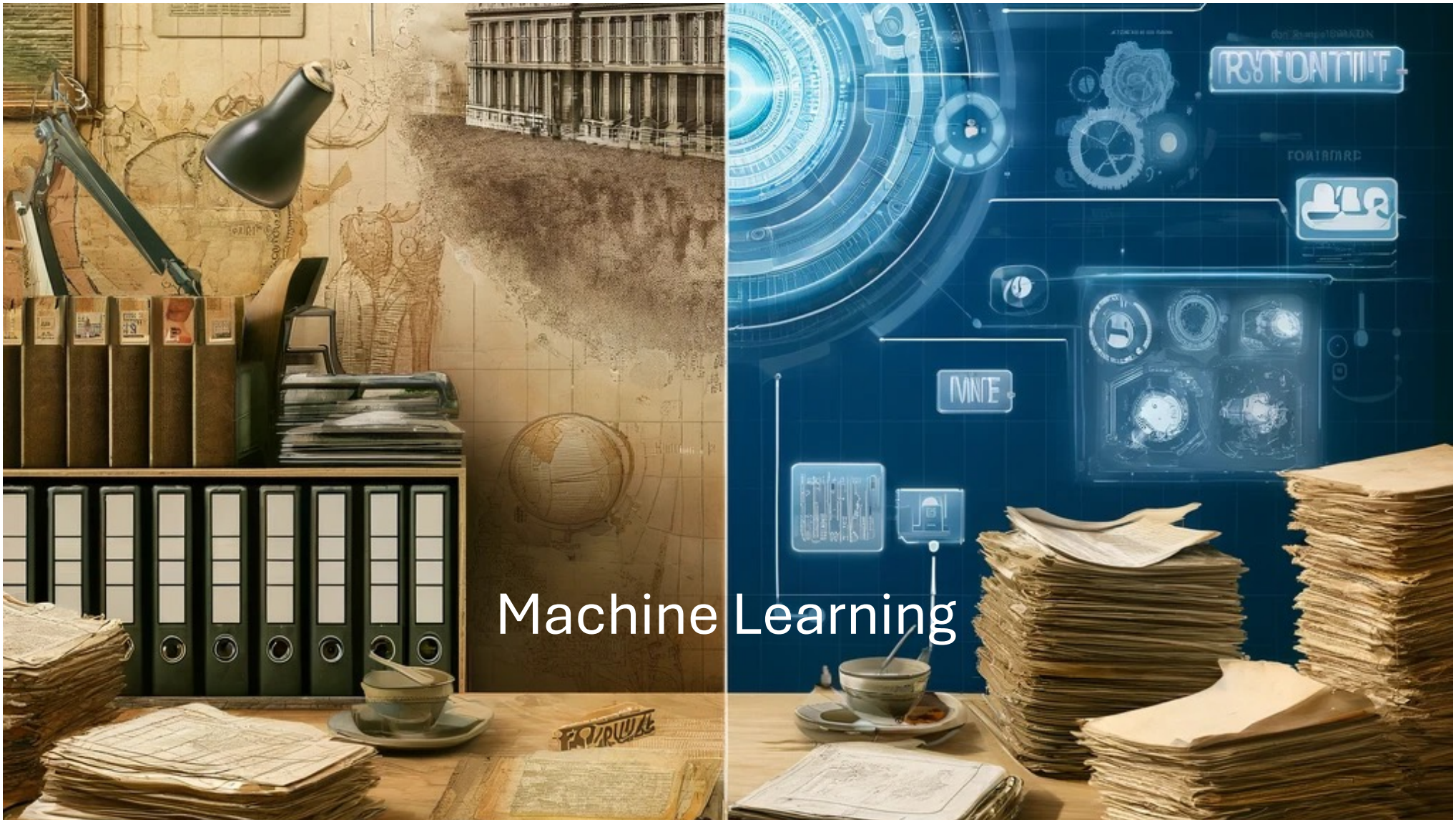
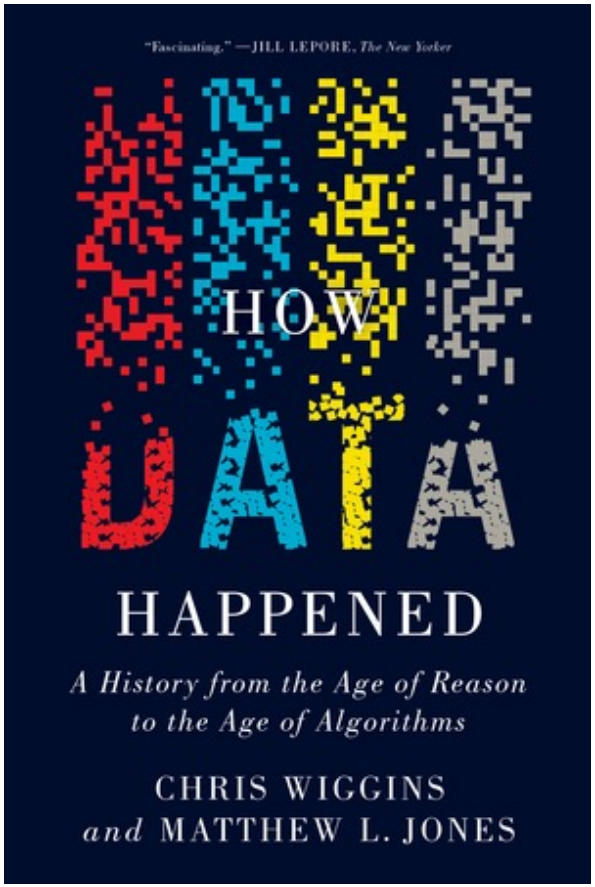

Machine Learning

What we'll cover today

- History of machine learning
- Epistemology of machine learning
- How can we use it for social scientific research?
- Under the hood of important ML techniques

History of machine learning

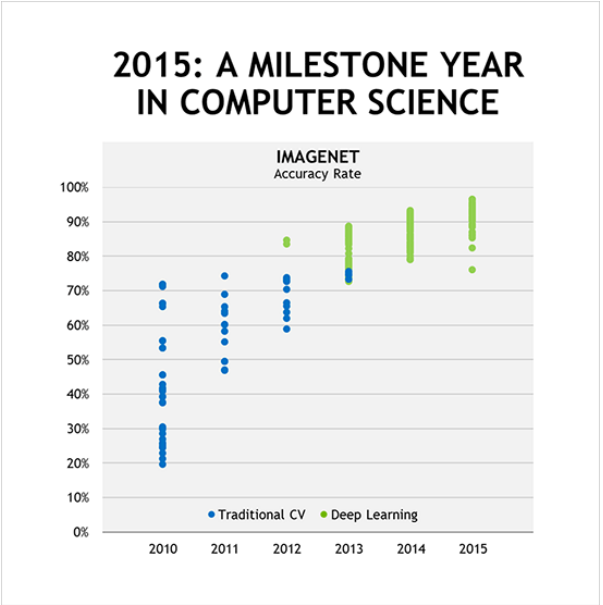
- Traces its origins back to the mid-20th century, early pioneers like Alan Turing.
- NSA pursued Signal Intelligence (SIGINT) Analysis: had a lot of data, and less interested in “it works” than “doing it right”
- The field gained momentum in the 1950s and 1960s with the development of simple neural networks and the exploration of algorithms like the perceptron.
- "AI winters" of the 1970s and 1980s due to limited computational power and high expectations.
- The revival of machine learning came in the late 1980s and early 1990s, fueled by the advent of more powerful computers.

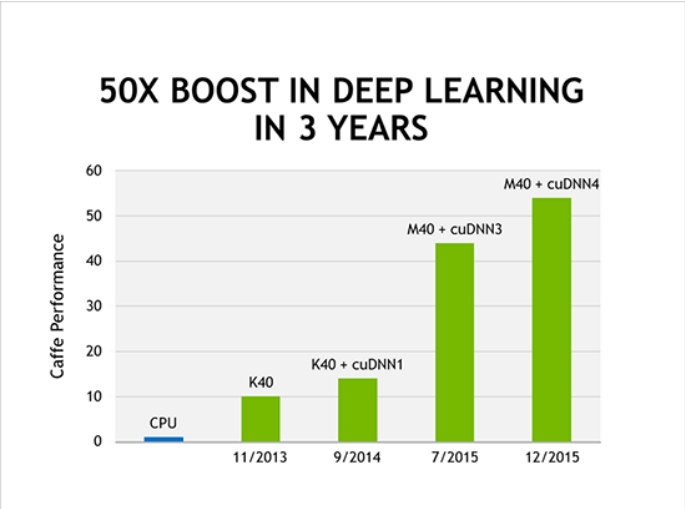


“Big Bang of Machine Learning”

- GPU processing + Massive data from platforms + Deep learning.
- Powerful deep Artificial Neural Networks.

- 2022 ChatGPT revolution: transformers and all text on the internet.





Machine learning and Surveillance Capitalism

- Economic dimension of the “Big Bang”
- Machine learning become the foundation for platforms:
  - Online targeted advertising.
  - Behavioral modification.
  - Maximizing engagement.
- These are based on prediction logic – they don’t care why it works.
- “Will this individual buy the product if we show them the advertisement?”

Machine learning is what makes your personal data valuable. This is what makes machine learning valuable.

What is machine learning?

- Boundary between statistics and machine learning is blurry
- Some methods are both, such as linear regression
- Less about techniques than about the culture and aims

$$Y = f(X) = X^T\beta,$$

- ML focuses on , statistics focuses on .

The Two Cultures

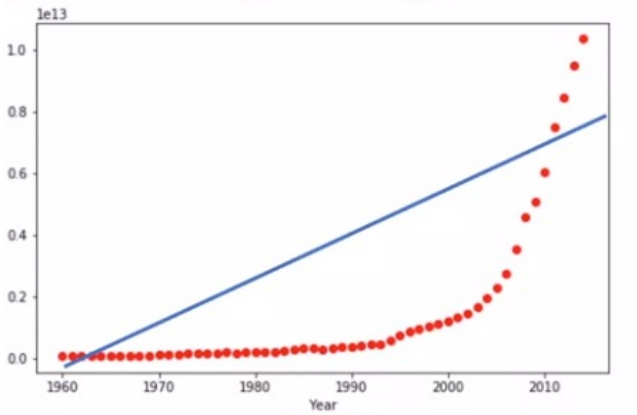
- Statistics:
  - *Understand* how an outcome is related to inputs.
  - Test theory

- Machine learning:
  - *Predict* new unseen data
  - Models don't need to be understandable
  - Inductive

Machine learning	Statistics
network, graphs	model
weights	parameters
learning	fitting
generalization	test set performance
supervised learning	regression/classification
unsupervised learning	density estimation, clustering
large grant = $1,000,000	large grant = $50,000
nice place to have a meeting: Snowbird, Utah, French Alps	nice place to have a meeting: Las Vegas in August

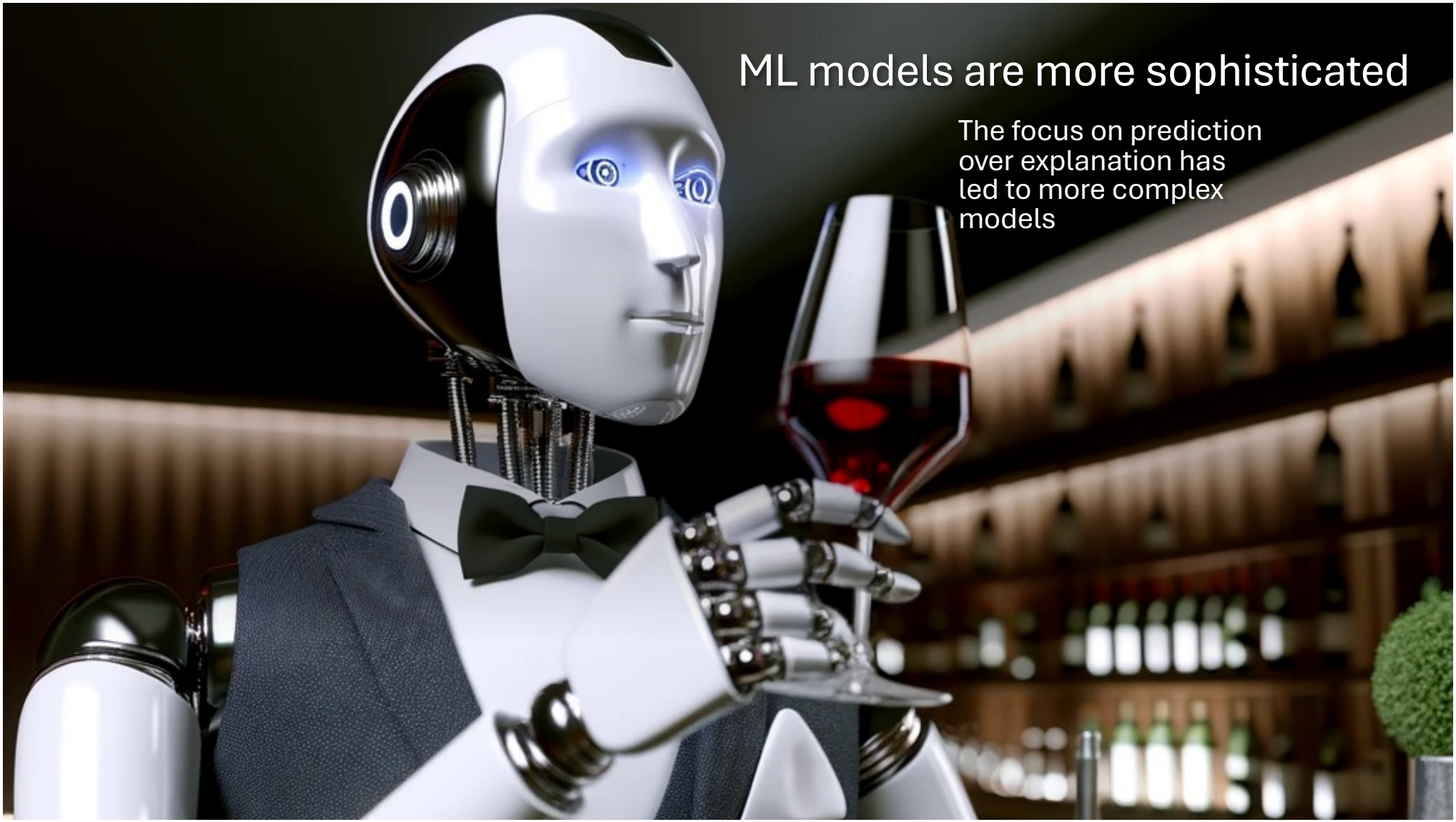
Advantages with Machine Learning

Reality is not linear!



