Machine Learning – Testing and Artificial Intelligence

15-110 – Monday 4/20

Learning Goals

- Identify how **training data, validation data,** and **testing data** is used in machine learning
- Define the following keywords: **artificial intelligence** and **heuristics**
- Recognize how AIs reach **goals** by using a **perception, reason,** and **action** cycle.
- Interpret **game decision trees** to see what an AI would decide to do.

Last Time

In the previous machine learning lecture, we discussed the core idea of what a machine learning algorithm is, how training works, and went over several examples of classic machine learning algorithms

This time, we'll talk more about what we can do with a model once we've generated with it. This includes **testing** the model to see how accurate it is, and how we can use models and algorithms to **make decisions**.

Testing

Building Good Models

Last time, we talked about how we want our machine learning algorithm to discover patterns that exist in a dataset, and incorporate those patterns into the model it produces.

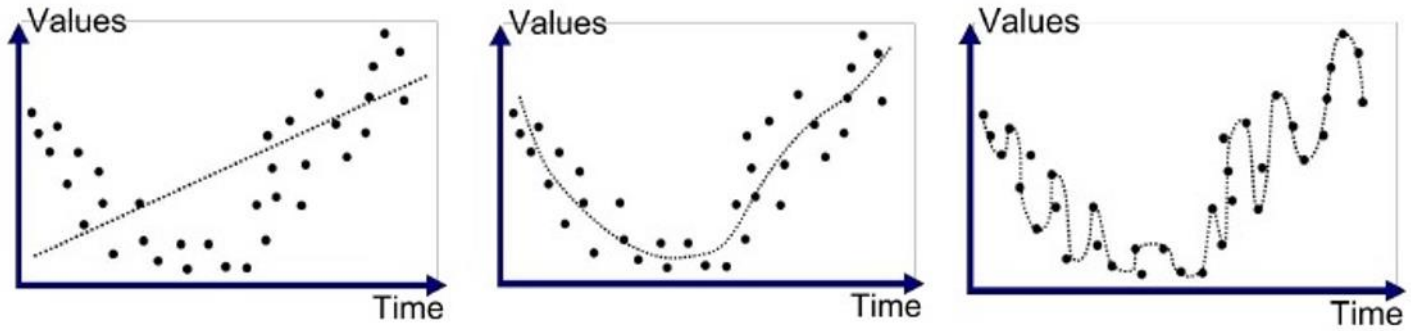
But we don't want the model to **only** work on the data we provided originally. We want it to work on future data too.

When you build a model with a machine learning algorithm, you need to separate your data out into three groups: **training** data, **validation** data, and **testing** data. This will let you evaluate your model on 'new' data once it is done.

Training Data Can Cause Overfitting

The **training data** is normally composed of the majority (maybe 70%) of the available dataset. This data is run through the machine learning algorithm to produce the model. The more training data there is, the more **accurate** the algorithm's model becomes.

This can go wrong if the algorithm over-optimizes the model. For example, it might identify a pattern that only exists within the training data, not in the general population – the pattern might just be noise. This is called **overfitting**. Overfitting can result in a model performing very well on training data, but poorly on test data.



Validation Data Identifies Overfitting

To detect and remove parameters in the model that cause overfitting, you can use **validation data**. This is a subset of the data (maybe 15%) that is **not** used when training the model, but has been labeled with results.

During training, the algorithm will repeatedly evaluate the model it has produced on the validation data, to see how accurate it is. This makes it possible for the algorithm to try out different sets of features, to see which parameters work best.

A common technique used in machine learning is **cross-validation**. One dataset is used for both training and validation data; it is split up in different ways while training, so that the model is not always evaluated on the same data. This avoid overfitting to the validation data.

Testing Data Provides Final Results

Finally, when the algorithm thinks it's achieved an optimal model, the **testing data** is used to determine how accurate that model actually is. This is a portion of the data (maybe 15%) that was set aside at the beginning, which is never used by the algorithm.

The model is run on the test data **once** (unlike validation data, which can be evaluated multiple times). It scores how close the predicted results were to the actual results. That score is the accuracy of the model.

You cannot train on your testing data if you want an accurate result!!!

