AI and Mathematics

Ivan Zelich

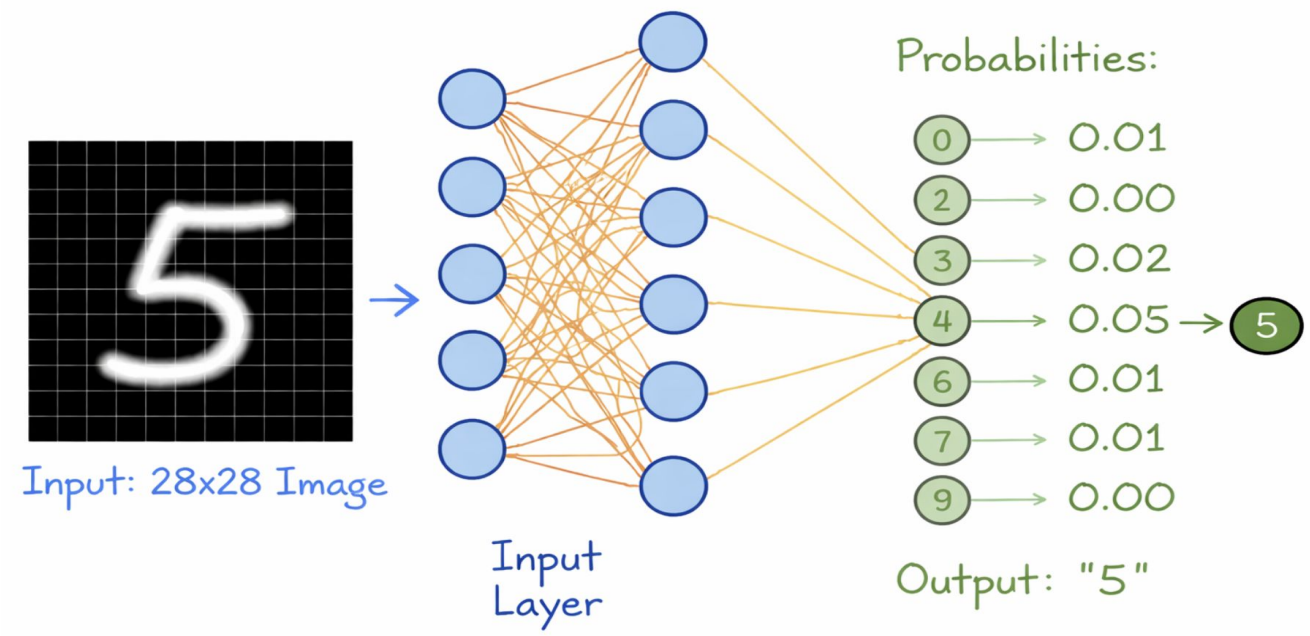
What is a neural network and how to make one?

The problem:

We have something that we want to predict (the output) given a huge amount of data the (input).

Neural networks are simply functions from the input to the output.

Example


Input: 28x28 Image
Input Layer
Probabilities:
0 → 0.01
2 → 0.00
3 → 0.02
4 → 0.05 → 5
6 → 0.01
7 → 0.01
9 → 0.00
Output: "5"

What is a neural network and how to make one?

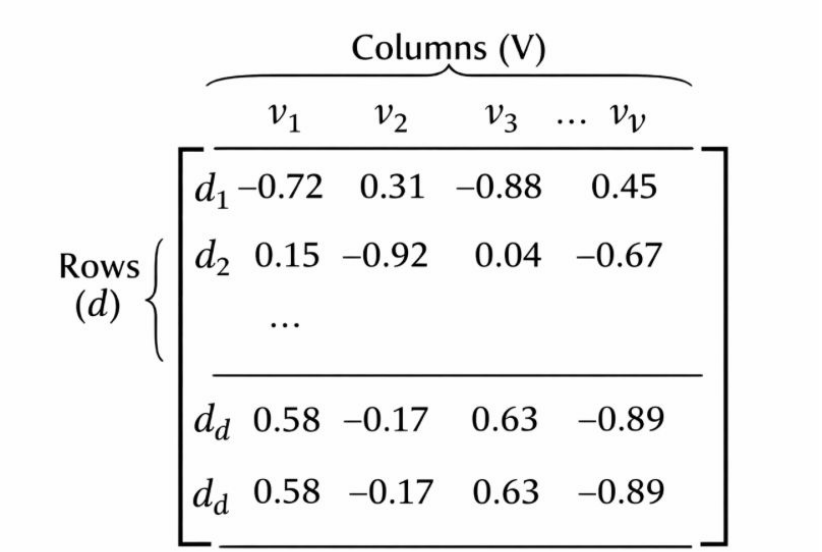
The basic building blocks are three steps:

- **Forward pass**: Data is compressed into a smaller space, and a final function outputs our predictions
  - Typically a combination of linear transformations and non-linear activation functions
  - An activation function has the ability to ‘turn-on/off’ components in a vector
- **Loss function**: Used to measure how far our predictions are from ‘ground truths’
- **Backward pass**: Our function is then iteratively improved via an optimization protocol (typically some variant of gradient descent)

Linear Transformations and non-linear activation functions

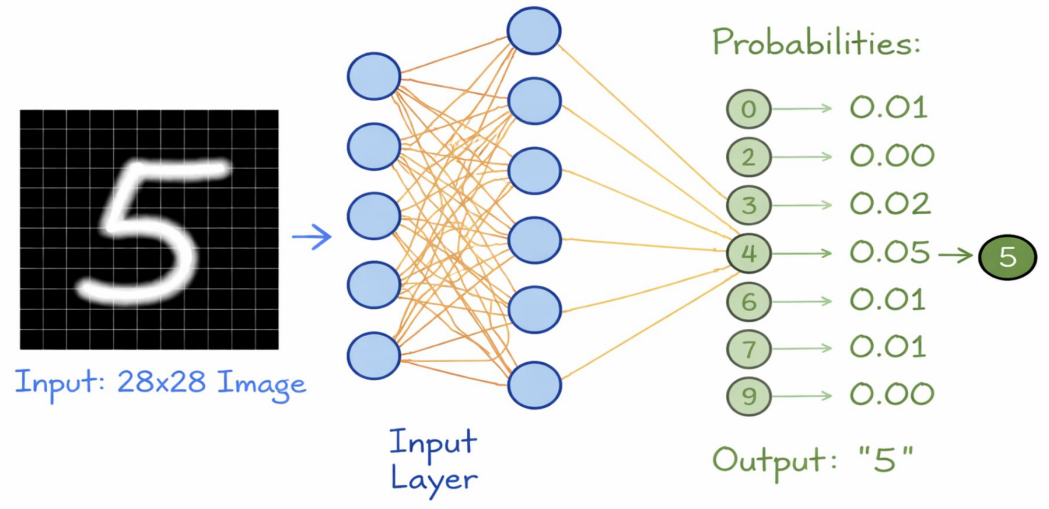
A linear transformation is typically represented as a matrix $\mathbb{R}^V \dashrightarrow \mathbb{R}^d$.

A non-linear activation function is typically of the form $\mathbb{R}^d \dashrightarrow \mathbb{R}^d$ where we set components to 0 if they're less than some threshold c in $\mathbb{R}$.



Loss functions

A loss function has to be chosen rather carefully based on the task at hand. Since we're working with vector spaces, they're typically some notion of 'distance' between 'predicted' vectors and underlying ground truths.



The output here is a vector that we interpret as probabilities.

A good choice of a loss function is a way to measure how far a probability distribution is from another.

Subject of 'information theory'.