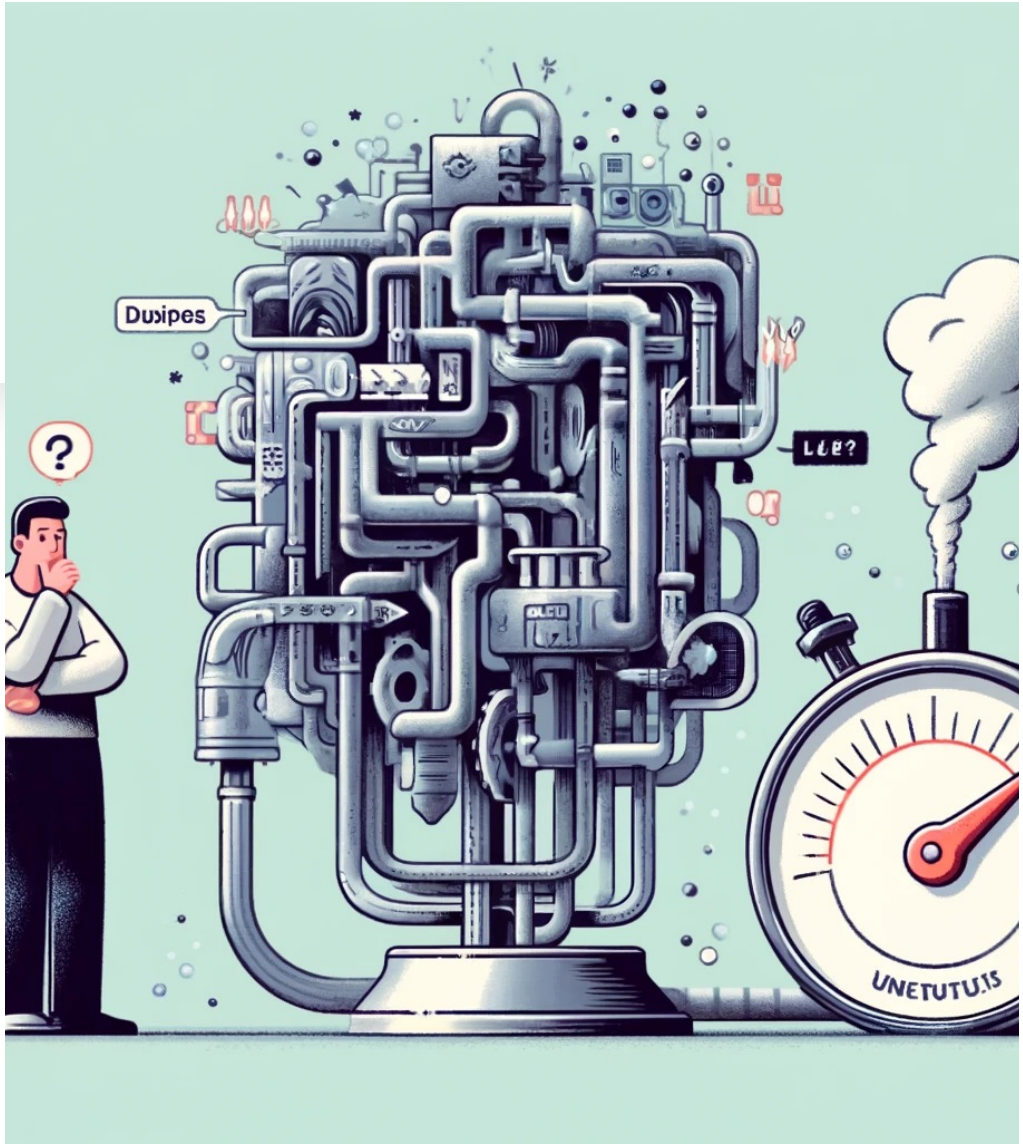
Population heterogeneity and causal complexity

- A cause doesn't always have the same effect
- Social scientific theory usually contains "if's" and "then's", but they cannot be captured by quantitative models.
- *On average* is often a terribly destructive way of dealing with data


ML models are more sophisticated
The focus on prediction over explanation has led to more complex models

Downsides of ML

- The predictions can be an opaque function of the inputs: hard to answer the *why?*
- The models may not easily provide meaningful estimates of uncertainty
- Estimation often more computationally intensive



Two classes of machine learning models

Unsupervised models

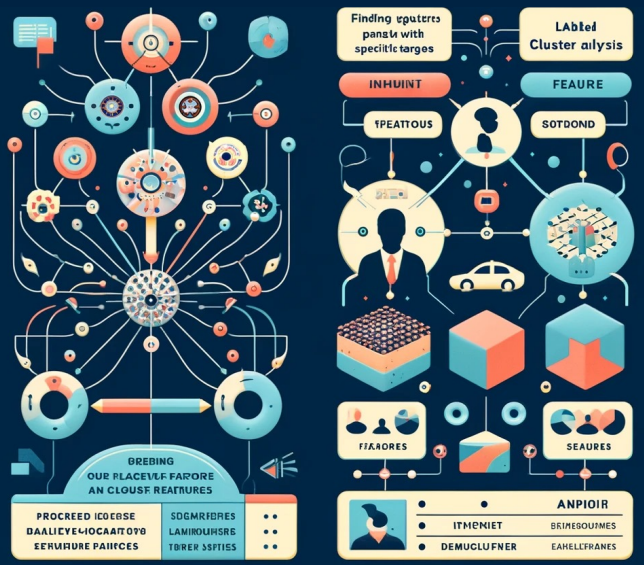
Identify pattern in data, or reduce dimensions.

- No target output to predict, no teacher showing the algorithm what it should aim for, and no immediate measure of success
- E.g. topic modeling, cluster analysis.

Supervised models

Mapping between features and outcomes.

- Uses labelled training data.
- E.g. identifying race from profile picture.



Reinforcement learning: learn-by-doing

- Train AI for skills we don’t fully understand: we know how to walk, but it’s awfully hard to explain.

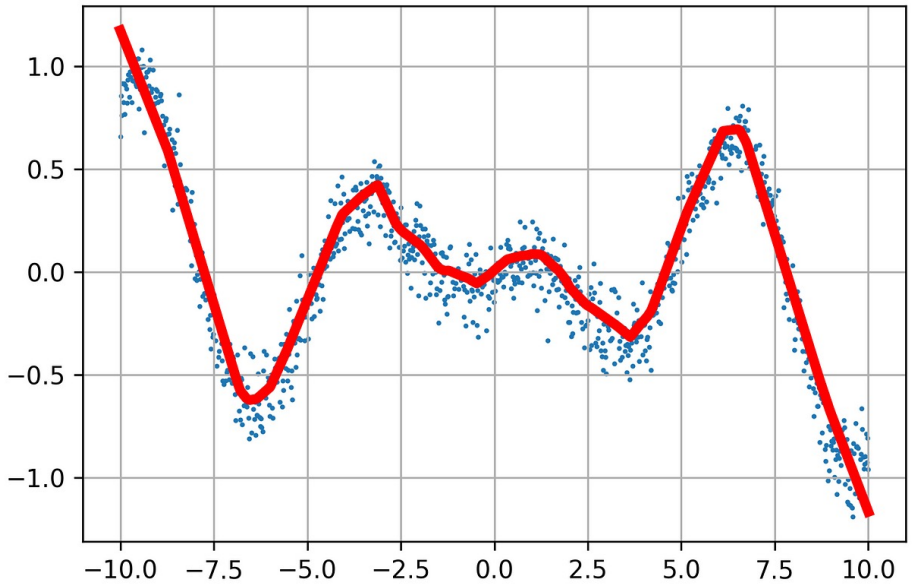
- Trial-and-error
- The better the AI walks, the more reward they get.
- The learning can be implemented in many different ways

Fitting supervised models

Optimization of parameters with respect to an objective function.
Often uses the good-old OLS model (Ordinary Least Squares)

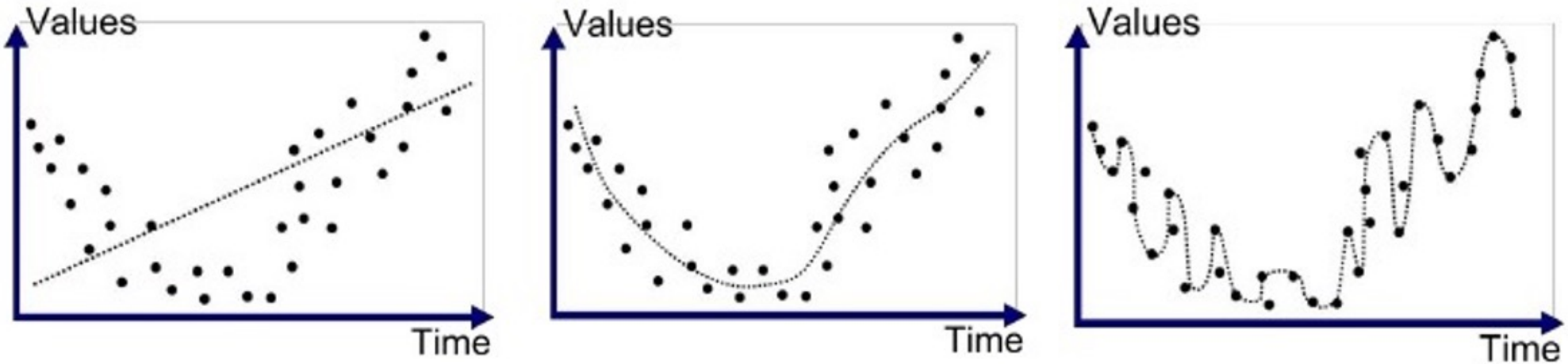
$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\arg\min} \sum_{i}^{m} (y_i - \boldsymbol{\beta}' \boldsymbol{x}_i)^2.$$

What distinguishes machine learning methods is that they can fit substantially more flexible functions.



Assessing model performance

Prediction error within the data set is not a good indicator of prediction in new data

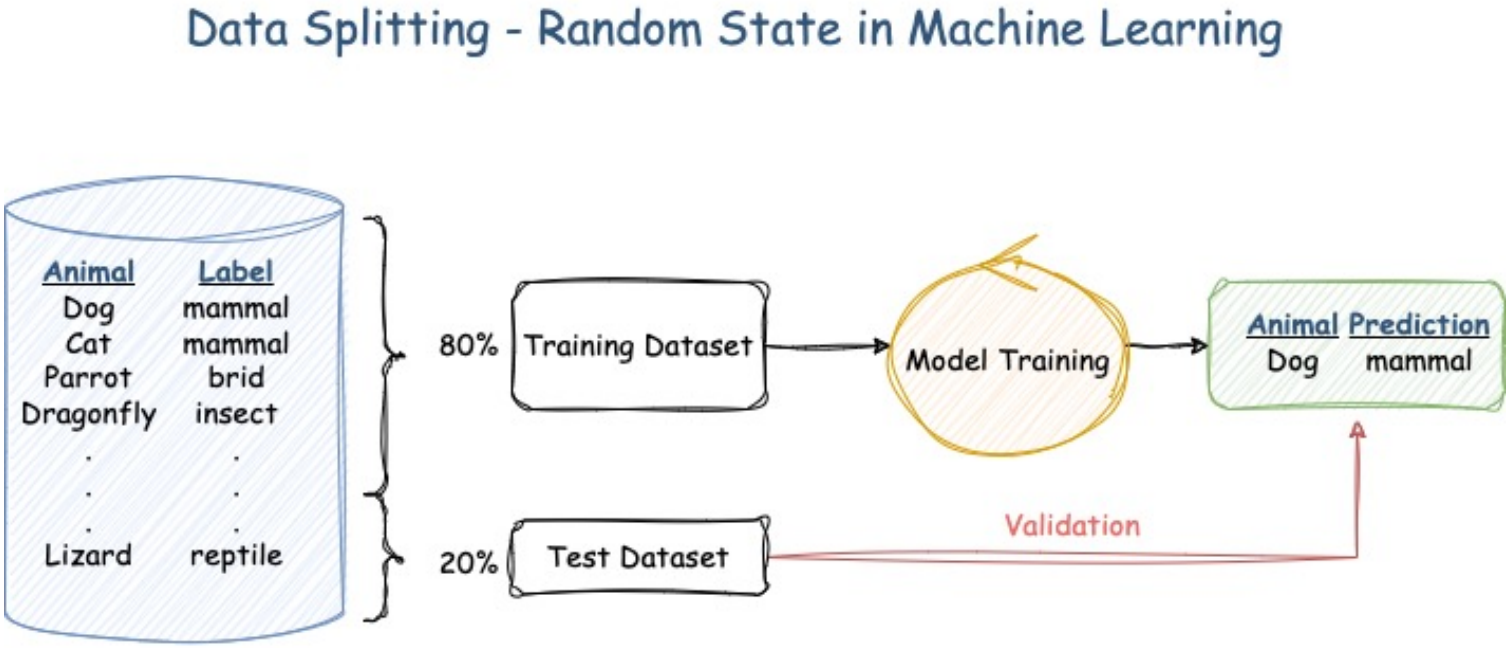


Underfitted

Good Fit/Robust

Overfitted

Sample splitting



How to train/test split in python

```
# Split the dataset into training and testing sets
# The text is what we use to try to predict the device.
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['device'], test_size=0.3, random_state=42)
```

Regularization: Keep it simple, stupid model

We can punish the model for being complex, as a way of keeping it as simple as possible.

- This is called regularization.

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \underbrace{\sum_i^n \left(y_i - \boldsymbol{\beta}' \boldsymbol{x}_i\right)^2}_{\text{sum of squared residuals}} + \overbrace{\lambda \sum_{j=1}^p |\beta_j|^q}^{\text{complexity penalty}}$$

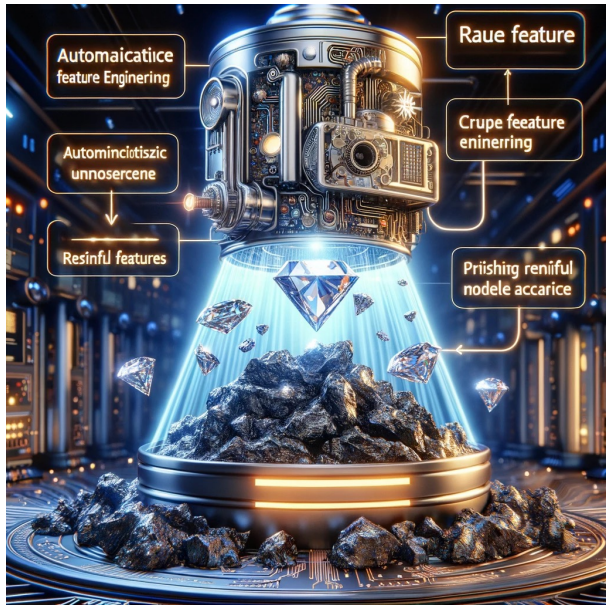
- λ: strength of penalty
- q: the type of regularizer (q=1 LASSO sets many coefficients to zero; q=2 Ridge regression, smooth push to zero)

Hyperparameter search

- What λ parameter will lead to the best results?
- We can optimize the parameters themselves!

Automatic feature engineering

- What variables/features should be included in prediction? How to preprocess?
- How to measure of latent concept like ideology or economic growth?

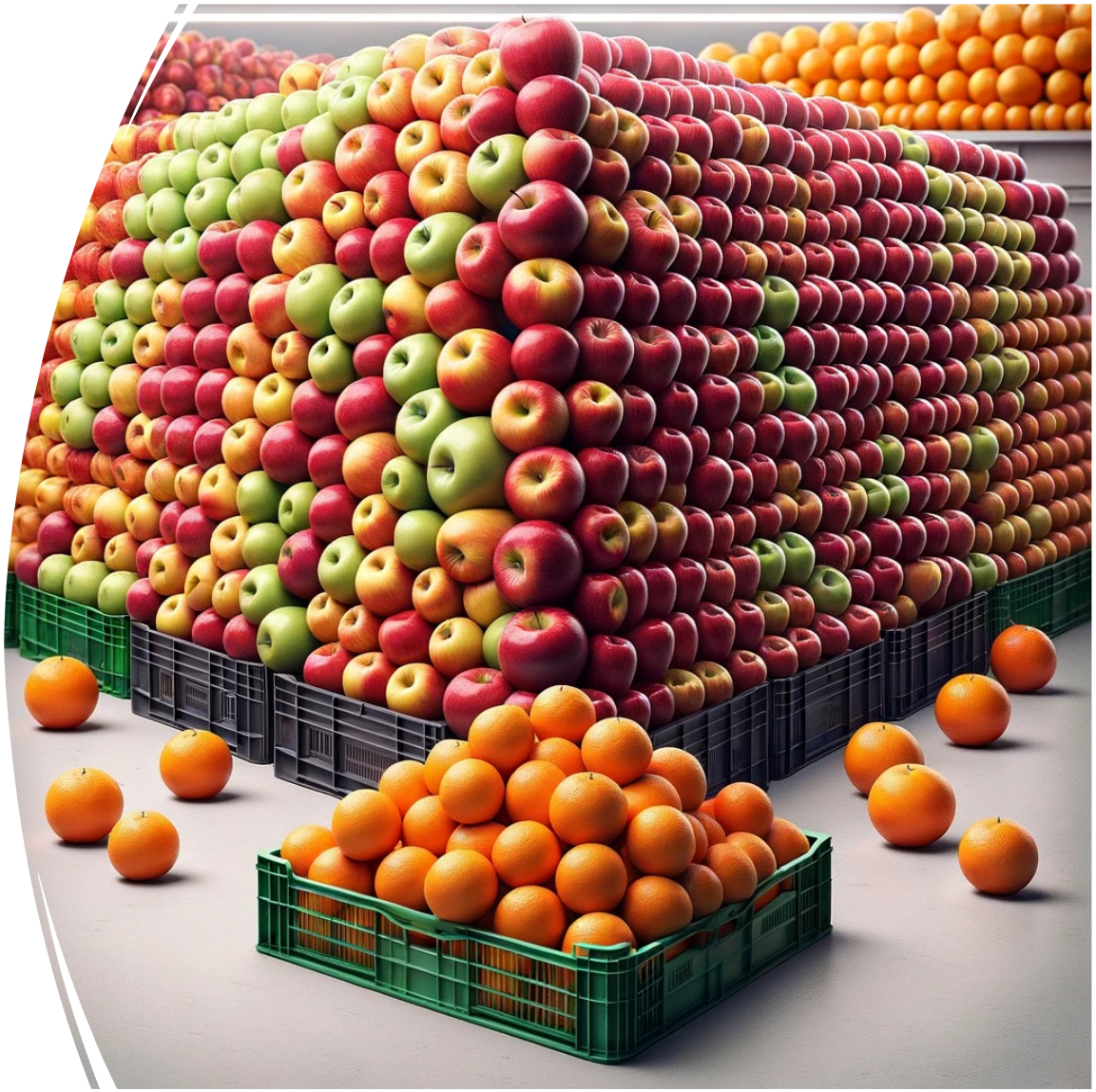


- Machine learning can figure it out automatically.
- Especially important in e.g. vision or sound, where a pixel says nothing on its own.

