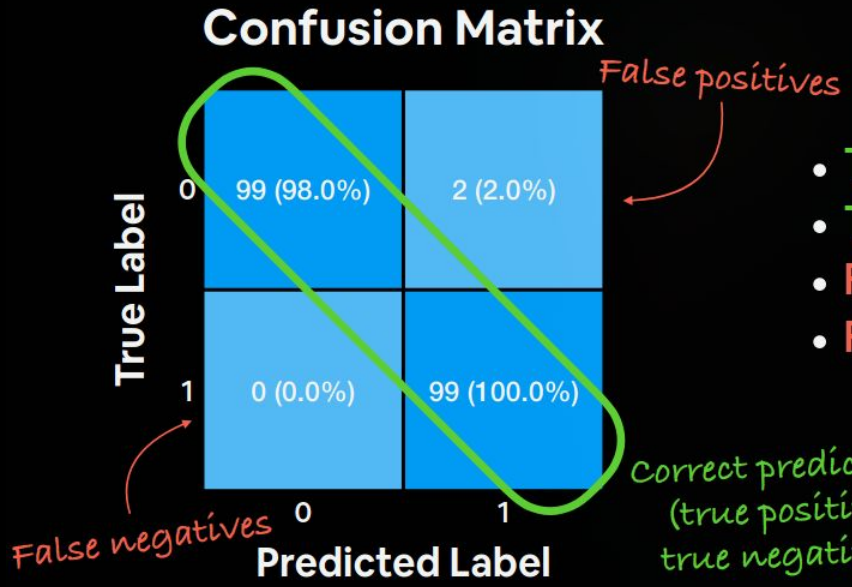
1. Construct or import a pretrained model relevant to your problem
2. Prepare the loss function, optimizer and training loop
    a. Loss - how wrong your model's predictions are compared to the truth labels (we want to minimize this)
    b. Optimizer - how your model should update its internal patterns to better its predictions
3. Fit the model to the training data so it can discover patterns
    a. Epochs - how many times the model will go through all of the training examples
4. Evaluate the model on the test data (how reliable are our model's predictions?

Classification evaluation methods

Key: **tp** = True Positive, **tn** = True Negative, **fp** = False Positive, **fn** = False Negative

Metric Name	Metric Forumla	Code	When to use
Accuracy	$\textbf{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$	`torchmetrics.Accuracy()` or `sklearn.metrics.accuracy_score()`	Default metric for classification problems. Not the best for imbalanced classes.
Precision	$\textbf{Precision} = \frac{tp}{tp + fp}$	`torchmetrics.Precision()` or `sklearn.metrics.precision_score()`	Higher precision leads to less false positives.
Recall	$\textbf{Recall} = \frac{tp}{tp + fn}$	`torchmetrics.Recall()` or `sklearn.metrics.recall_score()`	Higher recall leads to less false negatives.
F1-score	$\textbf{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$	`torchmetrics.F1Score()` or `sklearn.metrics.f1_score()`	Combination of precision and recall, usually a good overall metric for a classification model.
Confusion matrix	NA	`torchmetrics.ConfusionMatrix()`	When comparing predictions to truth labels to see where model gets confused. Can be hard to use with large numbers of classes.

Anatomy of a confusion matrix



- True positive = model predicts 1 when truth is 1
- True negative = model predicts 0 when truth is 0
- False positive = model predicts 1 when truth is 0
- False negative = model predicts 0 when truth is 1

05

Deep Computer Vision

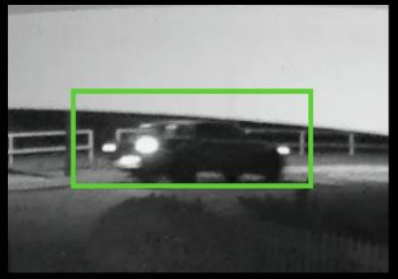
What is a computer vision problem?

Example computer vision problems

"Is this a photo of steak or pizza?"