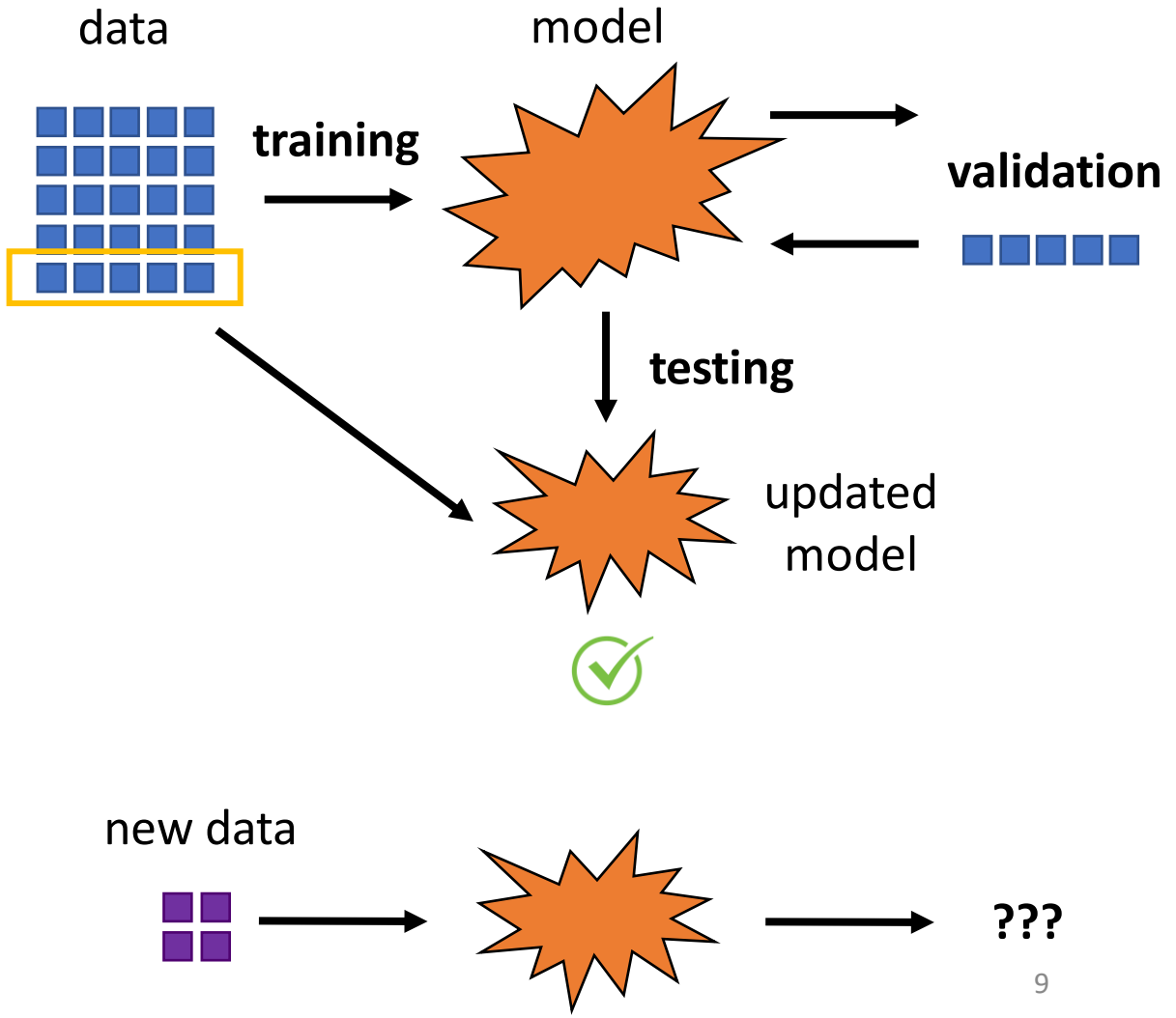
Example: Bad Training Process

What happens if we train on our test data?

The algorithm will get the opportunity to observe patterns in the test data. It will optimize the model to include those patterns.

When the model is tested, it will of course be accurate, because the model was optimized to notice the correct patterns.

But if we try to use the model on new, unlabeled data later on, the patterns may no longer be valid. We don't know for sure, because all of our labeled data was used for testing.