Entropy and coin flipping

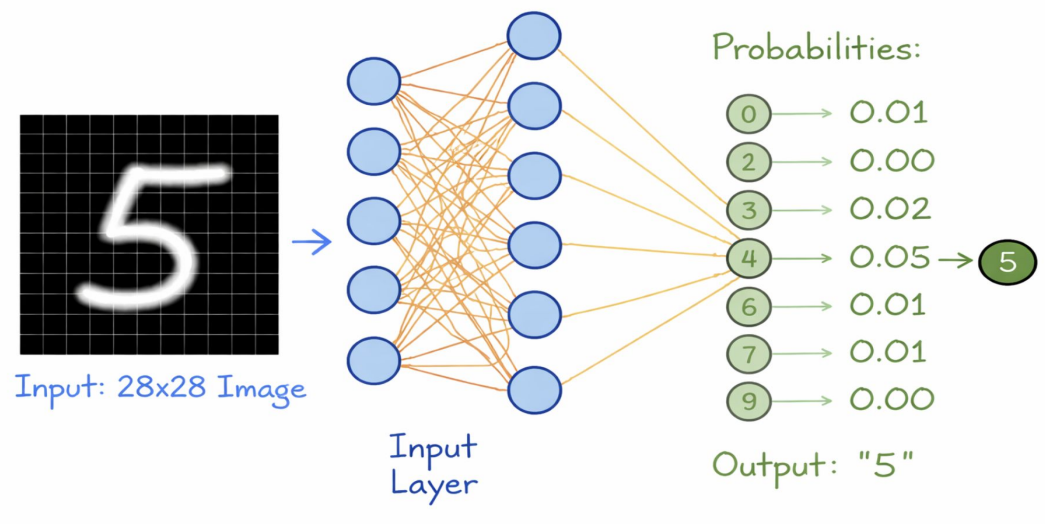
Entropy is a notion that originated from thermodynamics to describe the states of many particle systems.

In information theory, given a probability distribution P, it's entropy is given by $\mathbb{E}$ (-log(P)).

For coin flips, our probability distribution is (P=1 prob=p, P=0 prob=1-p), so the entropy is -p log p - (1-p)log(1-p).

In the case p=½, we get entropy = 1, and in the case p=0, we get 0.

Back to our example



Here, our outputs are a probability distribution P.

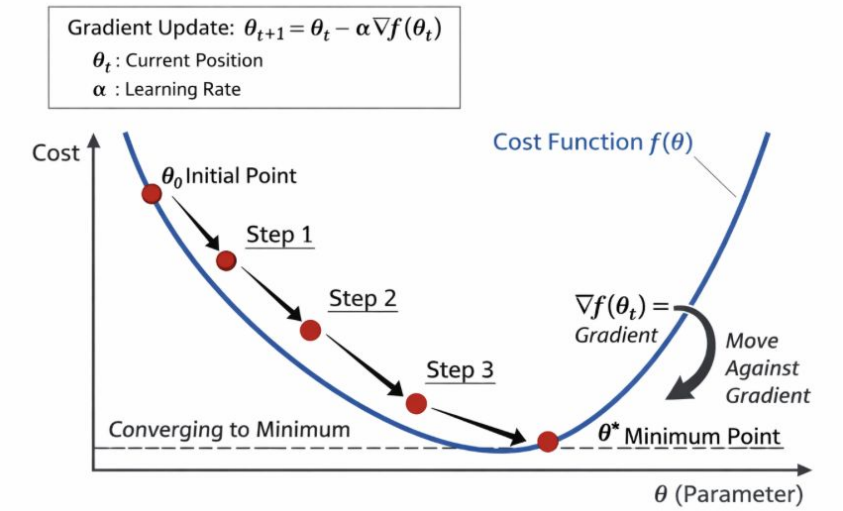
We want our function to be close to the ‘dirac’ distribution Q which has Prob(5) = 1.

A loss function we could choose is the relative entropy H(P || Q)

The optimization protocol

The optimization protocol aims to solving the following problem:

Given some parameters X, ground truths y, and a loss function Loss(y,X), what find X such that Loss(y,X) is minimized.



There are many different choices of optimization protocols.

Question: Explain the Adams optimizer and why it was developed?