Binary classification

"Where's the thing we're looking for?"



Object detection

"Is this a photo of sushi, steak or pizza?"

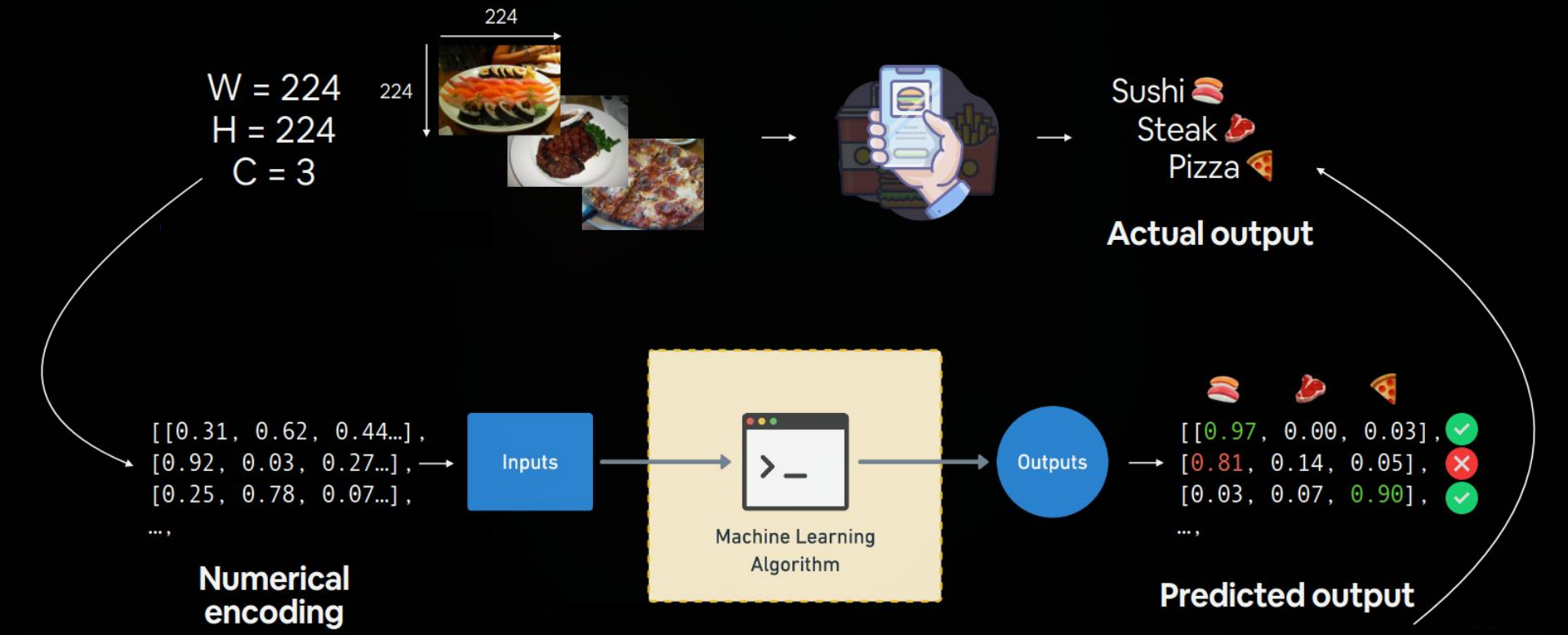
Multiclass classification

"What are the different sections in this image?"