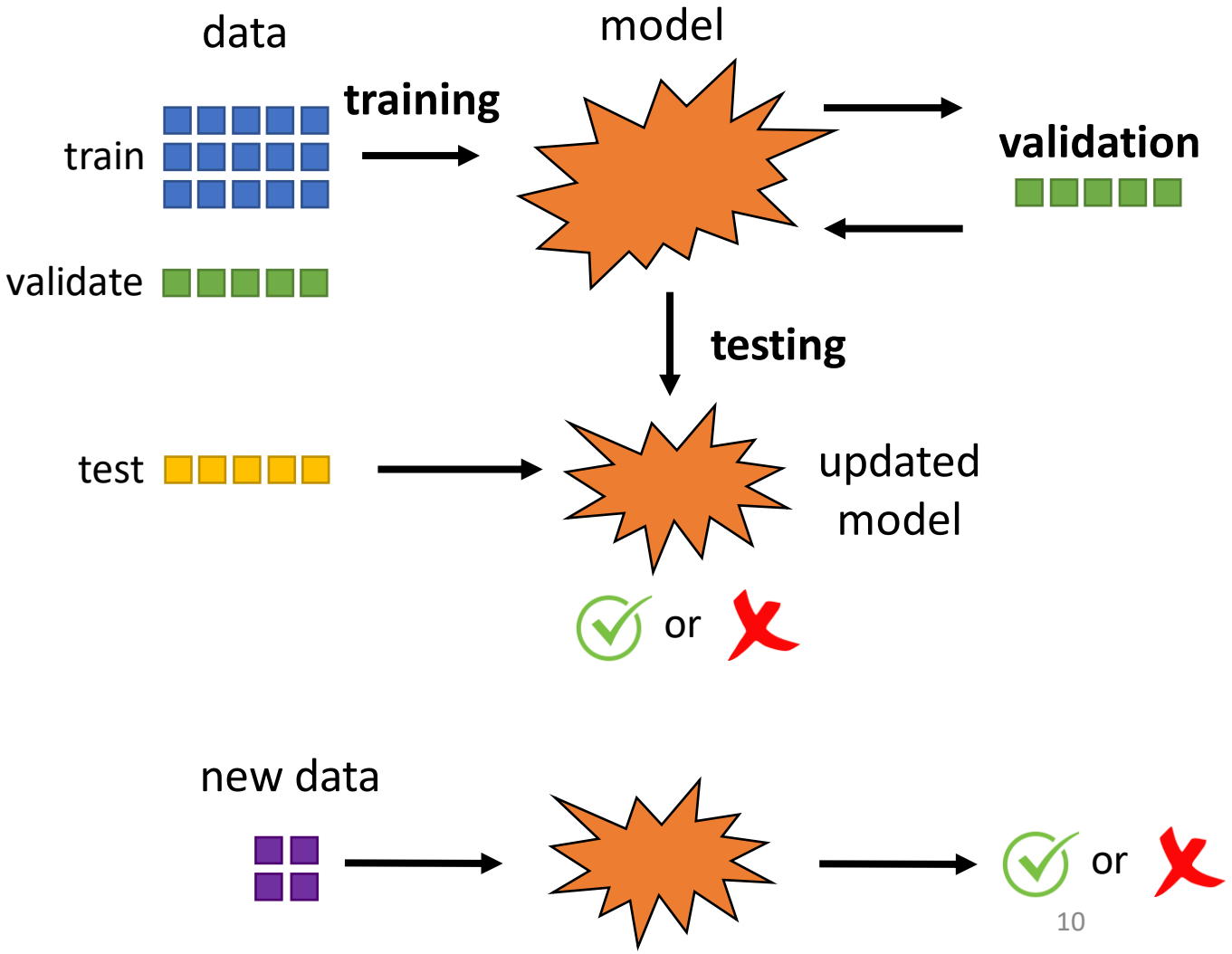
Example: Good Training Process

First, we split the data into training, validation, and testing sets.

We'll train on the training set, and repeatedly test on the validation set. This should remove some of the overfitting from the training data.

When we're done, we'll test on the test set once. That produces our final result. It might be good, or it might be bad; it depends on how the model turned out.

However, the new data should have about the same accuracy, since the model never saw the data before.



ML Models Can Be Used To Make Decisions

Once we've created a model that has a high level of accuracy, what can we do with it?