What does this mean for theory?

Evaluating performance

- You have pictures of 500 apples and 50 oranges.
- You apply a machine learning classification model?
- Your model reaches 92% accuracy.
- Is it a good model?



Performance metrics

Accuracy: Proportion that is correct. Misleading for imbalanced datasets.

Precision: The proportion of *positive* identifications that were actually correct.
Important when the cost of false positives is high (e.g., spam detection).

Recall: The proportion of actual positives that were correctly identified.
Crucial when it's important to capture all positives (e.g., disease screening).

F1-score: The harmonic mean of precision and recall.

Macro: combines the scores from both classes.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

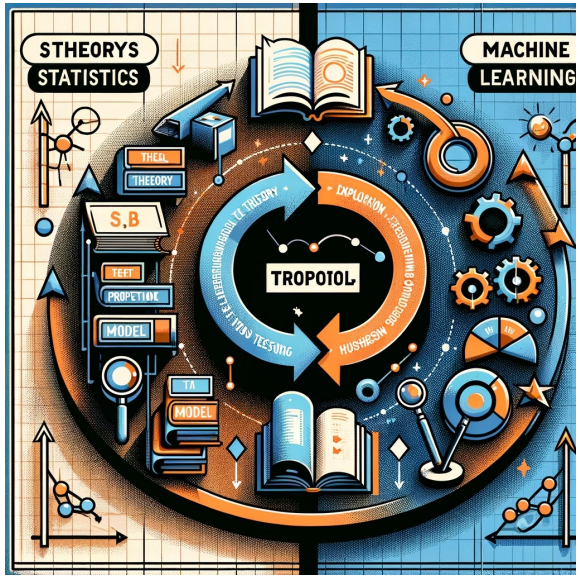
	precision	recall	f1-score	support
0	0.77	0.86	0.81	37584
1	0.84	0.75	0.79	37577
accuracy			0.80	75161
macro avg	0.81	0.80	0.80	75161
weighted avg	0.81	0.80	0.80	75161

What can we actually use ML for in the social sciences?

The epistemology of machine learning

1. From *variables* to *patterns*
2. From *rules* to *associations*
3. From *surveys* to *sensors*

Machine Learning as a social science approach



Statistics

Deductive. Linear approach: Derive testable propositions from clear theory, test using models.

Machine learning

Abductive. Grounded theory. Iterative: explore and develop theory.

Generate the research questions, concepts, and hypotheses that can later be rigorously tested with new data.

Improved regression models

- We can use ML as a form of statistics on steroids, with more complex modeling capabilities, handling non-linear relationships, and managing high-dimensional data
- **Interpretable models**: Some machine learning models, such as linear regression with regularization (like Lasso and Ridge), still allow for interpretation. (Yay, regression tables!)
- There are also ways to make sophisticated models interpretable. Bootstrapping can be used to estimate confidence intervals and standard errors.
- However, might not always provide exact p-values or the traditional statistics familiar from OLS regression.

Coding and classification

One of the most widely used tools in the social sciences is manually classifying observations into a set of categories that are determined before the analysis begins.

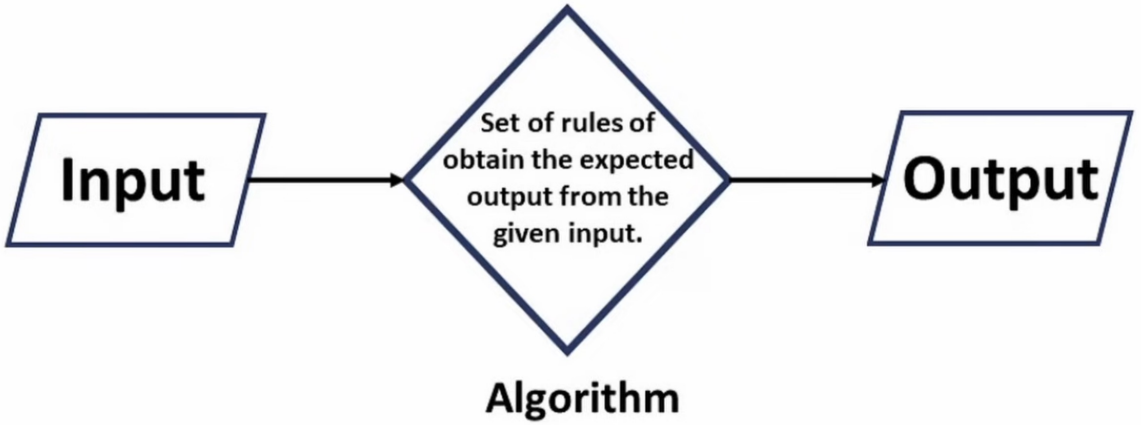
The models allow measuring virtually any latent aspect of data!

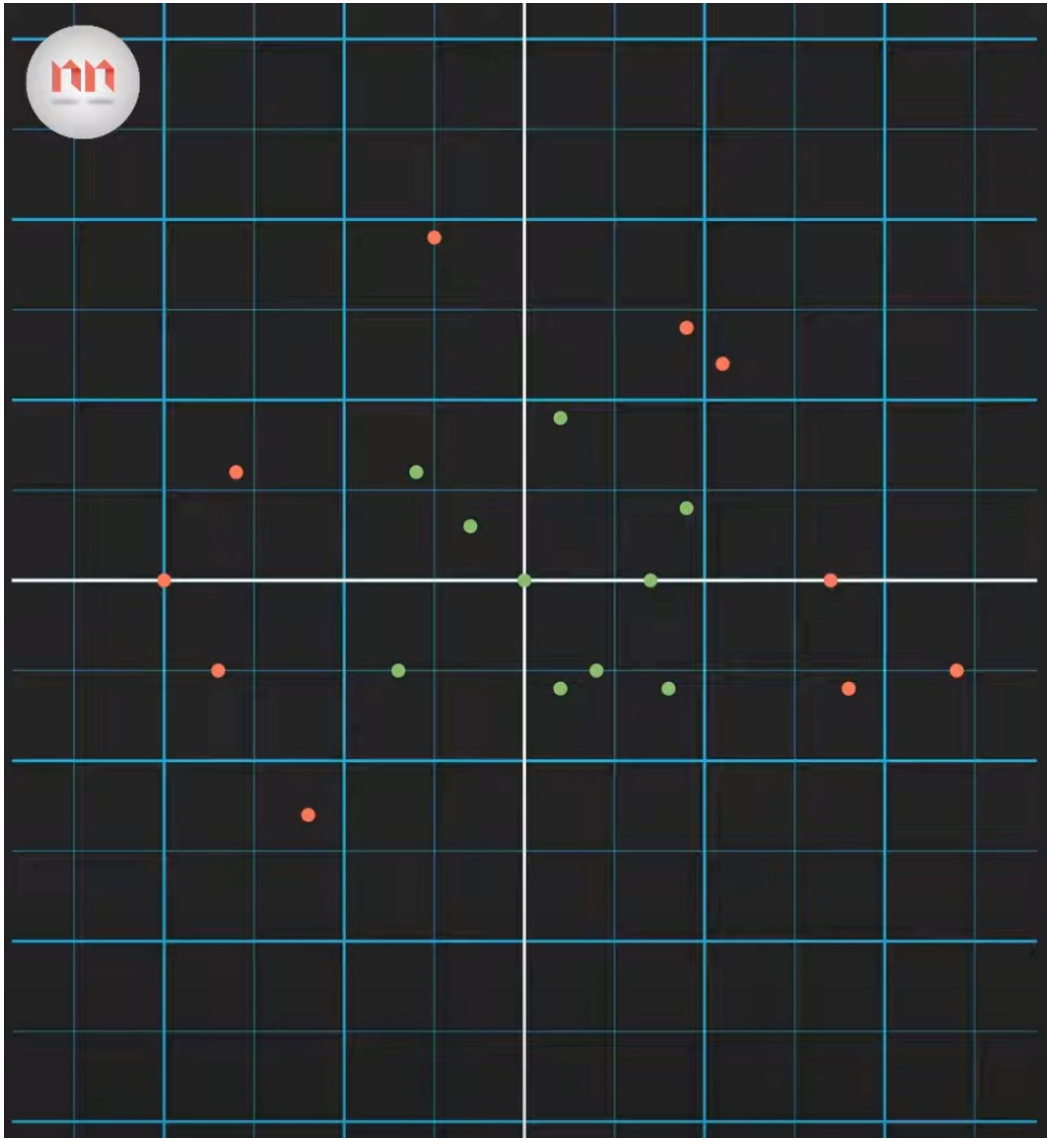
Causal inference from observational data

- Imagine we're interested in studying the effect of attending a private (treatment) vs a public high school (control) on college admission rates (outcome).
- Obviously, the problem is confounding factors. (E.g., having rich parents.)
- We calculate *the propensity score:* the probability of a student attending a private high school given their observed characteristics (covariates). We estimate this probability using statistical or machine learning models
- We match students based on their propensity scores.
- The effect of treatment is then = the average difference between the matched individuals

How do the models actually work?

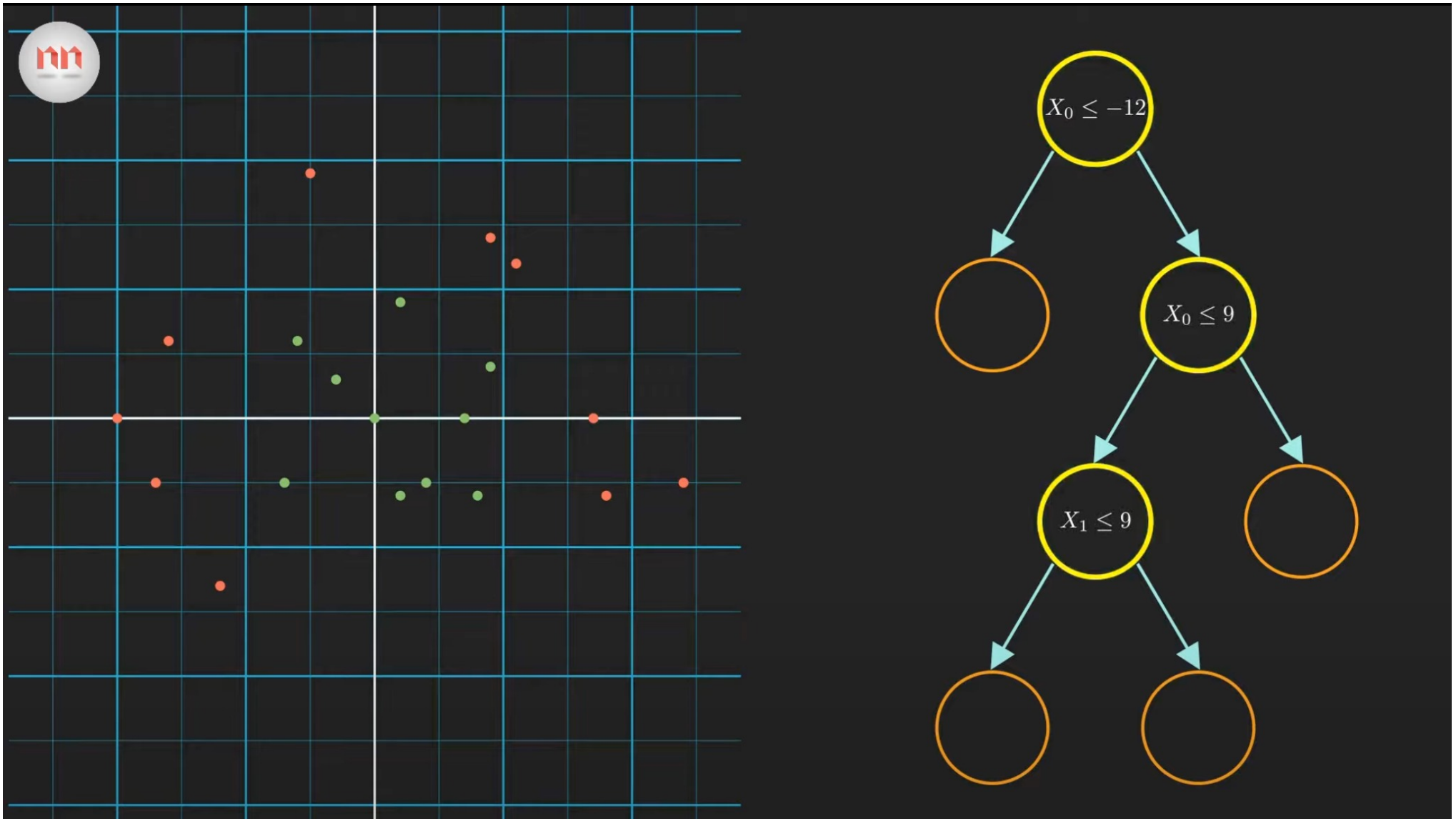
Let's get nerdy!

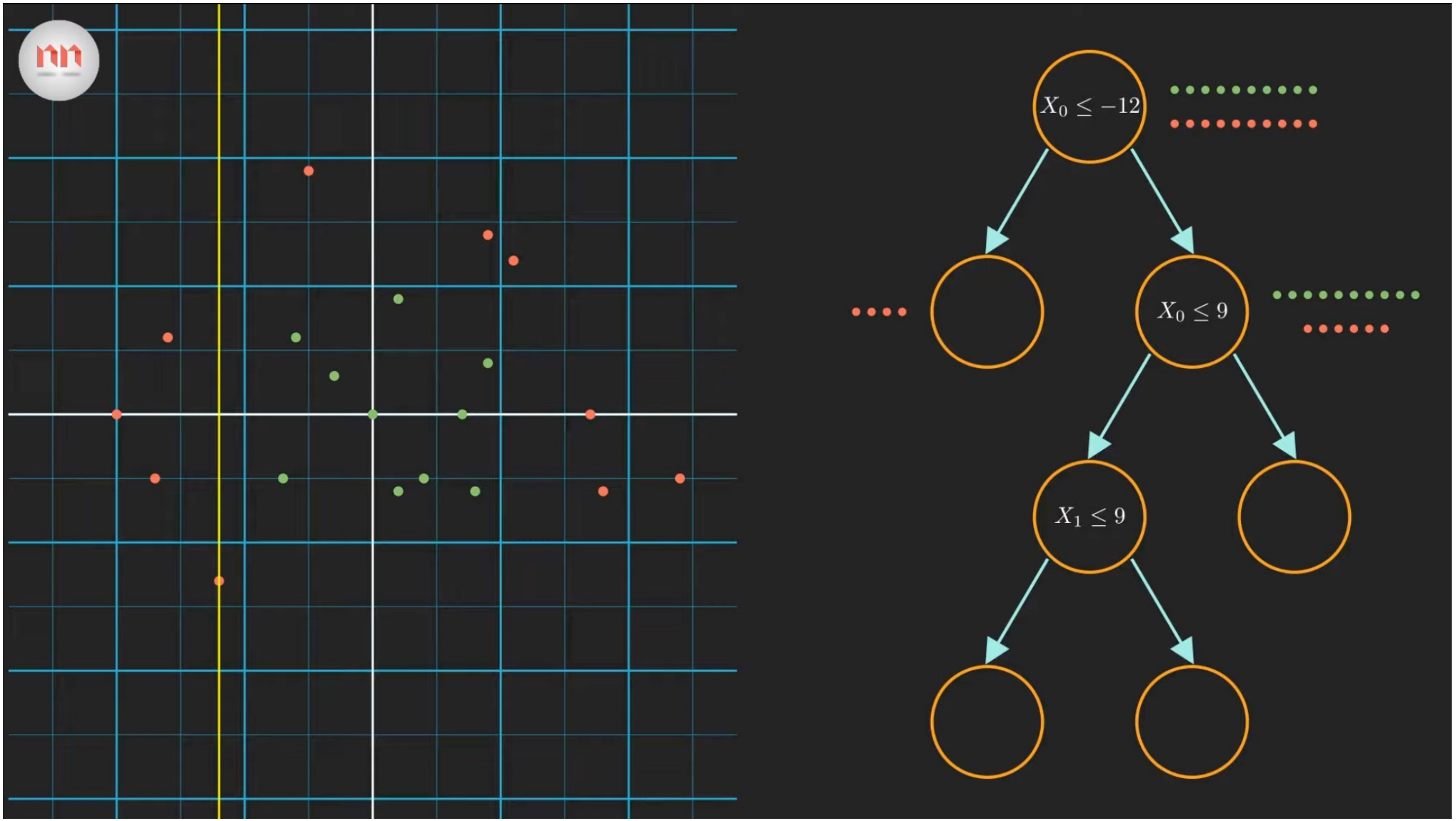

Input
Set of rules of obtain the expected output from the given input.
Output
Algorithm