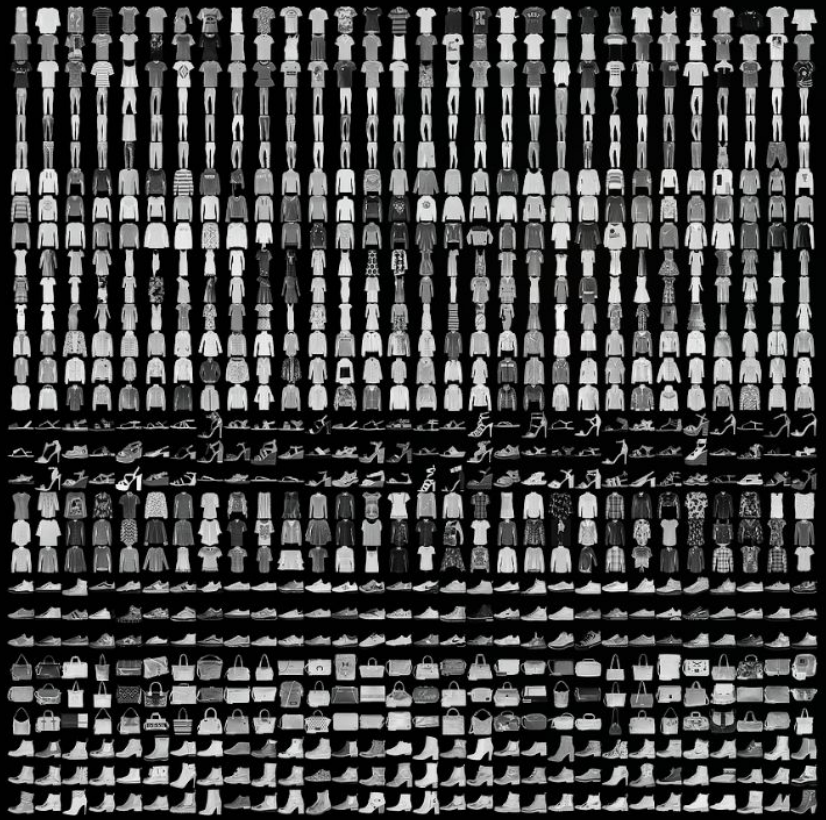
Segmentation

Source: On-device Panoptic Segmentation for Camera Using Transformers.

Computer vision inputs and outputs



FashionMNIST

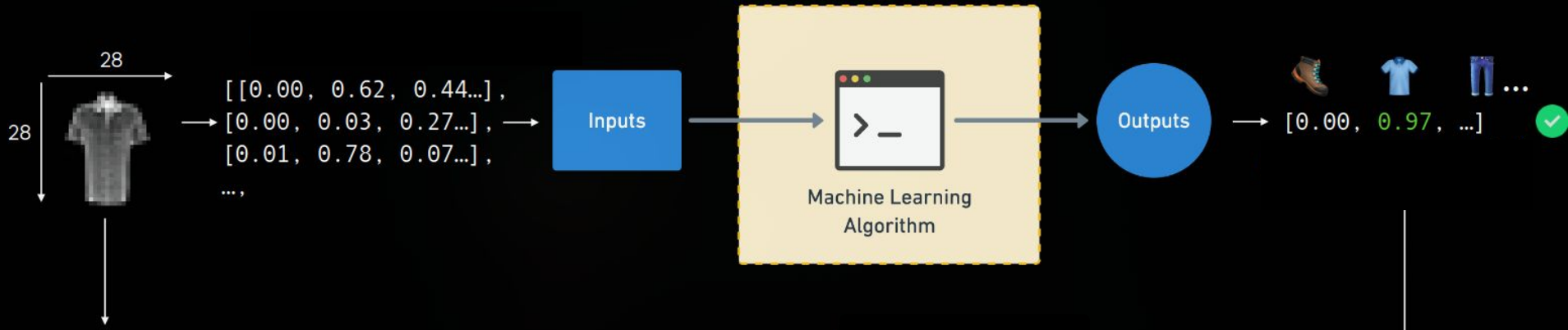


torchvision.datasets.FashionMNIST

"What type of clothing is in this image?"