The model can be used to help computers **make decisions** based on provided data. For example, if we train a model to identify tumors in brain scans by providing lots of labeled brain scans, we can use that model to identify potential tumors in new scans that haven't been looked at by human experts yet.

This process of using machine learning to make decisions is also used to support **artificial intelligence** in computers.

Artificial Intelligence

What is Artificial Intelligence?

The term **artificial intelligence (AI)** is used to describe computers that are 'intelligent' in the same way that humans are.

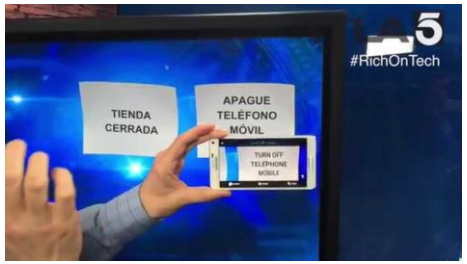
However, it's extremely hard to build a machine with **general intelligence-** that is, a machine that can do everything a human can do. We're still far away from this goal, as it includes many difficult tasks (image perception, language recognition, reasoning, planning, and more).

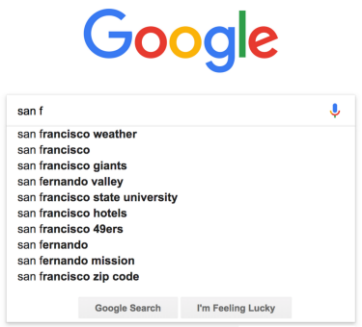
Most modern AIs are **specialized**; they do one specific task, and they do it very well.

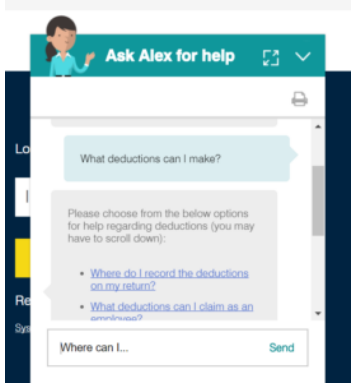
Examples of AIs

We've built AIs that can play games, run robots, and play Jeopardy.

AI is also used to translate text, predict what you'll type, and answer questions on websites.







Goals, Perception, Reason, and Action

Each AI we build has a specific **goal**, the thing it is trying to do. The AI attempts to reach that goal by cycling through three steps: **perceive** information, **reason** about it, then **act** on it.

This is similar to how human brains work! We constantly take in information from our senses, process it, and decide what to do (consciously or unconsciously) based on that 'data'.

AIs can perceive information through user input and through **sensors**, which collect data from the real world. They can then produce output in the computer or use an **actuator** to make a movement in the real world.

Example: IBM Watson

The IBM Watson was designed to play (and win!) the game Jeopardy. Its **goal** was to answer Jeopardy problems with a question. How did it work?

Watson **perceived** the questions by receiving them as text, then breaking them down into keywords using natural language processing.

It used that information to search documents in its database, looking for the most relevant information. With that information, Watson used **reasoning** to determine how confident it was that the answer it found was correct.

If Watson decided to answer, it would **act** by organizing the information into a sentence, then pressing the buzzer with a robotic 'finger'.

Search in Artificial Intelligence

In Watson (and many other artificial intelligence applications), the key to being able to perceive and act quickly lies in **fast search algorithms**.

Being able to search quickly makes it possible for an AI to look through hundreds of thousands of possible actions to find which action will work best. This is what makes it possible for Watson to find a correct answer so quickly, or for a self-driving car to identify when it needs to stop immediately.

Good algorithm design can help speed up search, but AIs also use **heuristics** to search as fast as possible.