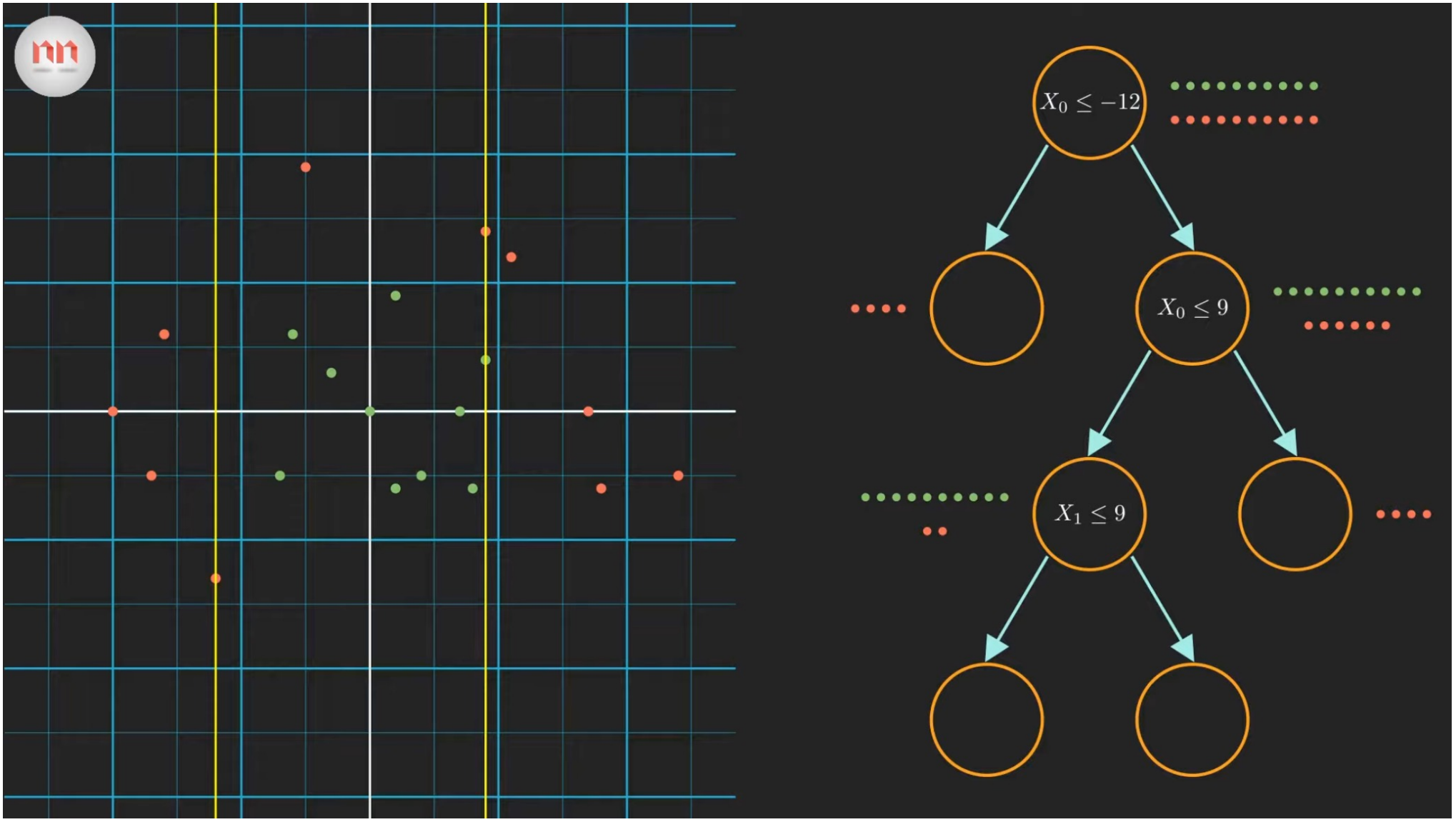


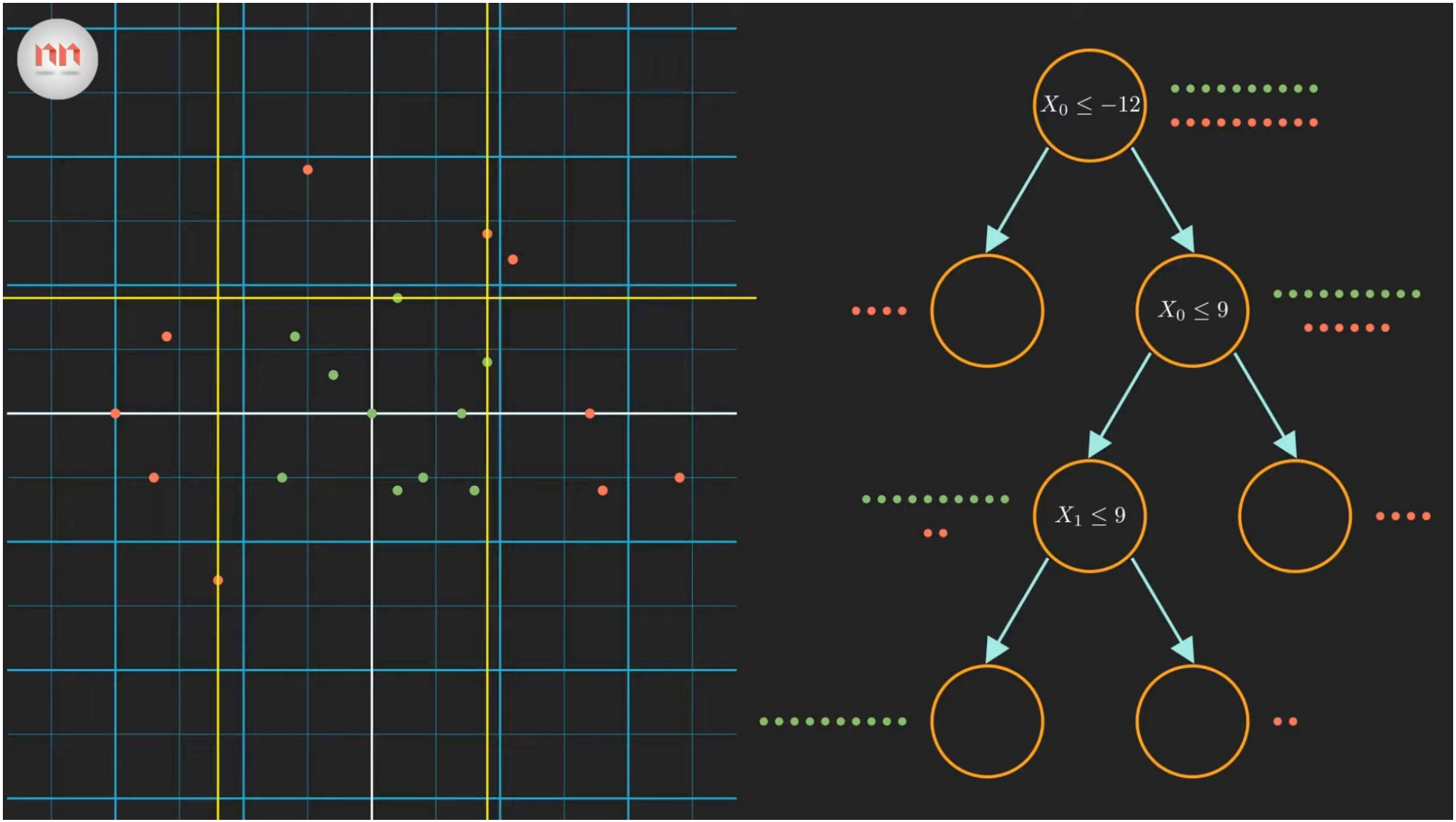
Decision trees

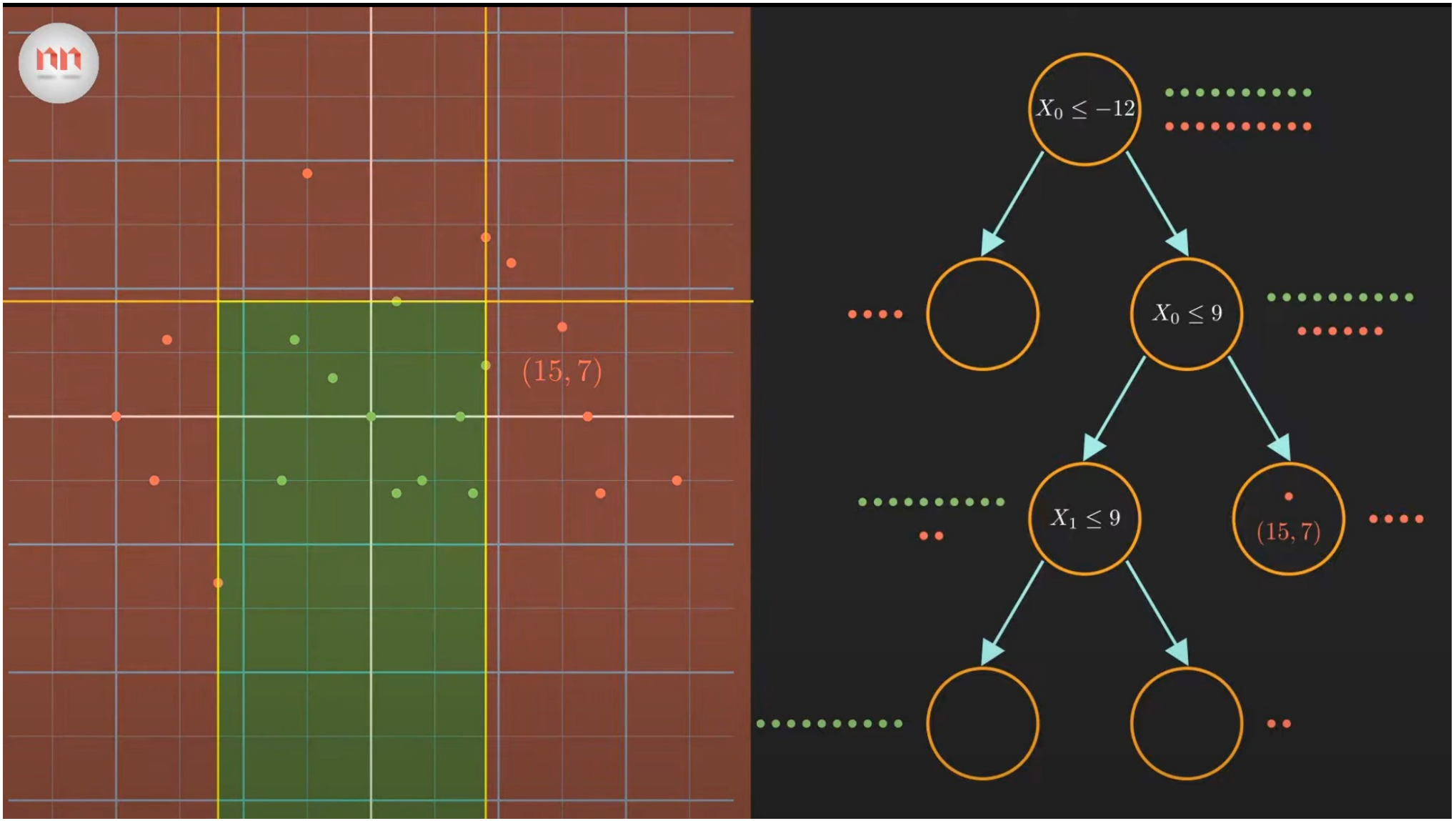
- Among the simplest models
- We want to separate the two classes of dots
- Not possible with a straight line


$X_0 \leq -12$
$X_0 \leq 9$
$X_1 \leq 9$


$X_0 \leq -12$
$X_0 \leq 9$
$X_1 \leq 9$


$X_0 \leq -12$
$X_0 \leq 9$
$X_1 \leq 9$


$X_0 \leq -12$
$X_0 \leq 9$
$X_1 \leq 9$

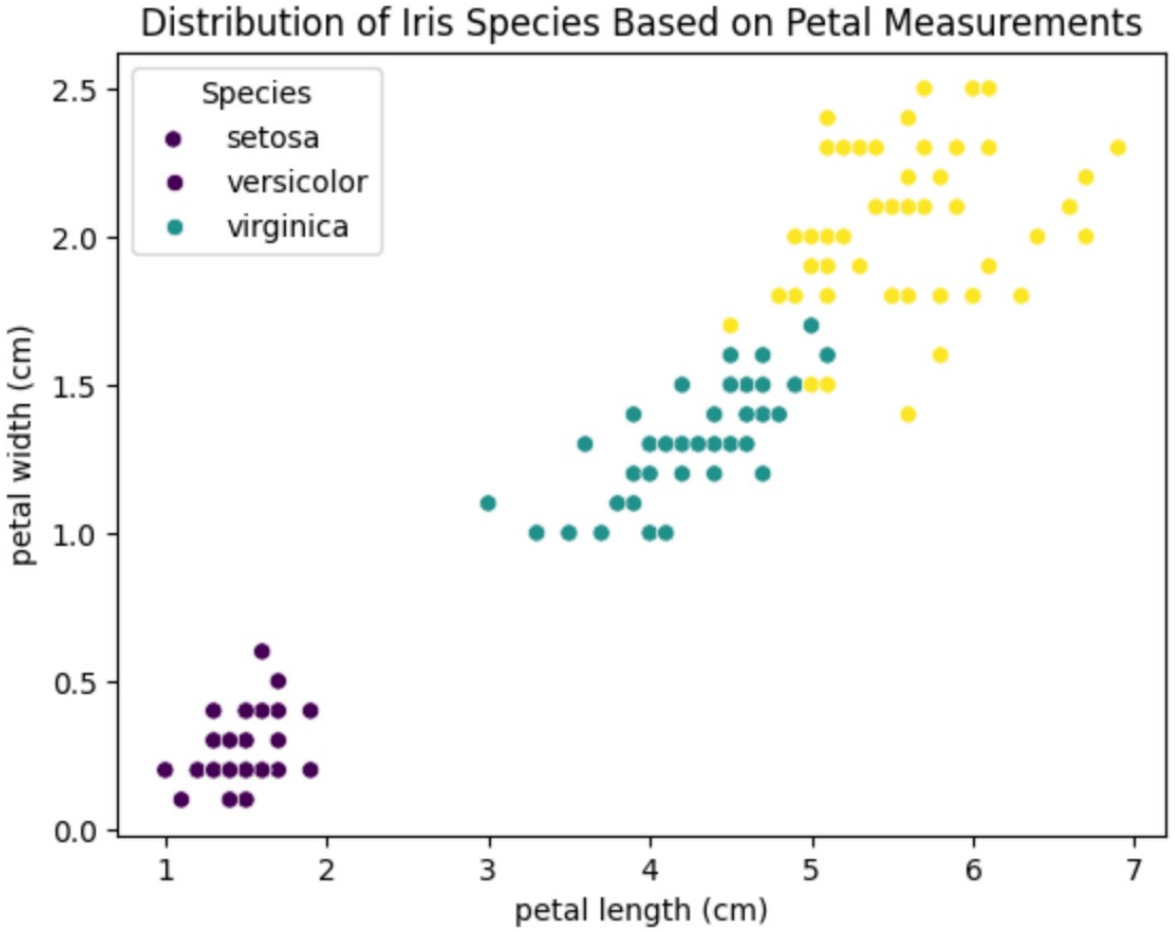

(15, 7)
$X_0 \leq -12$
$X_0 \leq 9$
$X_1 \leq 9$
(15, 7)