A brief history of LLMs

- ~1940s: Mccullough and Pitts conceived the idea of a neural network
- ~1950s: F. Rosenblatt introduced the perceptron
- ~1960s: S. Amari and H. Saito proposes the first deep learning neural network trained with gradient descent
- ~1980s: D. Rumelhart, G. Hinton and G. Williams popularized the use of back-propagation in neural networks
- ~2000s: GPUs become integrated with deep learning, and many different language learning algorithms became trainable (word2vec, glove, etc…)
- 2017: ‘Attention is all you need’ paper, detailed a highly scalable model, the Transformer, able to capture complex interrelationships of the input
- 2018-: Large Language Models as we know them today were developed.

The Transformer

Issue: Human language has strong temporal relationships.

How can we model this?

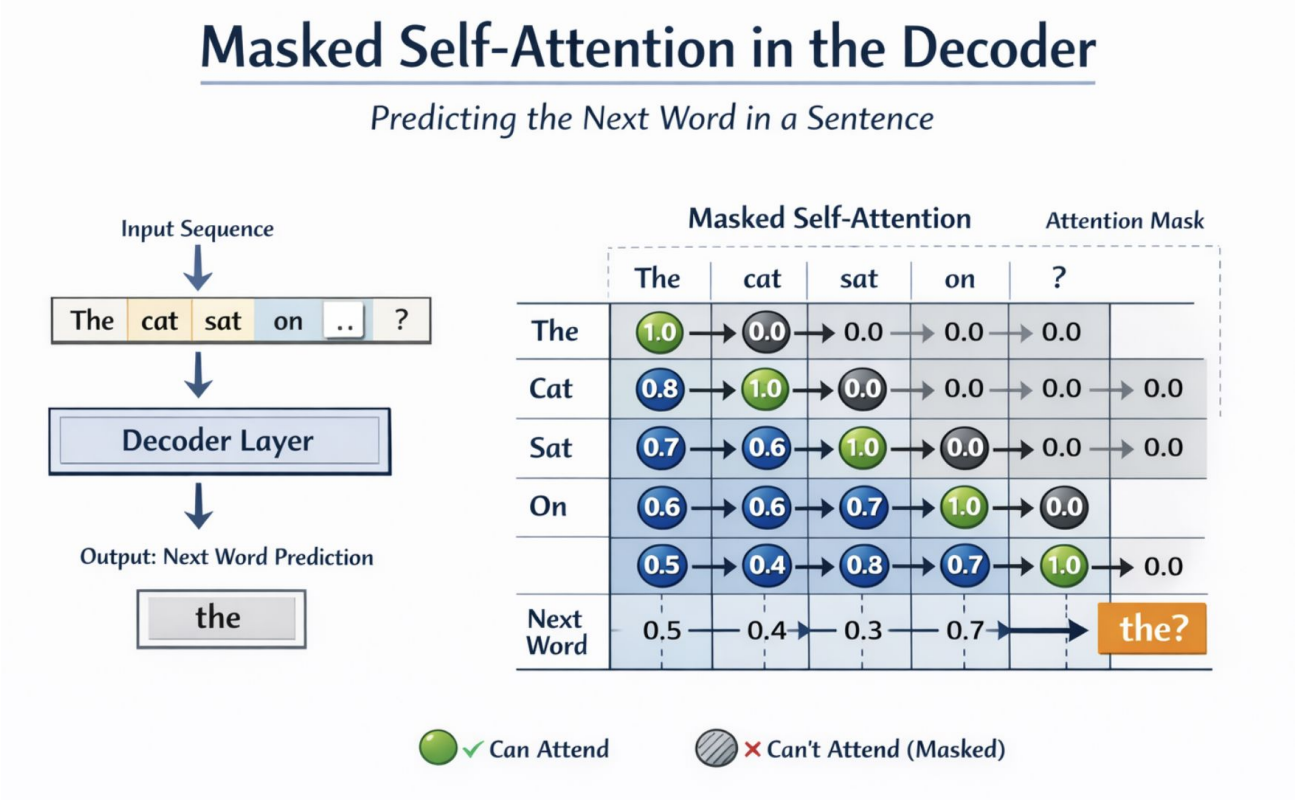
Key: Let the computer learn how to ‘attend’ to parts of sentences.