Multiclass classification

Input and output shapes



`[batch_size, height, width, colour_channels]` (NHWC)

or

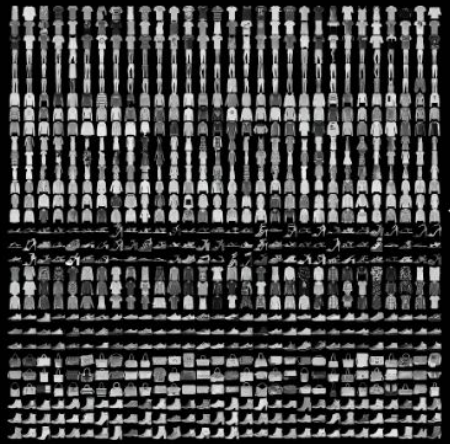
`[batch_size, colour_channels, height, width]` (NCHW)

`Shape = [10]`

```
Shape = [None, 28, 28, 1] (NHWC)
Shape = [None, 1, 28, 28] (NCHW)
        or
Shape = [32, 28, 28, 1]
```

These will vary depending on the problem you're working on.

FashionMNIST: Batched



torchvision.datasets.FashionMNIST

```
1 from torch.utils.data import DataLoader
2 train_dataloader = DataLoader(dataset=train_data,
3                               batch_size=32,
4                               shuffle=True)
```

torch.utils.data.DataLoader

batch_size=32
(32 samples per batch)

	Sample 0	1	2	3	4	5	32
Batch 0							
1							
2							
3							
4							
⋮							

shuffle=True
(samples all mixed up)

Num samples/
batch_size

Architecture of a CNN

Hyperparameter/Layer type	What does it do?	Typical values
Input image(s)	Target images you'd like to discover patterns in	Whatever you can take a photo (or video) of
Input layer	Takes in target images and preprocesses them for further layers	`input_shape = [batch_size, image_height, image_width, color_channels]` (channels last) or `input_shape = [batch_size, color_channels, image_height, image_width]` (channels first)
Convolution layer	Extracts/learns the most important features from target images	Multiple, can create with `torch.nn.ConvXd()` (X can be multiple values)
Hidden activation/non-linear activation	Adds non-linearity to learned features (non-straight lines)	Usually ReLU (`torch.nn.ReLU()`), though can be many more
Pooling layer	Reduces the dimensionality of learned image features	Max (`torch.nn.MaxPool2d()`) or Average (`torch.nn.AvgPool2d()`)
Output layer/linear layer	Takes learned features and outputs them in shape of target labels	`torch.nn.Linear(out_features=[number_of_classes])` (e.g. 3 for pizza, steak or sushi)
Output activation	Converts output logits to prediction probabilities	`torch.sigmoid()` (binary classification) or `torch.softmax()` (multi-class classification)