Training decision trees

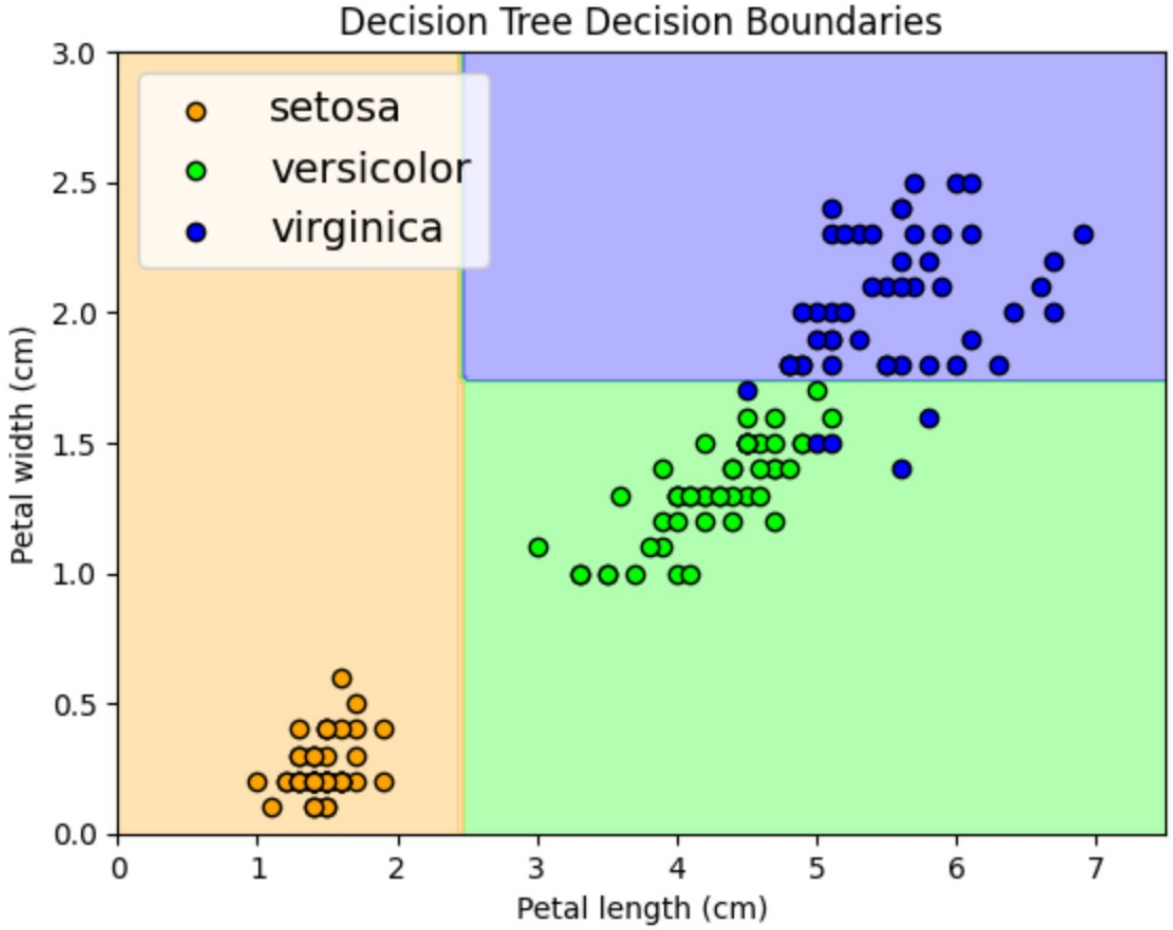
- In each step, the model uses the split that maximizes the information gain – minimizes the *entropy*
- How can I draw the line to *most separate* the two classes?
- Compares all possible splits.

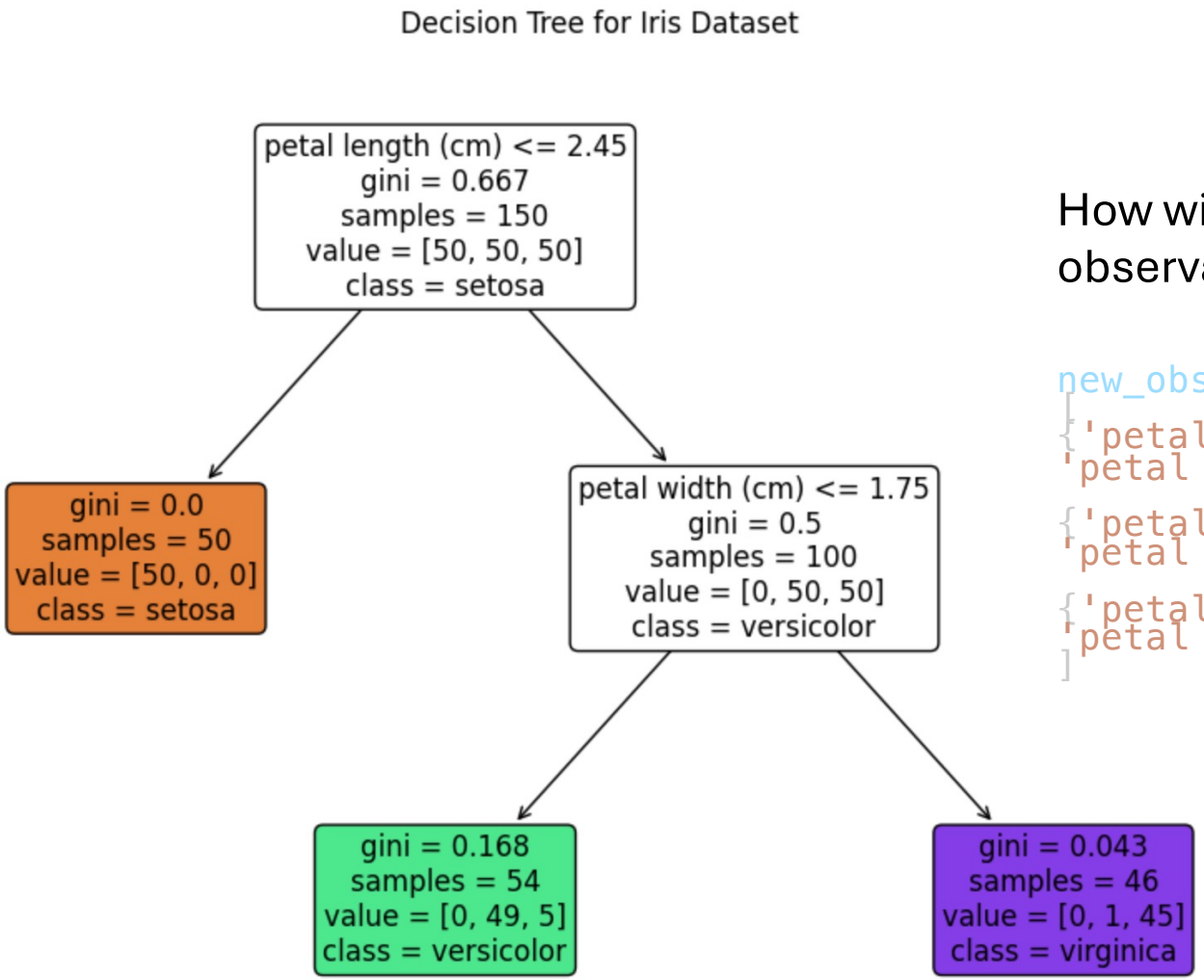
- Downside with Decision Trees: they are highly sensitive to training data. They can fail to generalize.

Example


Distribution of Iris Species Based on Petal Measurements
Species
setosa
versicolor
virginica
petal width (cm)
petal length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

```python
from sklearn.tree import DecisionTreeClassifier, plot_tree
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

# Create and fit the model
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X.drop('species', axis=1), y)  # Ensure to drop the species column for training

def plot_decision_boundary(clf, X_df, axes=[0, 7.5, 0, 3]):
    # Define color map for consistent color usage in contourf and scatter plot
    cmap = ListedColormap(['#FFA500', '#00FF00', '#0000FF'])

    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    X_new_df = pd.DataFrame(X_new, columns=X_df.columns[:-1])  # Use feature names
    y_pred = clf.predict(X_new_df).reshape(x1.shape)
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=cmap)
    for i, class_name in enumerate(iris.target_names):
        plt.scatter(X_df[X_df['species'] == i].iloc[:, 0], X_df[X_df['species'] == i].iloc[:, 1],
                    color=cmap(i), label=class_name, edgecolor='k')
    plt.axis(axes)
    plt.xlabel("Petal length (cm)")
    plt.ylabel("Petal width (cm)")
    plt.legend(loc="upper left", fontsize=14)

# Plotting the decision boundary
plot_decision_boundary(tree_clf, X, [0, 7.5, 0, 3])
plt.title("Decision Tree Decision Boundaries")
plt.show()

# Plot the decision tree
plt.figure(figsize=(12,8))
plot_tree(tree_clf, filled=True, feature_names=iris.feature_names[2:4], class_names=iris.target_names, rounded=True, fontsize=12)
plt.title('Decision Tree for Iris Dataset')
plt.show()
```


Decision Tree Decision Boundaries
setosa
versicolor
virginica
Petal width (cm)
Petal length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7



How will these new observations be classified?

```
new_observations =
[{'petal length (cm)': 5.1,
'petal width (cm)': 1.5},
{'petal length (cm)': 4.5,
'petal width (cm)': 1.3},
{'petal length (cm)': 6.0,
'petal width (cm)': 2.2}
]
```

1. 'versicolor'
2. 'versicolor'
3. 'virginica'

id	x_0	x_1	x_2	x_3	x_4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

id
2
0
2
4
5
5

id
2
1
3
1
4
4

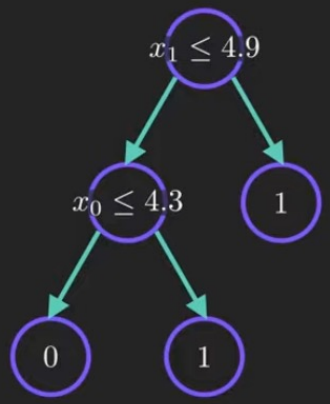
id
4
1
3
0
0
2

id
3
3
2
5
1
2

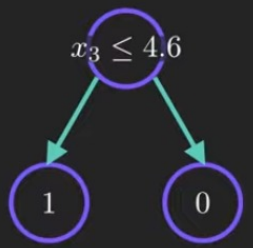
Random Forest

- A forest is several trees.
- Each tree is trained on a random subset of the training data and makes predictions independently.
- The final prediction is obtained by *averaging* or *voting* the predictions.

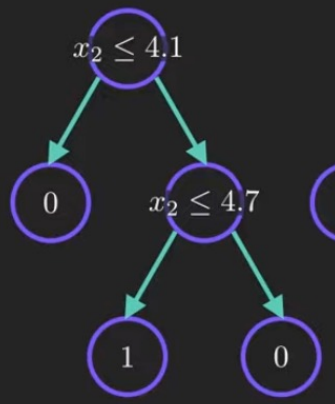
x_0, x_1



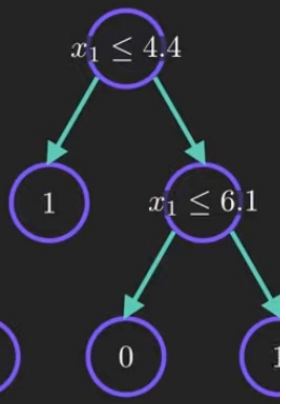
x_2, x_3



x_2, x_4



x_1, x_3

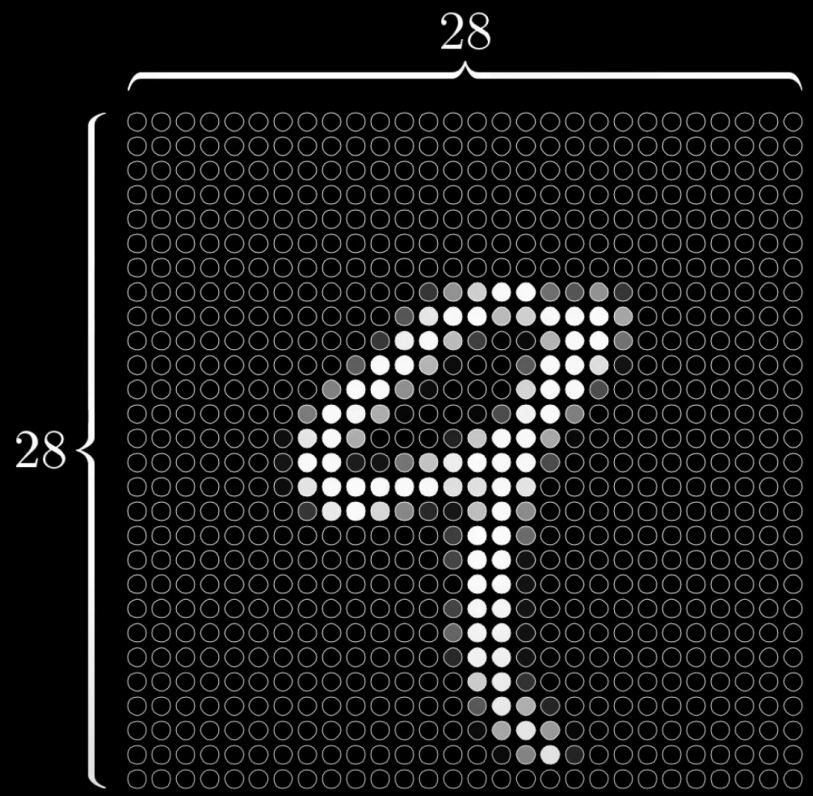


```
# Split the dataset into training and testing sets
# The text is what we use to try to predict the device.
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['device'], test_size=0.3, random_state=42)

# Vectorize the text messages into a bag-of-words model
vectorizer = CountVectorizer(stop_words='english') # Drop stopwords in English
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Use a machine learning classifier, in this case, a Random Forest
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train_vectorized, y_train)

# Predict the device used for the test set
predictions = classifier.predict(X_test_vectorized)
```