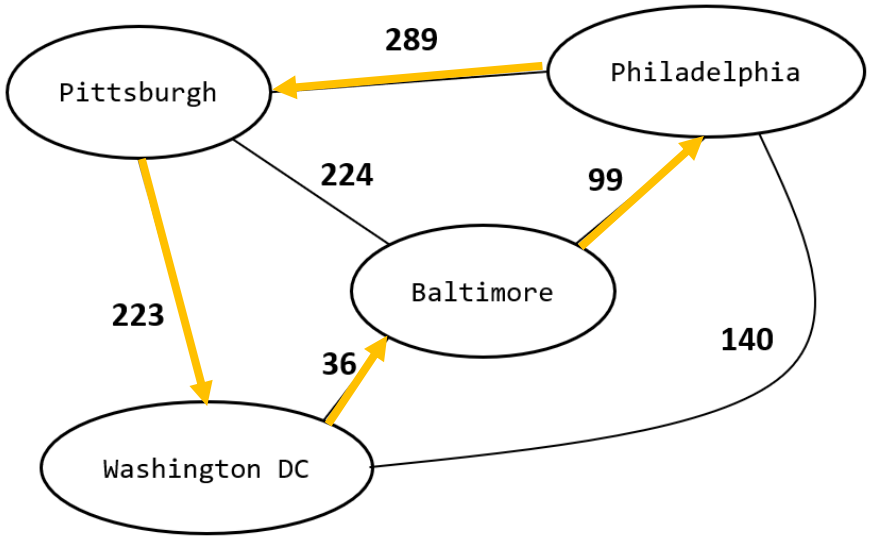
Heuristics Provide Approximate Answers

A **heuristic** is a technique used by an algorithm to find a **good-enough solution** to a problem. Heuristics are typically used because they're **faster** than optimal algorithms.

For example, we can create a search heuristic for a graph search that ranks possible next steps. The AI can then try the highest-ranked next step, instead of looking at all possible options.

Think back to the Travelling Salesperson problem. A heuristic for this problem would be to rank paths based on their length. The algorithm can then always choose the next city to visit by trying the shorter paths first.

Heuristics are fast, but they also have drawbacks. If we use the Travelling Salesperson heuristic, we lose **optimality;** the path we find will be good, but it might not be the best possible path.