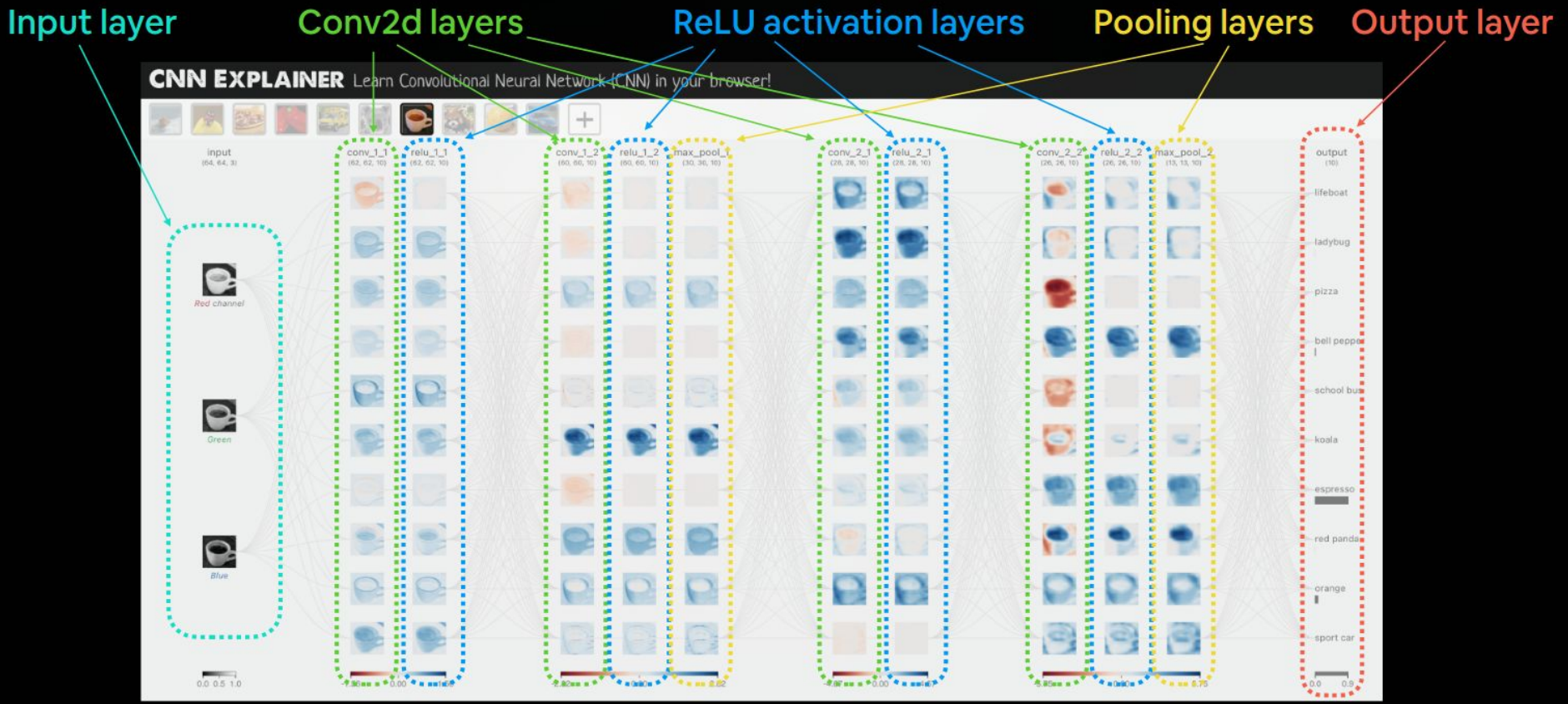
```python
# Create a Convolutional Neural Network
import torch
from torch import nn
class CNN_model(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.cnn_layers = nn.Sequential(
            nn.Conv2d(in_channels=input_shape,
                      out_channels=hidden_units,
                      kernel_size=3, # how big is the square that's going over the image?
                      stride=1, # take a step one pixel at a time
                      padding=1), # add an extra pixel around the input image
            nn.ReLU(), # non-linear activation
            nn.MaxPool2d(kernel_size=2,
                         stride=2) # default stride value is same as kernel_size
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=hidden_units * 32 * 32, # same shape as output of self.cnn_layers
                      out_features=output_shape)
        )

    def forward(self, x: torch.Tensor):
        x = self.cnn_layers(x)
        x = self.classifier(x)
        return x

cnn_model = CNN_model(input_shape=3, # same as number of input color channels
                      hidden_units=10,
                      output_shape=3) # same as number of classes
```