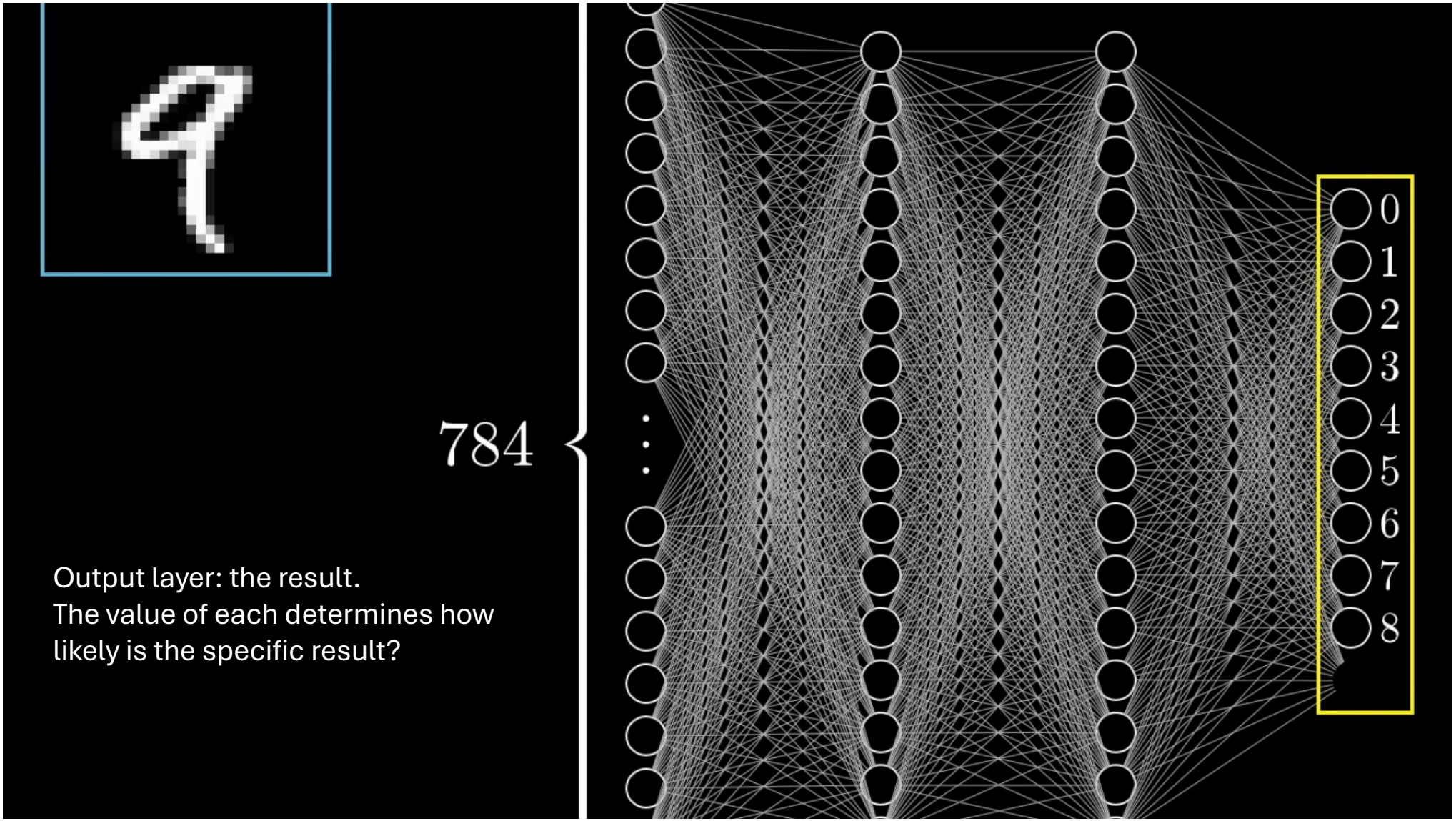
Artificial Neural Networks

Task: identify the handwritten number

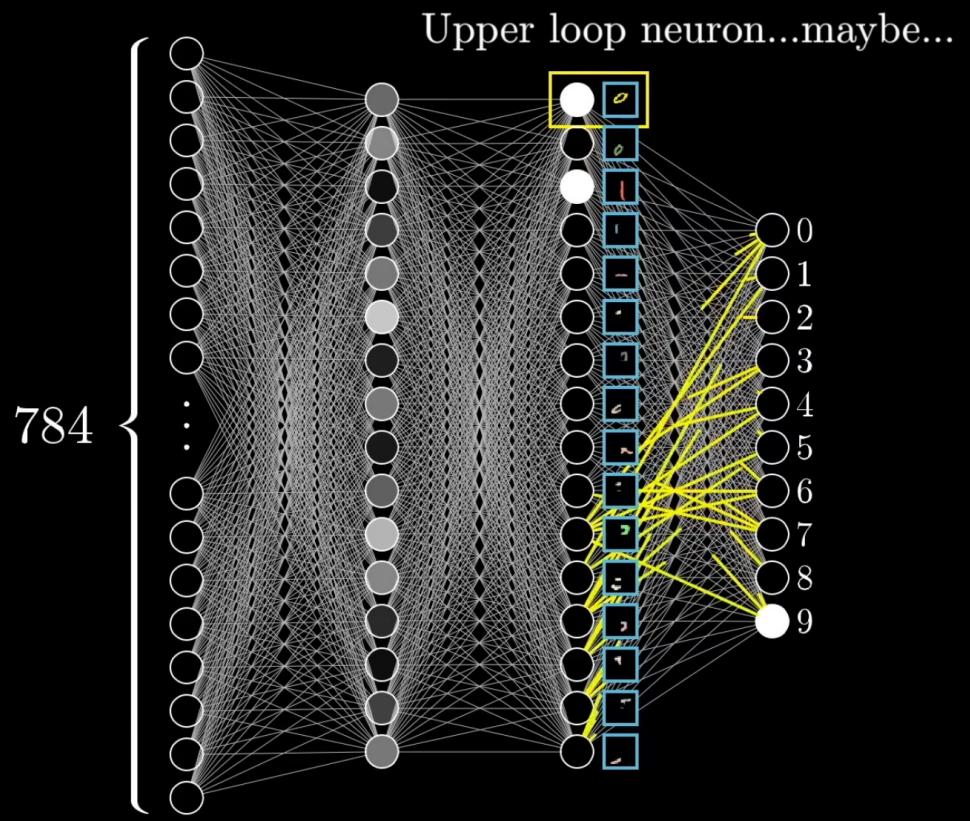
Greyness is activation: from 0 to 1.



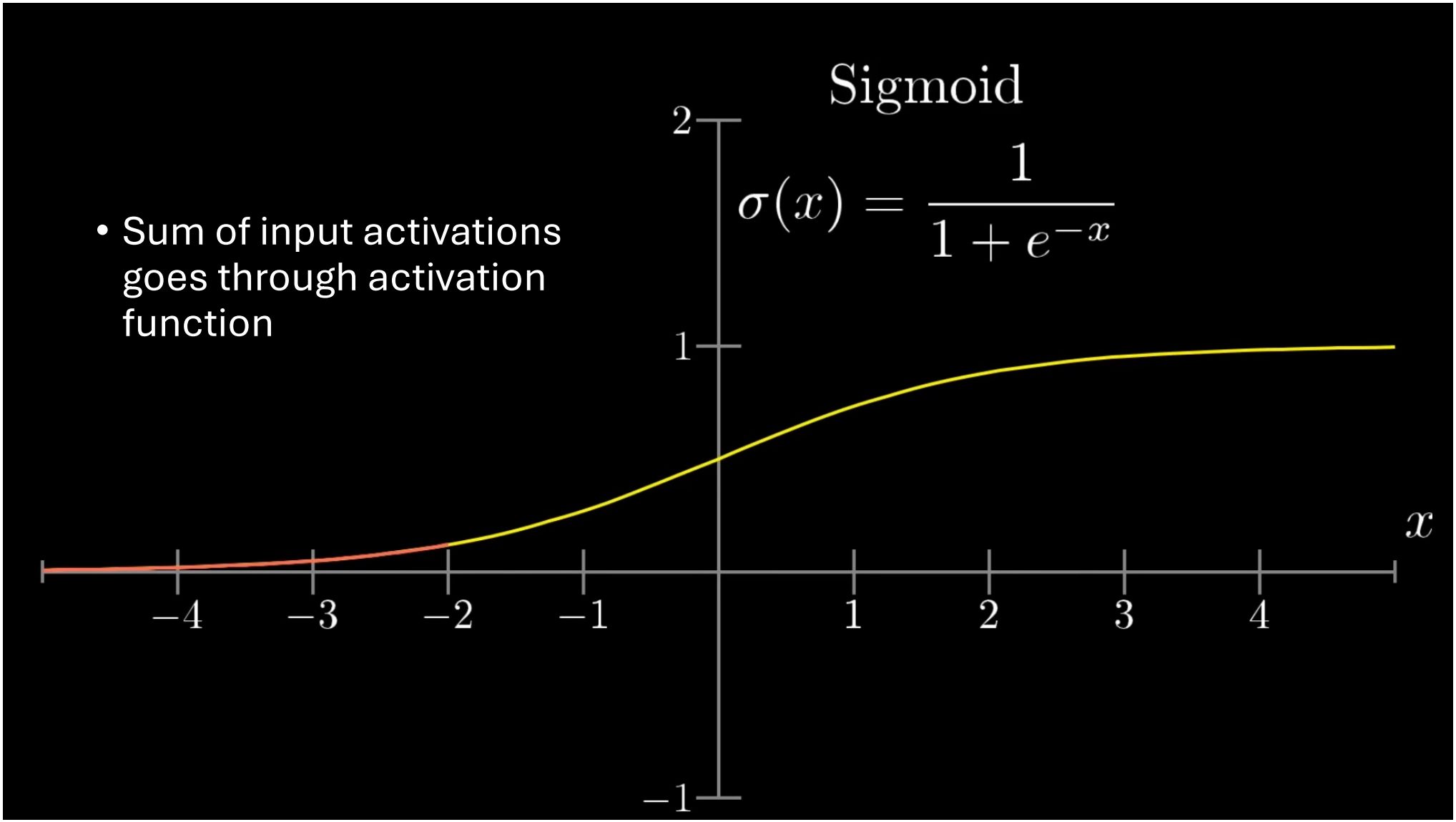
$$28 \times 28 = 784$$


784
0
1
2
3
4
5
6
7
8
Output layer: the result.
The value of each determines how
likely is the specific result?

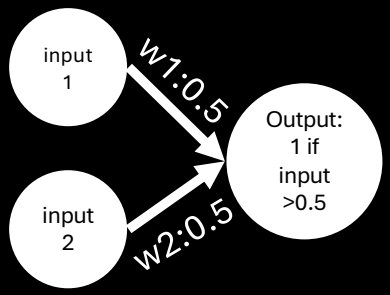
- Activation of each layer depends on the previous layer
- The neurons are connected with *weights*
- Each layer learns ‘features’ - subcomponents

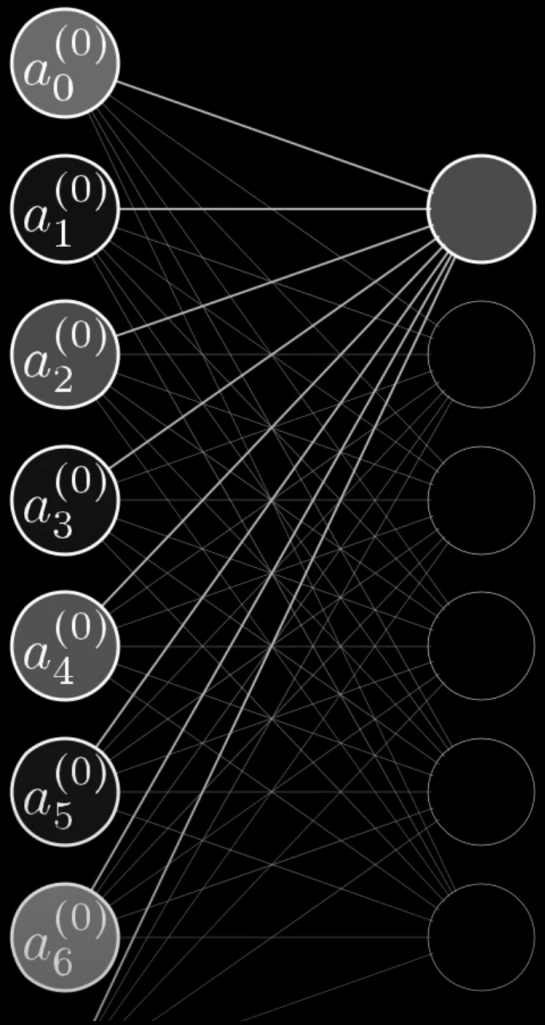


- Sum of input activations goes through activation function



What logical function is implemented by this ANN?





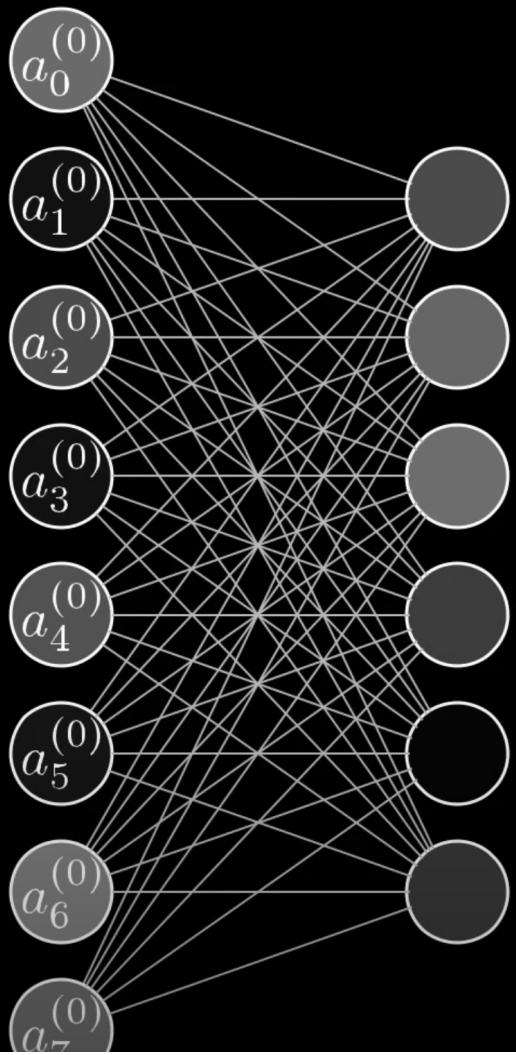
Sigmoid

$$a_0^{(1)} = \sigma\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + b_0 \right)$$

Bias

Network can be understood as a single function with parameters
This small network has *13,002* parameters!

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$



It's just repeated matrix vector products between weights and activation + bias: extremely fast and parallelizable!

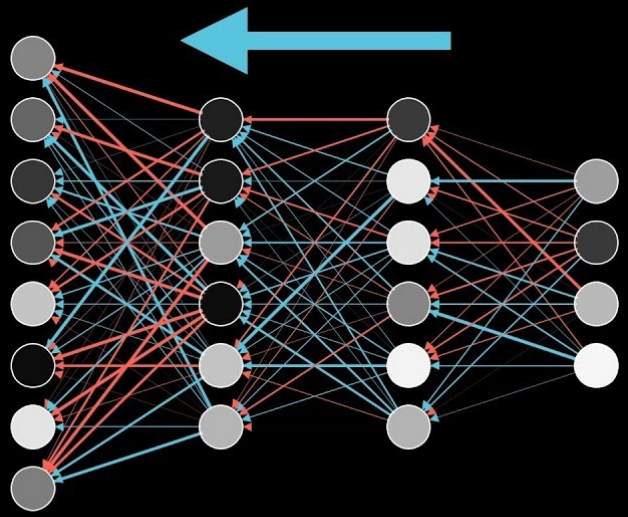
$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$$

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right)$$

Training: Backpropagation

- The error is fed backwards in the network from the output.
- Error = difference between your answer and the correct answer.

- The error moves backwards in the network, adjusting weights by *gradient descent*
- How should I change this weight to most reduce the error?

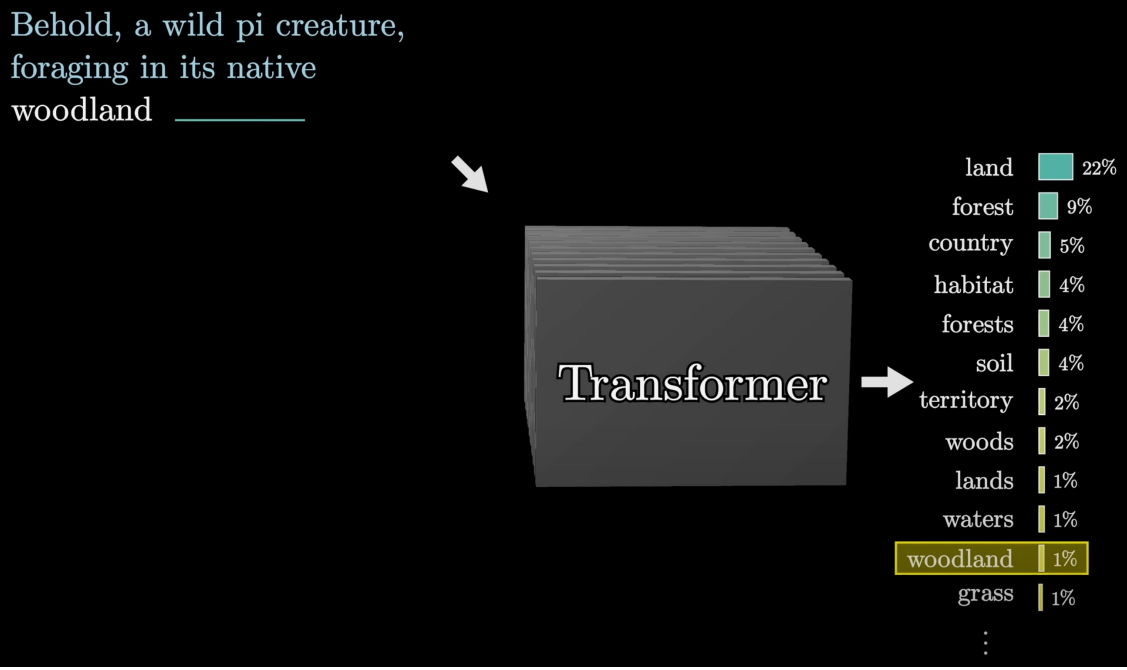


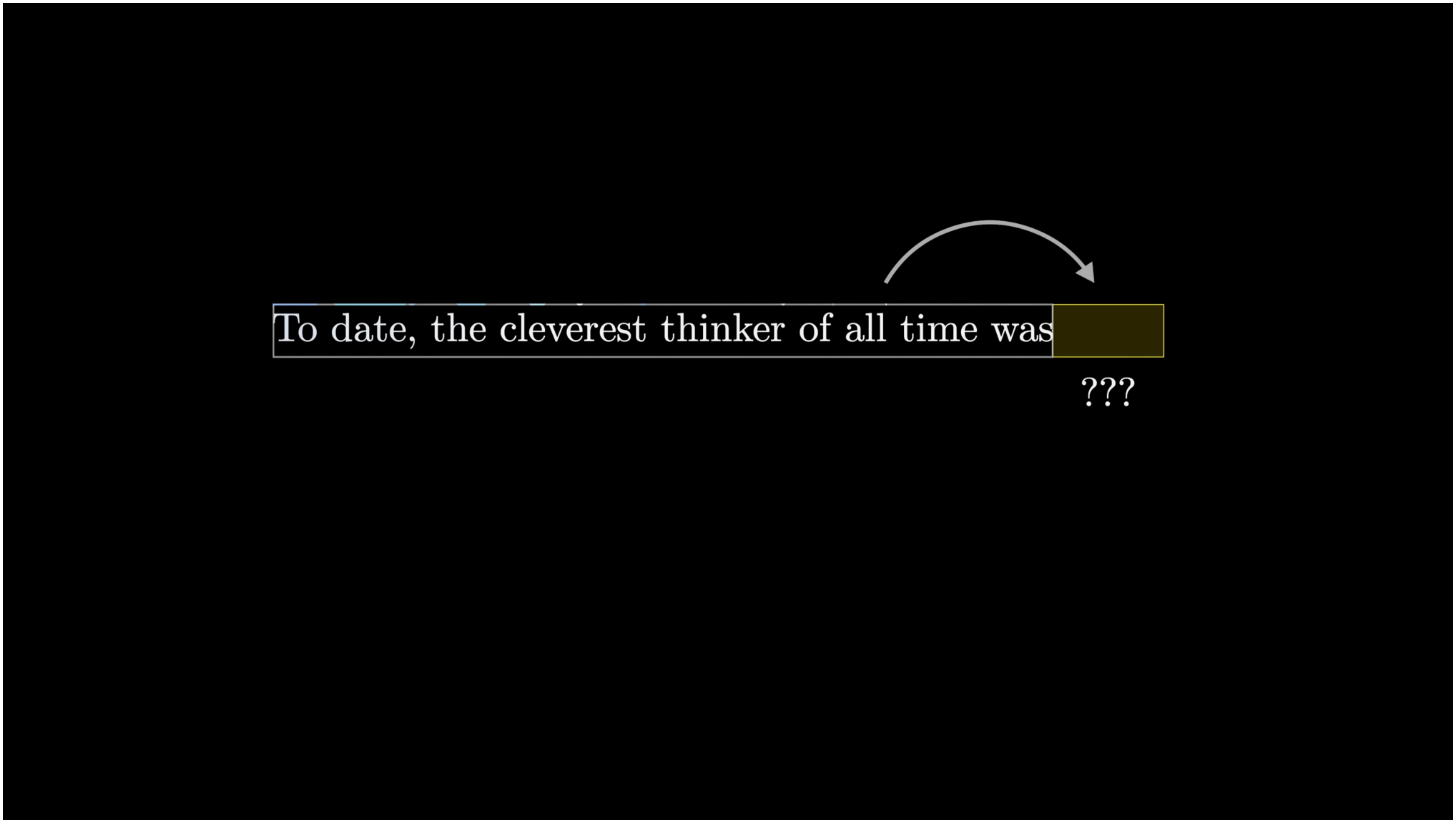
Transformers: fancy neural networks for text