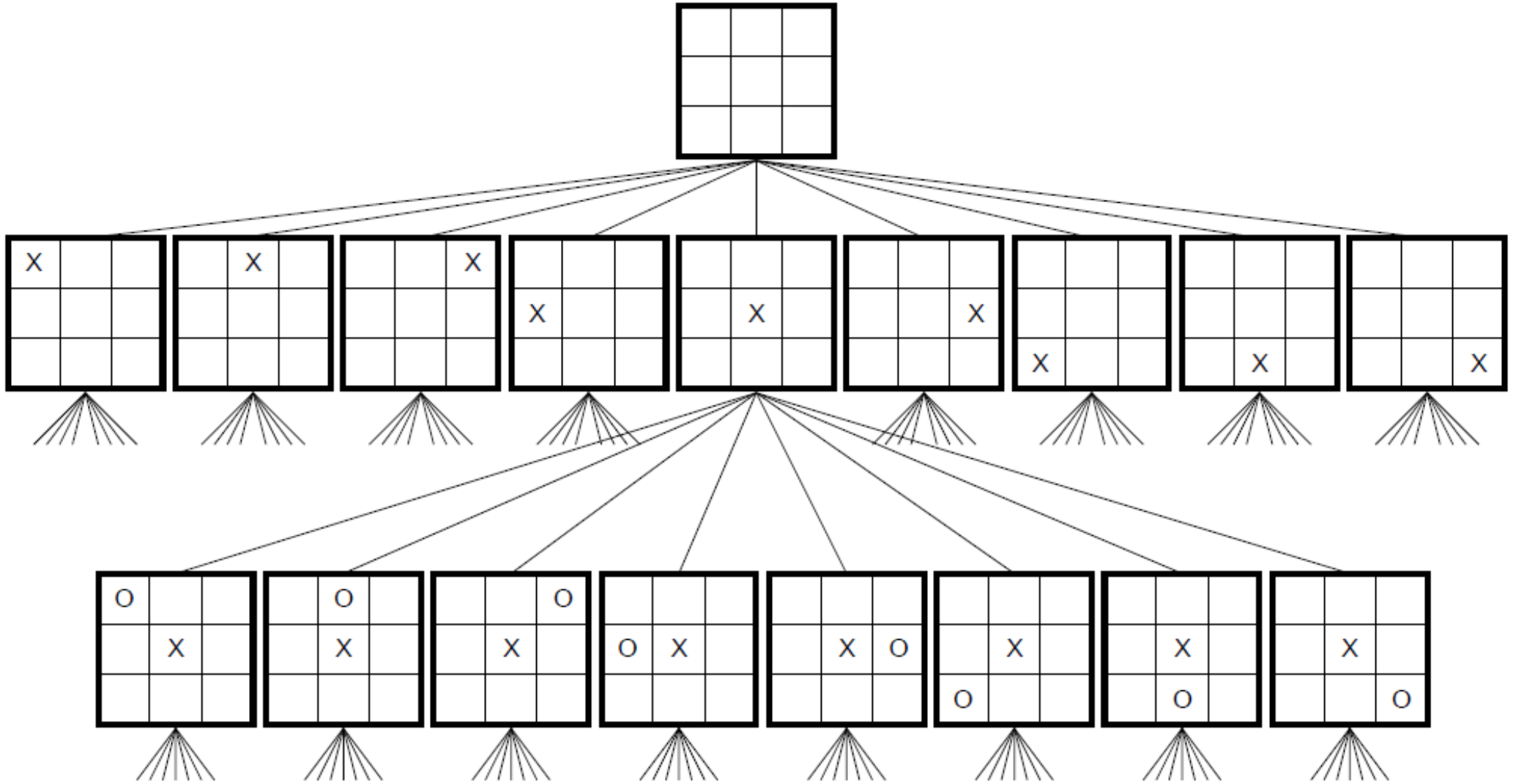
Game Trees

Game Trees Use Heuristics To Choose Moves

AIs also use heuristics in **game trees**, which are used to decide what move a game AI should make out of all possible options.

A game tree is a tree where the nodes are **game states**, and the edges are **actions** made by the AI or the opposing player.

For example, the game tree for Tic-Tac-Toe looks like this...



Full board here: https://xkcd.com/832/

Reading a Game Tree

The **root** of a game tree is the current state of the game. That can be the start state (as in the previous example), or it can be a game state after some moves have been made.

The **leaves** of the tree are the final states of the game, when the AI wins, loses, or ties.