Steak 🥩
Pizza 🍕
Sushi 🍣

CNN Explainer model



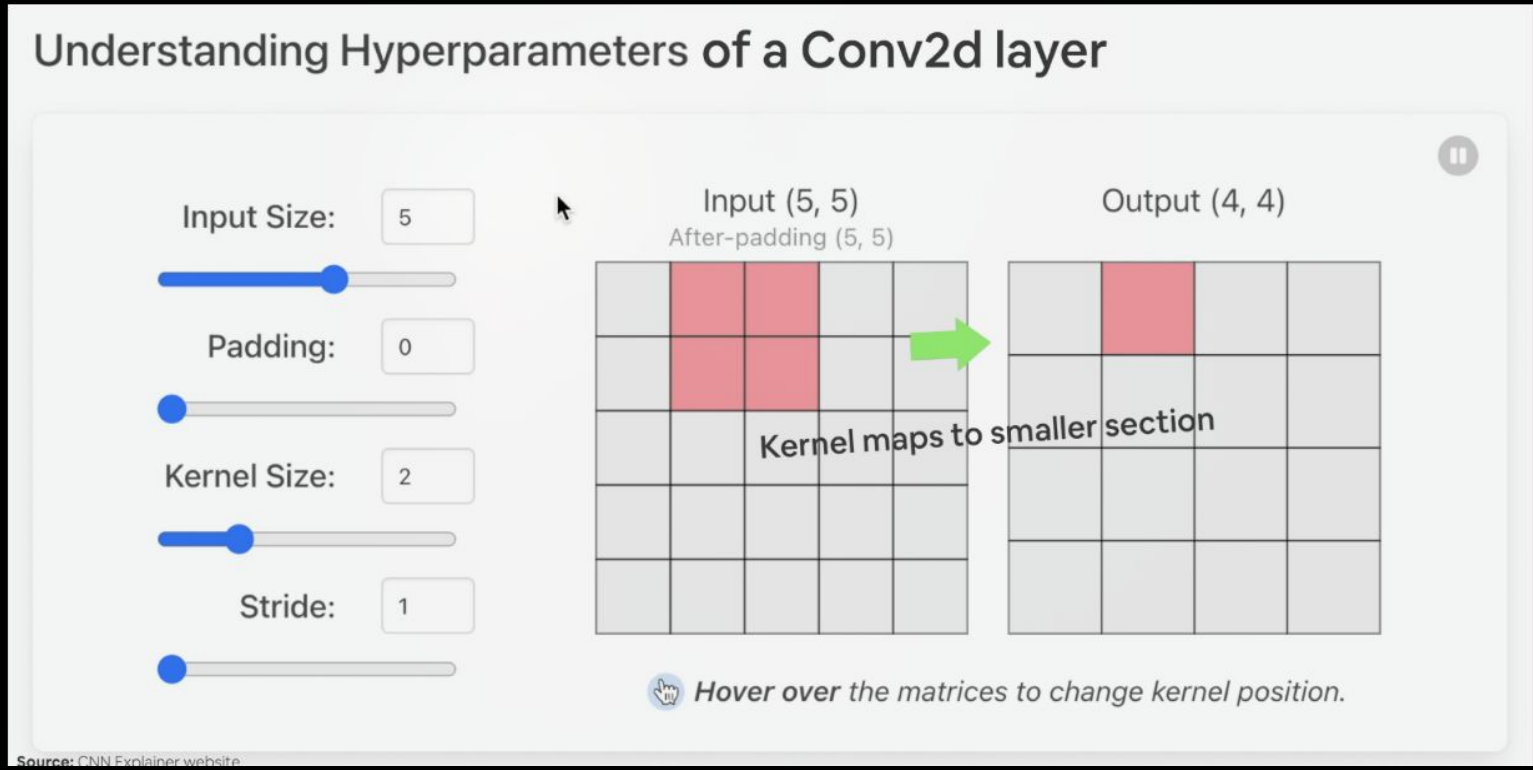
Source: https://poloclub.github.io/cnn-explainer/

Breakdown of torch.nn.Conv2d layer

Example code: `torch.nn.Conv2d(in_channels=3, out_channels=10, kernel_size=(3, 3), stride=(1, 1), padding=0)`
Example 2 (same as above): `torch.nnConv2d(in_channels=3, out_channels=10, kernel_size=3, stride=1, padding=0)`

Hyperparameter name	What does it do?	Typical values
`in_channels`	Defines the number of input channels of the input data.	`1 (grayscale), 3 (RGB color images)`
`out_channels`	Defines the number output channels of the layer (could also be called hidden units).	`10, 128, 256, 512`
`kernel_size` (also referred to as filter size)	Determines the shape of the kernel (sliding windows) over the input.	`3, 5, 7 (lowers values learn smaller features, higher values learn larger features)`
`stride`	The number of steps a filter takes across an image at a time (e.g. if `strides=1`, a filter moves across an image 1 pixel at a time).	`1 (default), 2`
`padding`	Pads the target tensor with zeroes (if "`same`") to preserve input shape. Or leaves in the target tensor as is (if "`valid`"), lowering output shape.	`0, 1, "same", "valid"`