The set-up:

- Inputs: Words (# words = V), <START> and <END>, $\mathbb{R}^V$ vector space
- Hidden layers: an $\mathbb{R}^d$ space, d << V
- Output: vector in $\mathbb{R}^V$ representing ‘probabilities’

In essence, a ‘matrix factorization’: $\mathbb{R}^V \dashrightarrow \mathbb{R}^d \dashrightarrow \mathbb{R}^V$

The Transformer



Outcomes

- Human language can be accurately understood by linear algebra and activation functions
- The compressed representations of language captured universal features that can be transferred to other tasks such as:
  - Use news data to predict stock price fluctuations
  - Spam prediction
  - Recommendation systems
  - Speech recognition
  - And more
- Fine-tuning: With low computation cost, we can optimize an LLM for new tasks
  - Chain of thought reasoning: We might feed in solutions to math problems with step-by-step proofs to teach an LLM to provide sequential reasoning

Language Models on the market

- ChatGPT:
  - Original implementer of the ‘decoder transformer’ for language, MCoT
  - All-round good, general purpose AI
- Gemini:
  - Natively multimodal, MCoT
  - All-round good, general purpose AI
- Claude:
  - Cautionary, uses prompts between stages in CoT to better align itself
  - Excels at more technical tasks
- Deepseek:
  - Like Claude, tries to perform more careful reasoning, MCoT
  - Excels at more complicated tasks

LLMs and math proofs

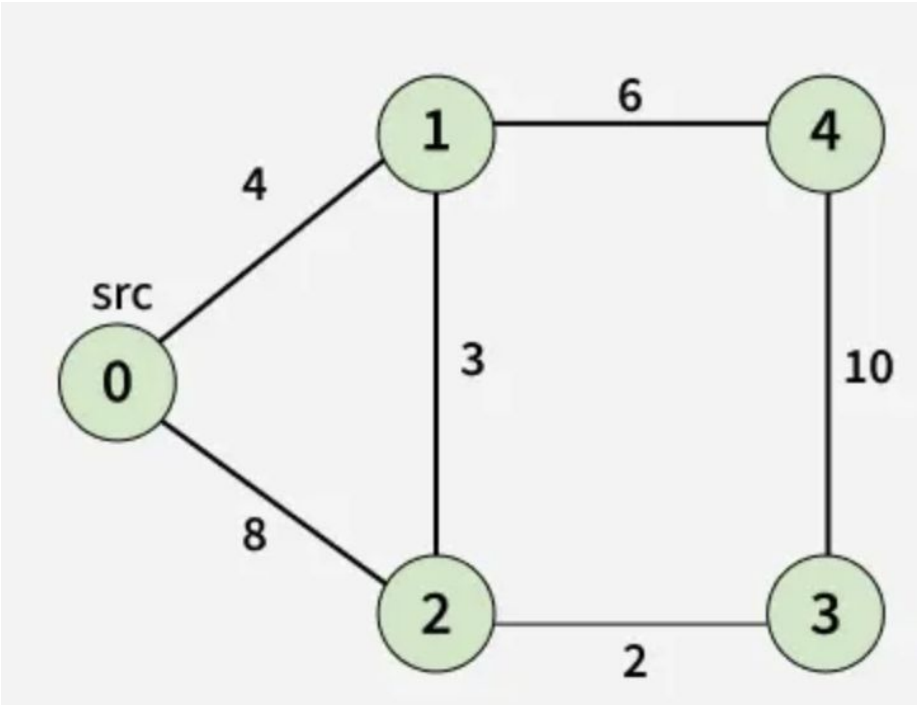
- LLMs have showed an ability to solve contest mathematics problems.
- They have been also shown to help mathematics researchers
- Surprisingly, current off-the-shelf LLMs do not ‘understand’ when they’ve proven something

FAQs:

- Do I trust AI to be correct in general? NO!
- What tasks do I use AI for? Searching and pattern recognition
- Do I think they will be able to do independent mathematical research? Maybe

Learning in Games

There is one key principle, Bellman's principle.



AlphaZero

In 2017, Deepmind developed a novel chess-playing engine that integrated neural works into a game-learning architecture