The edges between the root and the first set of children are the possible moves the AI can make. Then the next set of edges (from the first level to the second) is the moves the opponent can make. These alternate all the way down the tree.

Game Trees are Big

How many possible outcomes are there in a game of Tic-Tac-Toe?

Let's assume that all nine positions are filled. That means the **depth** of the tree is 9 (there are nine moves, so the root + 9 levels). There are 9 options for the first move, 8 for the second, 7 for the third, etc... that's **9!**, which is 362,880.

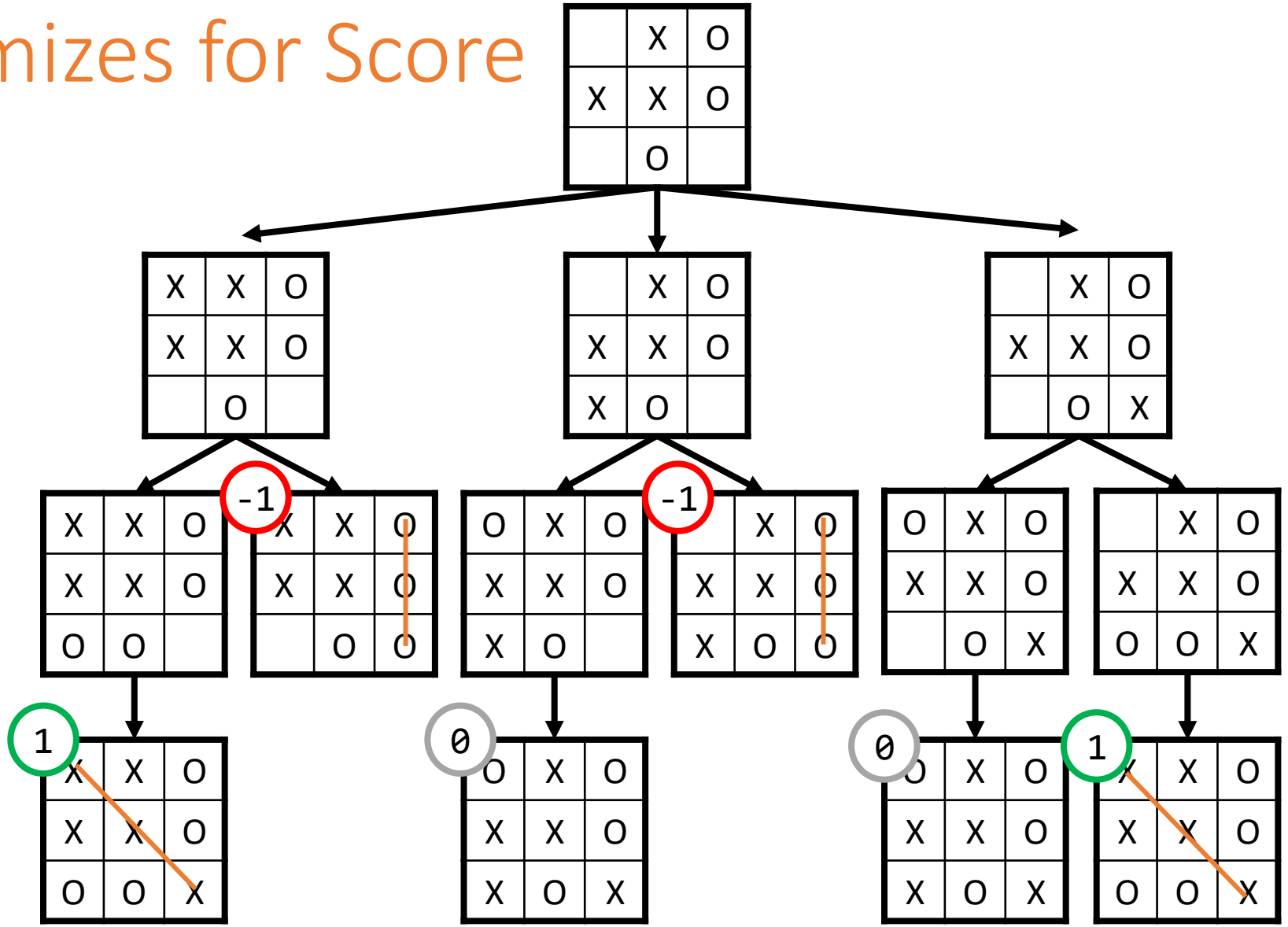
This number is a bit larger than the real set of possibilities (some games end early), but it's a good approximation.