Transformers

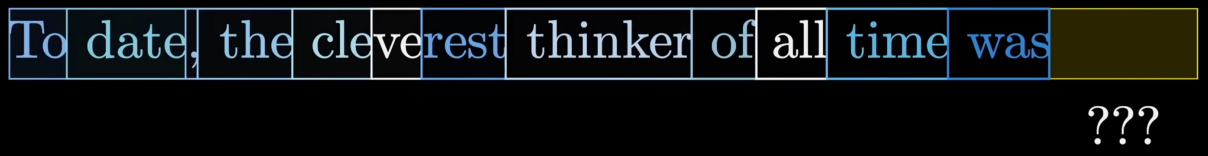
(as in GPT – Generative Pretrained *Transformers*)

- Transformers are a particular neural network structure.
- The task of predicting the next word given a word sequence.
- The easiest way of predicting the next word is to build an understanding of the world and how it works.

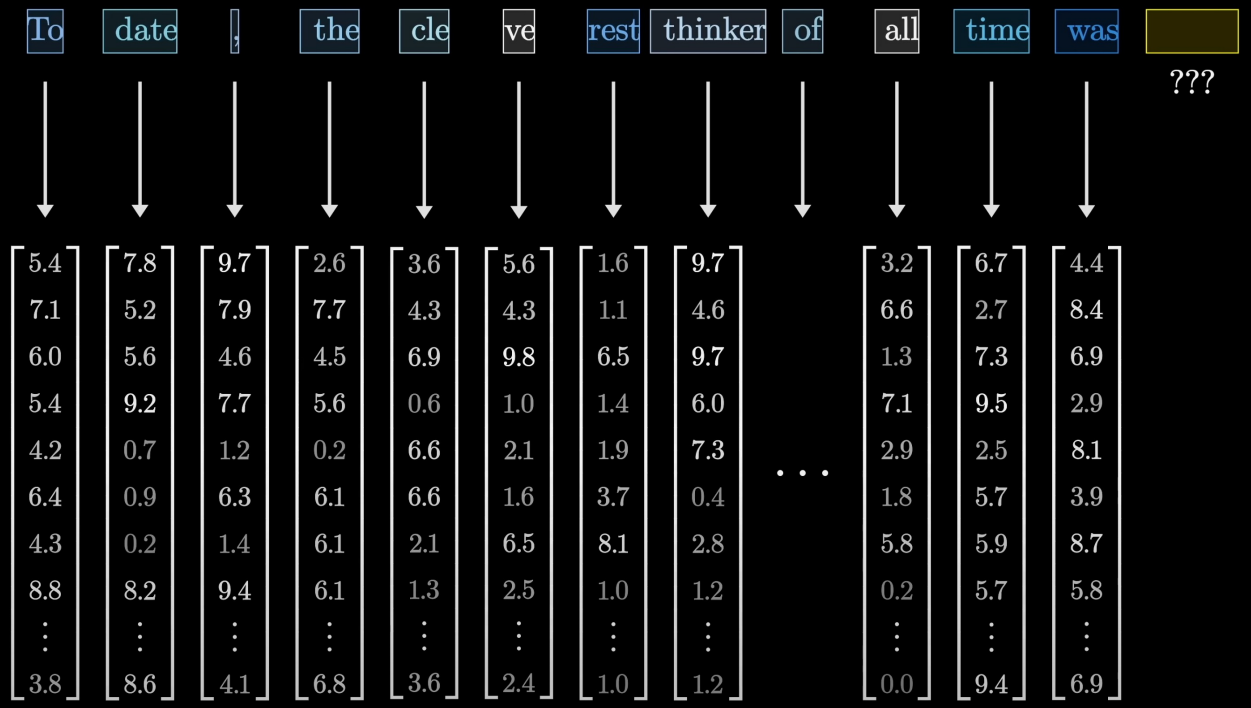



To date, the cleverest thinker of all time was
???

Tokenization

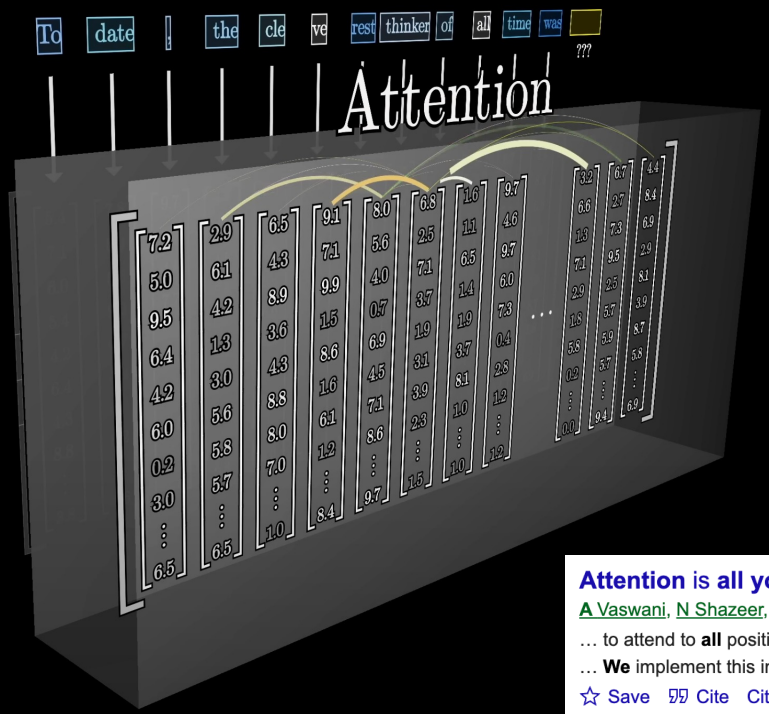


Word embeddings



Attention-block

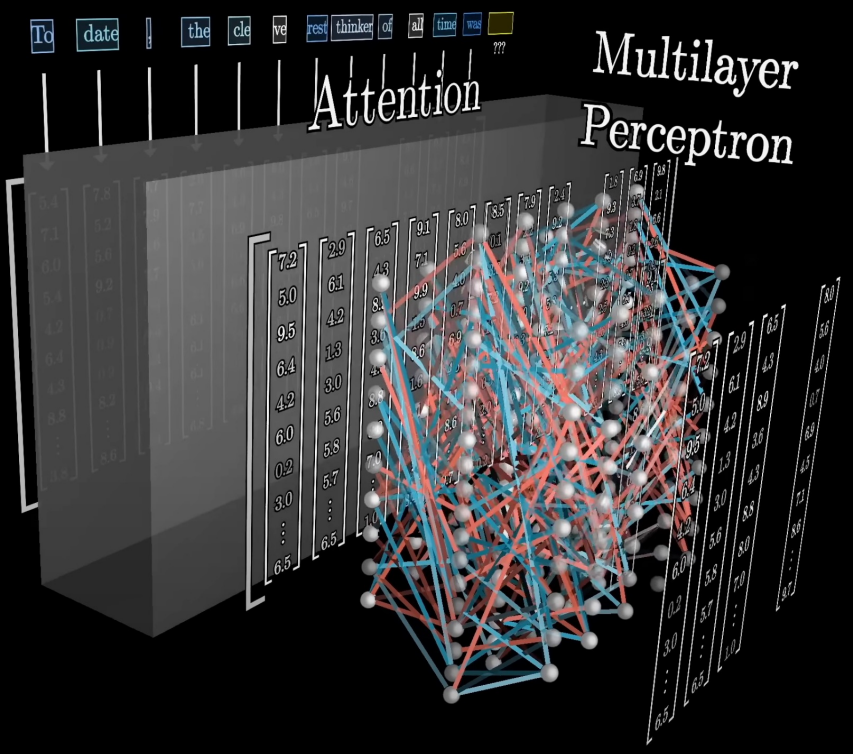
The words talk to each others and give meaning to one another.
Which words change the meaning of each other, and how?



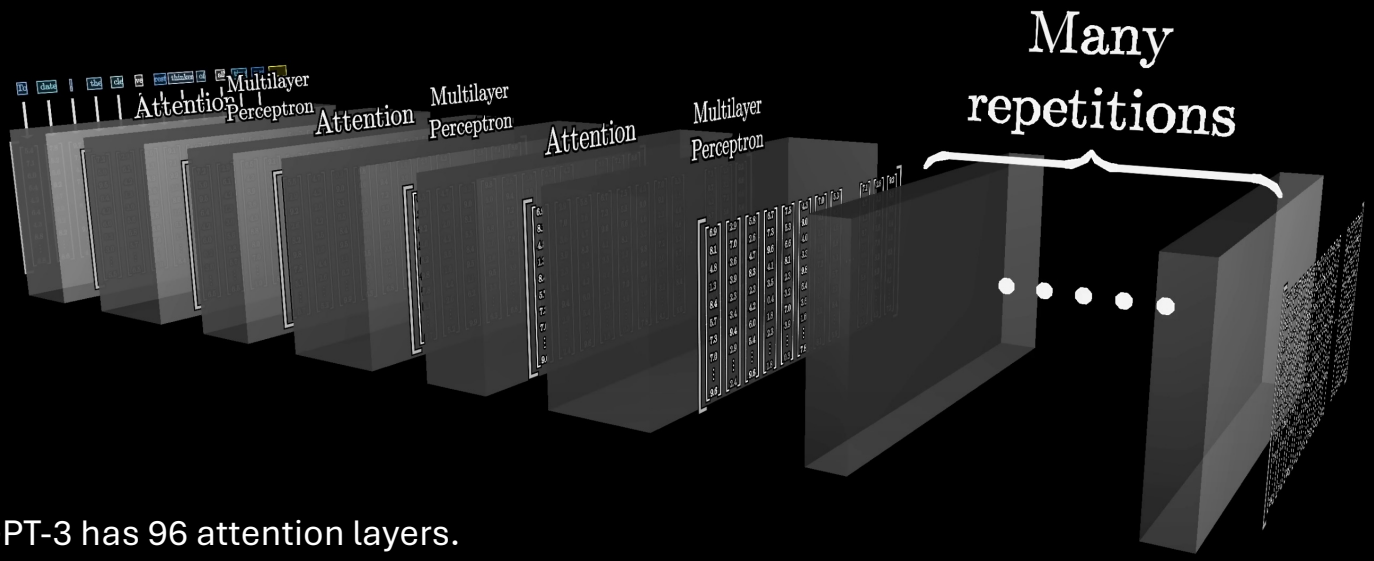
Attention is all you need
A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc
… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to −∞) …
☆ Save 99 Cite Cited by 119393 Related articles All 87 versions »

Multi-layer perceptron / Feed-forward layer

A normal neural network: does the actual processing

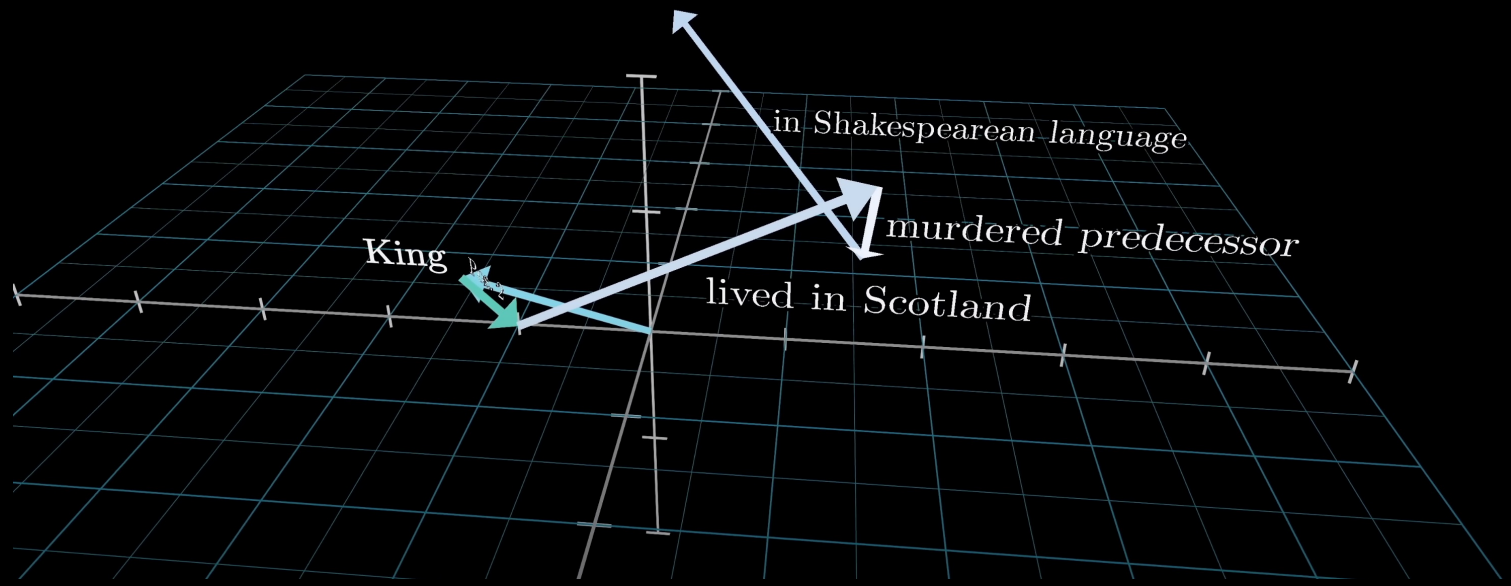


Rinse and repeat: attention x multilayer

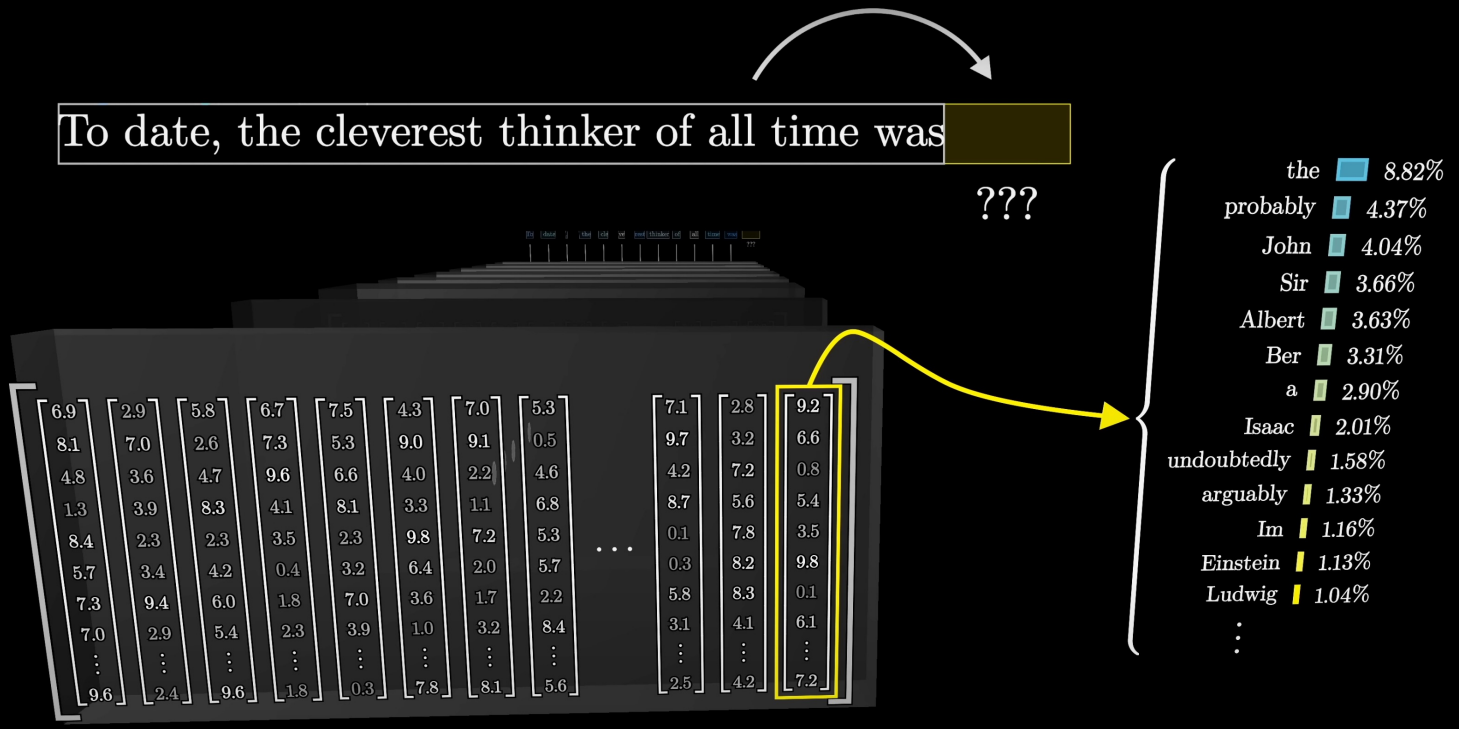


- ChatGPT-3 has 96 attention layers.
- Instead of just a word embedding, it builds a much richer representation of contextual meaning