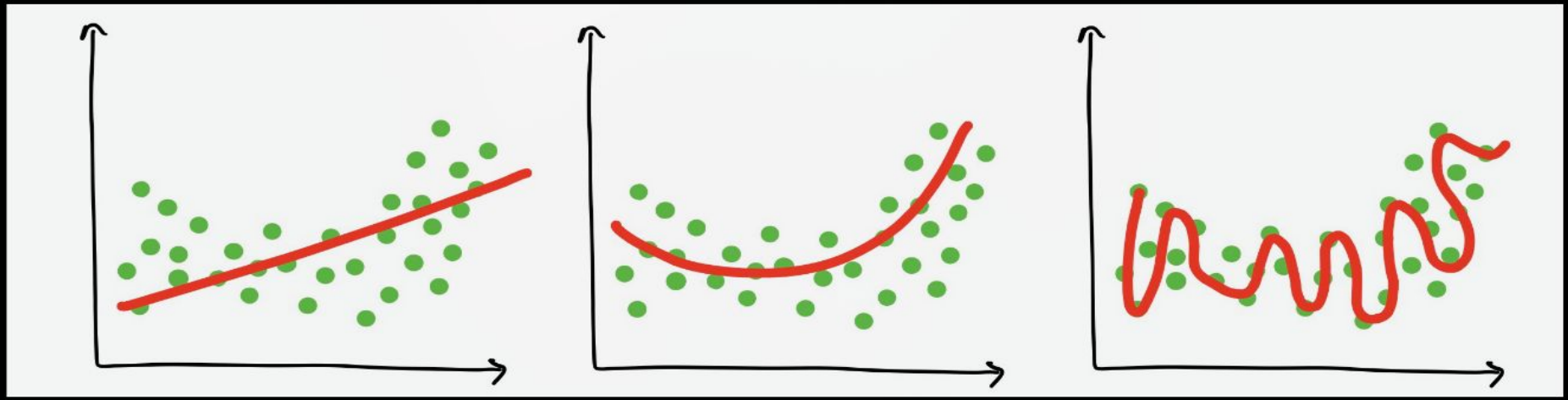
Breakdown of torch.nn.Conv2d layer



What is overfitting (in DL)

Overfitting — when a model over learns patterns in a particular dataset and isn't able to generalise to unseen data.

For example, a student who studies the course materials too hard and then isn't able to perform well on the final exam. Or tries to put their knowledge into practice at the workplace and finds what they learned has nothing to do with the real world.



Improving a model

Method to improve a model (reduce overfitting)	What does it do?
More data	Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza).
Data augmentation	Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns.
Better data	Not all data samples are created equally. Removing poor samples from or adding better samples to your dataset can improve your model's performance.
Use transfer learning	Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.

What is data augmentation?

Looking at the same image but from different perspective(s)*.

Original

Rotate

Shift

Zoom

Popular Computer Vision Models

Architecture	Release Date	Paper	Use in PyTorch	When to use
ResNet (residual networks)	2015	https://arxiv.org/abs/1512.03385	torchvision.models.resnet...	A good backbone for many computer vision problems
EfficientNet(s)	2019	https://arxiv.org/abs/1905.11946	torchvision.models.efficientnet...	Typically now better than ResNets for computer vision
Vision Transformer (ViT)	2020	https://arxiv.org/abs/2010.11929	torchvision.models.vit_...	Transformer architecture applied to vision
MobileNet(s)	2017	https://arxiv.org/abs/1704.04861	torchvision.models.mobilenet...	Lightweight architecture suitable for devices with less computing power

Let’s get coding