How can the AI choose the best move to make out of all of these options?

Minimax Optimizes for Score

The **minimax** algorithm can be used to maximize the final 'score' of a game for an AI.

In Tic-Tac-Toe, we'll say that the score is 1 if the computer wins, 0 if there's a tie, and -1 if the human wins.



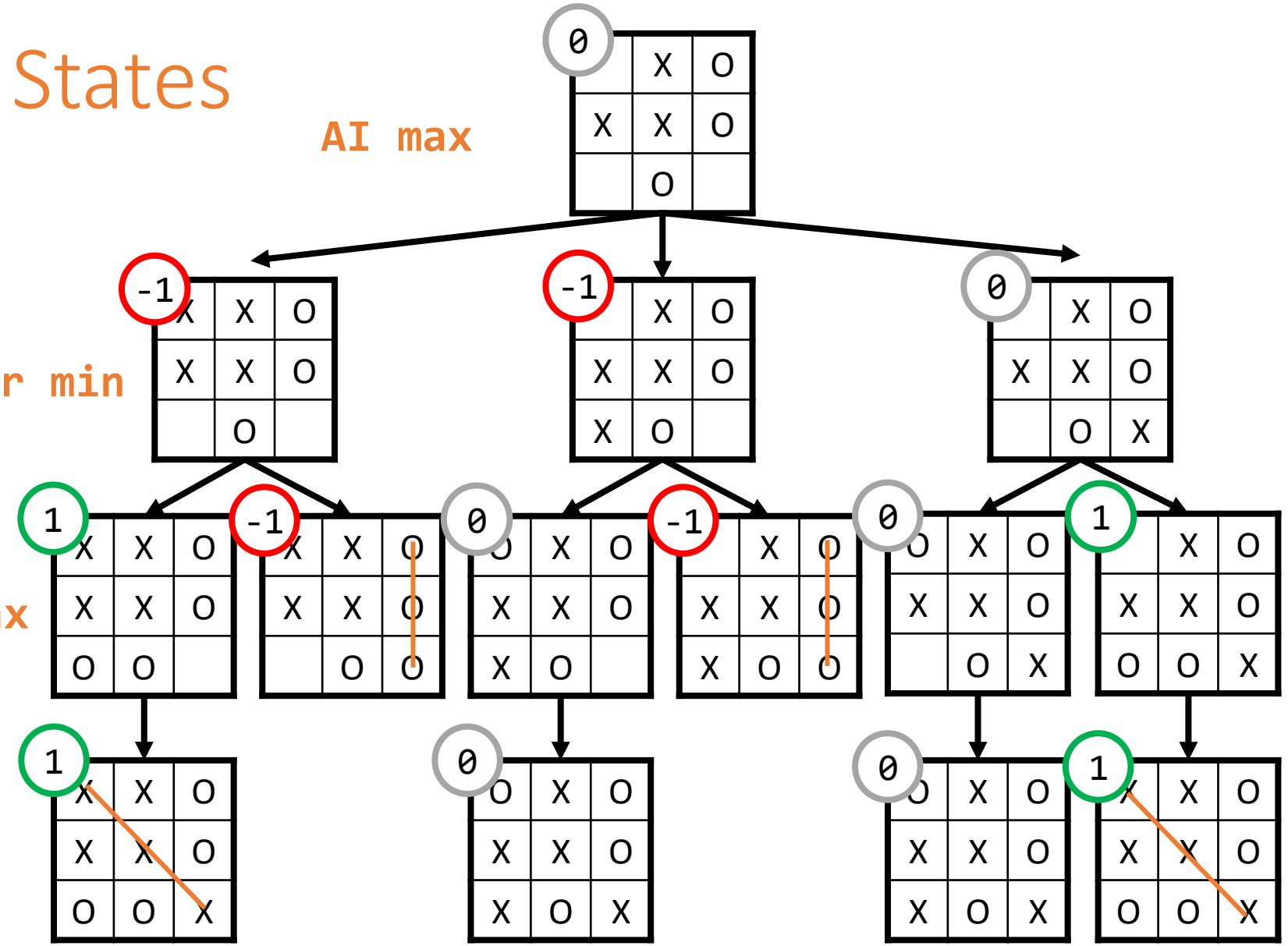
Scoring Game States

How do we score the intermediate states? Look at the scores of the state's children.

If the next move is made by the AI, take the **maximum** of the scores.

If it's made by the opponent, take the **minimum**.

Start from the leaves, and build up to the root.



Minimax Algorithm

```
# Need to use a general tree- "children" instead of "left" and "right"
def minimax(tree, isMyTurn):
    if len(tree["children"]) == 0:
        return score(tree["value"]) # base case: score of the leaf
    else:
        results = [] # recursive case: get scores of all children
        for child in tree["children"]:
            # switch whose turn it will be for the children
            results.append(minimax(child, not isMyTurn))
        if isMyTurn == True:
            return max(results) # my turn? maximize!
        else:
            return min(results) # opponent's turn? minimize!

def score(state):
    ??? # this depends on your goal
```

Heuristics in Minimax

Tic-Tac-Toe is a simple game, so its game tree is small (for a computer).

More complex games have much larger trees. For example, in Chess, there's an average of 35 possible next moves per turn, with an average of 100 turns per game. That means there are 35^{100} possible states to check – way too many!!

Instead, the computer can move down a set number of levels in the game tree, then use a **heuristic** to score all the states at that level. Then it can use minimax to find the next-best move based on the heuristic scores.

Designing Heuristics

How could we design a good heuristic for Chess?

Use information about the state of the board. How many pieces does the computer have left? What about the user?

How **valuable** are each of those pieces? The queen should get a higher rating than a pawn.

And how **safe** is the king? Can we write an algorithm to measure how protected the king is on the board?

Game AIs

Algorithms like minimax and the use of heuristics have made it possible for AIs to beat world champions at games like Chess, Go, and Poker.

Why did it take 9 years to get from Chess to Go? Go has many more next moves than Chess, so it needed more advanced algorithms (including Monte Carlo randomization and machine learning!).

These AIs will keep improving as computers grow more powerful and we design better algorithms.

DeepBlue beat chess grandmaster Garry Kasparov in 1997

AlphaGo beat 9-dan ranked Go champion Lee Sedol in 2016

Learning Goals

- Identify how **training data, validation data,** and **testing data** is used in machine learning
- Define the following keywords: **artificial intelligence** and **heuristics**
- Recognize how AIs reach **goals** by using a **perception, reason,** and **action** cycle.
- Interpret **game decision trees** to see what an AI would decide to do.