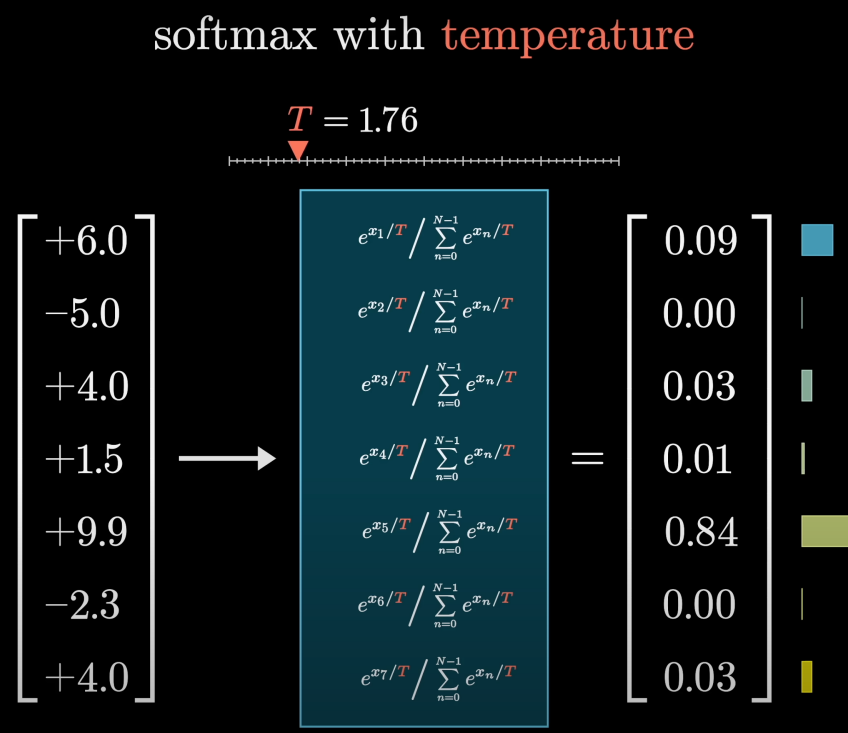
The King doth wake tonight and takes his rouse ...



At the end, the last vector is a probability distribution over words
We sample over this distribution



The sampling is done with softmax



A standard way of turning numbers into a probability distribution

Temperature controls evenness of the distribution

$$e^{x_1/T} \Big/ \sum_{n=0}^{N-1} e^{x_n/T}$$

Has Wall Street Journal become more clickbaity over time?

Let’s use LLM to find out!

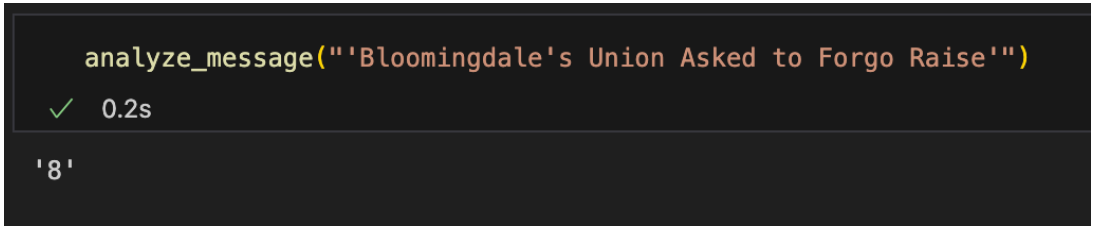
- Calling an LLM from python with ollama

```python
import ollama
response = ollama.chat(model='llama3.2', messages=[
  {
    'role': 'user',
    'content': 'Is the sky blue?',
  },
])
print(response['message']['content'])
```

- We can use LLMs to annotate text!

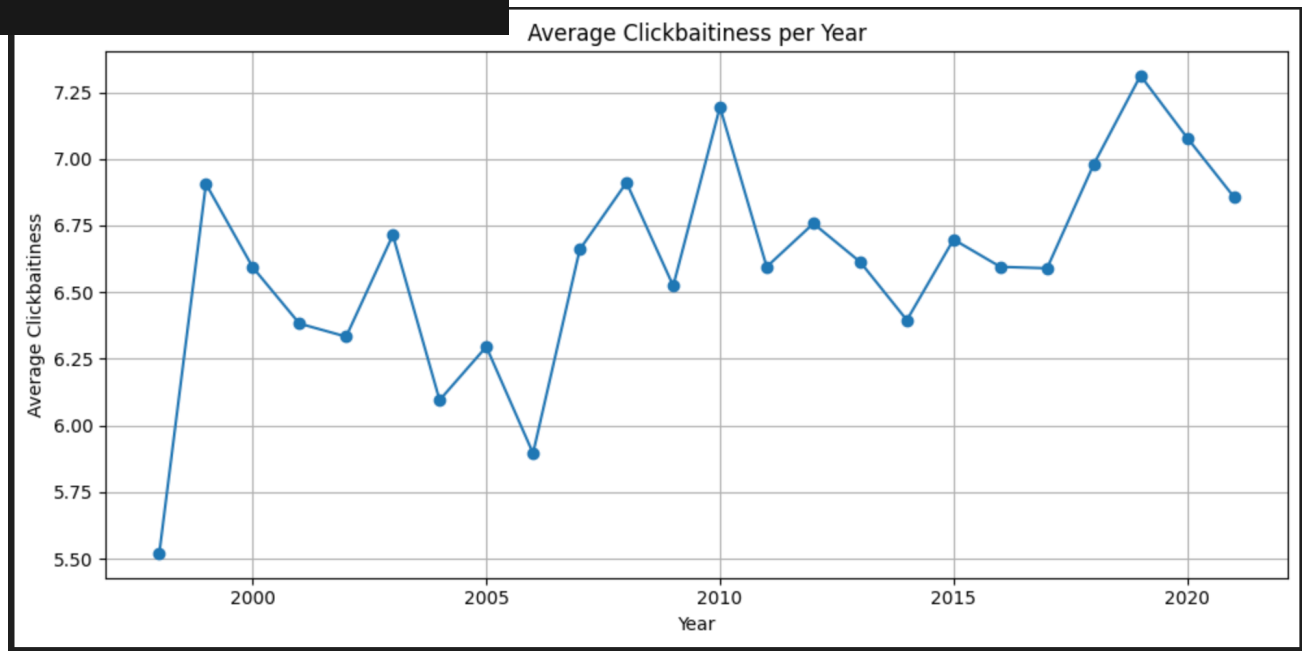
```python
import ollama

# This is our classification prompt
PROMPT = """On a scale from 1 to 10, how 'clickbaity' is this headline?
Rate it based on how much it uses sensational language, emotional appeal, or curiosity gaps to attract clicks —
where 1 means not at all clickbaity (neutral and informative), and 10 means extremely clickbaity (highly sensational or manipulative).
[Respond ONLY with the number. Do not motivate!]"""

def analyze_message(text):
    response = ollama.chat(model='llama3.2', messages=[
        {
            'role': 'user',
            'content': f'{PROMPT}. \n \n"{text}"?',
        },
    ])
    return response['message']['content']
```

- But the quality of the results may vary with the model...



```python
# We store original persistently
sample.to_csv('sample.csv.gz',compression='gzip',index=False)
import time
i = 0
while(True):

    #Find all unprocessed lines; where
    left = sample.loc[sample['clickbaitiness'].isna()]

    #No lines left? Then we're done
    if len(left)==0:
        print("All done!")
        break

    #Take a random line
    line = left.sample()
    index = line.index.values[0]
    text = line['Headline'].values[0]

    print(f"There are {len(left)} left to process. Processing: {index}")

    #Analyze the specific line, chunk by chunk
    result = analyze_message(text)
    sample.loc[index,'clickbaitiness'] = result

    i+=1
    #Save the result to persistent file
    sample.to_csv('sample.csv.gz',compression='gzip',index=False)
```

```python
# Let's plot average clickbaitness by year to see if we have a trend!


# Group by year and calculate the mean clickbaitiness
yearly_avg = sample.groupby('year')['clickbaitiness'].mean().reset_index()

# Plot
plt.figure(figsize=(10, 5))
plt.plot(yearly_avg['year'], yearly_avg['clickbaitiness'], marker='o')
plt.xlabel('Year')
plt.ylabel('Average Clickbaitiness')
plt.title('Average Clickbaitiness per Year')
plt.grid(True)
plt.tight_layout()
plt.show()
```



BERT

(Bidirectional Encoder Representations from Transformers)

- Pre-trained to understand language and context: trained both to guess next word, and to guess a masked word.
- Gives it an understanding of language and context

- We can finetune it to problems we want to solve using additional layers that are trained from scratch

This is the crème of supervised text models!

How clickbaity?

- We finetune a BERT model using some existing labelled data.

```
# Install needed libraries if you haven't already
# !pip install transformers datasets scikit-learn torch

import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from transformers import DataCollatorWithPadding
from torch.utils.data import Dataset
import torch

# 1. Load and preprocess your data
df = pd.read_csv('clickbait_data.csv')  # adjust path
df = df[['headline', 'clickbait']].dropna()

# 2. Train-test split
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['headline'].tolist(), df['clickbait'].tolist(), test_size=0.2, random_state=42)

# 3. Load tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# 4. Tokenize the data
train_encodings = tokenizer(train_texts, truncation=True, padding=True)
val_encodings = tokenizer(val_texts, truncation=True, padding=True)
```

```python
# 5. Create Dataset class
class ClickbaitDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx]) for k, v in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
    def __len__(self):
        return len(self.labels)

train_dataset = ClickbaitDataset(train_encodings, train_labels)
val_dataset = ClickbaitDataset(val_encodings, val_labels)

# 6. Load pre-trained model
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)

# 7. Training configuration
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
)

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# 8. Train the model
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()
```

```python
# Let's try out our new model!

from transformers import pipeline

# Create a pipeline for classification
clickbait_classifier = pipeline("text-classification", model=model, tokenizer=tokenizer, return_all_scores=True)

# Example headline
headline = "10 Things You Won't Believe Happened on Mars"
scores = clickbait_classifier(headline)[0]

# Extract the probability of being clickbait (label 1)
clickbait_prob = [s['score'] for s in scores if s['label'] == 'LABEL_1'][0]
print(f"Clickbait probability: {clickbait_prob:.2f}")
```

✓ 0.0s

```
Device set to use mps:0
Clickbait probability: 1.00
```

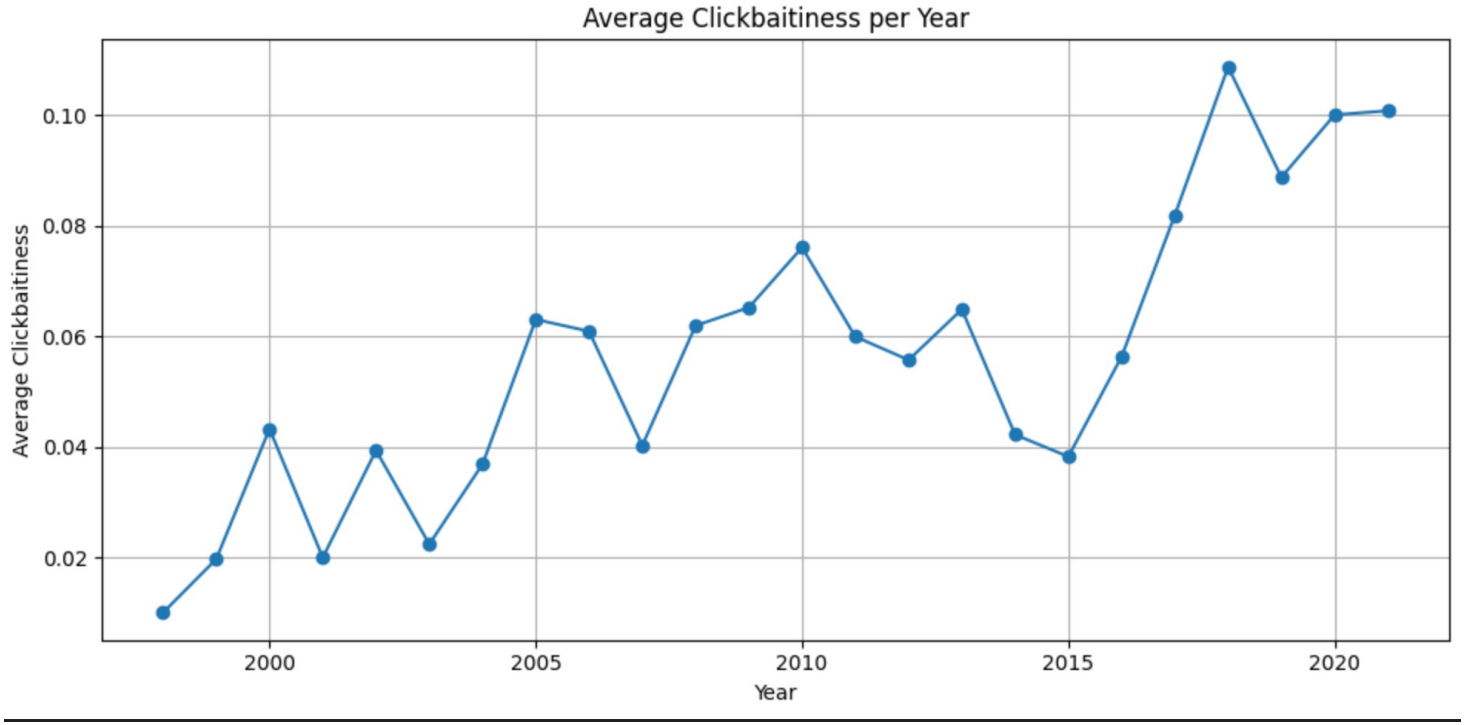
Apply to analyze the WSJ headlines

```
from tqdm import tqdm

batch_size = 32
results = []

for i in tqdm(range(0, len(wsj), batch_size)):
    batch = wsj['Headline'].iloc[i:i+batch_size].tolist()
    outputs = clickbait_classifier(batch)
    for o in outputs:
        clickbait_prob = next(s['score'] for s in o if s['label'] == 'LABEL_1')
        results.append(clickbait_prob)

wsj['clickbaitiness'] = results
```

✓ 5m 42.0s

```
  0%|          | 0/601 [00:00<?, ?it/s]
100%|██████████| 601/601 [05:41<00:00,  1.76it/s]
```

Results!


Average Clickbaitiness per Year
Average Clickbaitiness
0.02
0.04
0.06
0.08
0.10
2000
2005
2010
2015
2020
Year