Mechanistic Interpretability

Part 1: Introduction and Transformer circuits

Sondre Wold

University of Oslo



In this lecture:

1. What do we know about neural networks and LLMs?
   - Theoretical limitations
2. Mechanistic interpretability
3. The circuits framework

Measures of progress in ML

- Downstream performance provides a measure of utility, but it does not explain *how* a model works.

Measures of progress in ML

- Downstream performance provides a measure of utility, but it does not explain *how* a model works.
- LLMs are not like linear regression models.
- Unsupervised representation learning is extremely efficient but intellectually unsatisfying.
- Science uses empirical methods to construct new knowledge, but ML typically stops at the empirical part.

The expressive power of NNs

- We know from mathematical theory that NNs can approximate any f within $\epsilon > 0$ closeness.
- Most famously: a single hidden layer network with a non-linear activation function can approximate any f, given enough neurons (arbitrary width) (Hornik et al., 1989; Hornik, 1991).

The expressive power of NNs

- We know from mathematical theory that NNs can approximate any f within $\epsilon > 0$ closeness.
- Most famously: a single hidden layer network with a non-linear activation function can approximate any f, given enough neurons (arbitrary width) (Hornik et al., 1989; Hornik, 1991).
- Using computability theory, we can also investigate the expressiveness of NNs using formal constructs such as Turing machines (Siegelmann and Sontag, 1992).
- Given unbounded precision (unrealistic), RNNs are known to be Turing complete.

LLMs as neural networks

- LLMs use Transformers, what do we know about them?

LLMs as neural networks

- LLMs use Transformers, what do we know about them?
- We still have universal function approximation.
- Given strong assumptions (infinite precision, infinite width MLPs), Transformers are also Turing complete (Pérez et al., 2019).

LLMs as neural networks

- LLMs use Transformers, what do we know about them?
- We still have universal function approximation.
- Given strong assumptions (infinite precision, infinite width MLPs), Transformers are also Turing complete (Pérez et al., 2019).
- But in reality, Transformers:
  - cannot compute PARITY, a basic regular language that contains all bit-strings with an even number of 1's (Hahn, 2020).
  - suffer from representational collapse: the representation of different sequences converge to the same point as sequence-length increases (Barbero et al., 2024).

RASP

- What primitives do we have in a Transformer?

RASP

- What primitives do we have in a Transformer?
  - MLP
  - Attention

RASP

- What primitives do we have in a Transformer?
  - MLP
  - Attention
- These can be used to build a programming langauge!
- Restricted Access Sequence Processing Language (RASP) (Weiss et al., 2021).
- Key result: Everything you can program in RASP, can be learned by a Transformer!

RASP

- What primitives do we have in a Transformer?
  - MLP
  - Attention
- These can be used to build a programming langauge!
- Restricted Access Sequence Processing Language (RASP) (Weiss et al., 2021).
- Key result: Everything you can program in RASP, can be learned by a Transformer!
- . . . and that is important because?

In this lecture:

1. What do we know about neural networks and LLMs?
   - Theoretical limitations
2. Mechanistic interpretability
3. The circuits framework

Mechanistic interpretability

- Mechanistic interpretability seeks to reverse-engineer neural networks.
- Common analogy, from Chris Olah, 2022:

Computers	**Neural networks**
Reverse engineer	Mechanistic interpretability
Program binary	Network parameters
Interpreter	Network Architecture
Program state	Activations
Variable / Memory location	Neuron / Feature direction

Breaking the curse of dimensionality

Given a continuous function f, how many samples of do we need to approximate f in general?

Breaking the curse of dimensionality

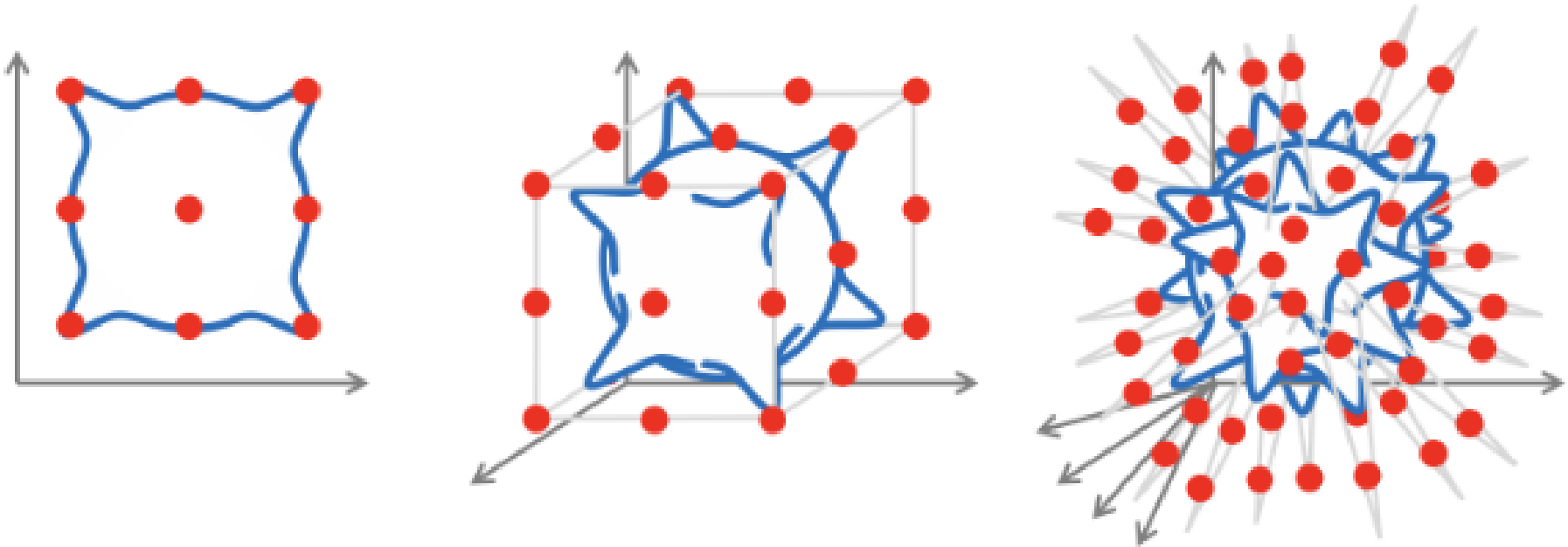


Figure: Illustration taken from Bronstein et al. (2021).

The number of samples we need to approximate f is exponential in the number of dimensions!

Breaking the curse of dimensionality

How can we learn a function over a high-dimensional space (such as language) without needing exponential amounts of data? And perhaps even more pressing: How the heck do you interpret such a function?

Breaking the curse of dimensionality

How can we learn a function over a high-dimensional space (such as language) without needing exponential amounts of data? And perhaps even more pressing: How the heck do you interpret such a function?

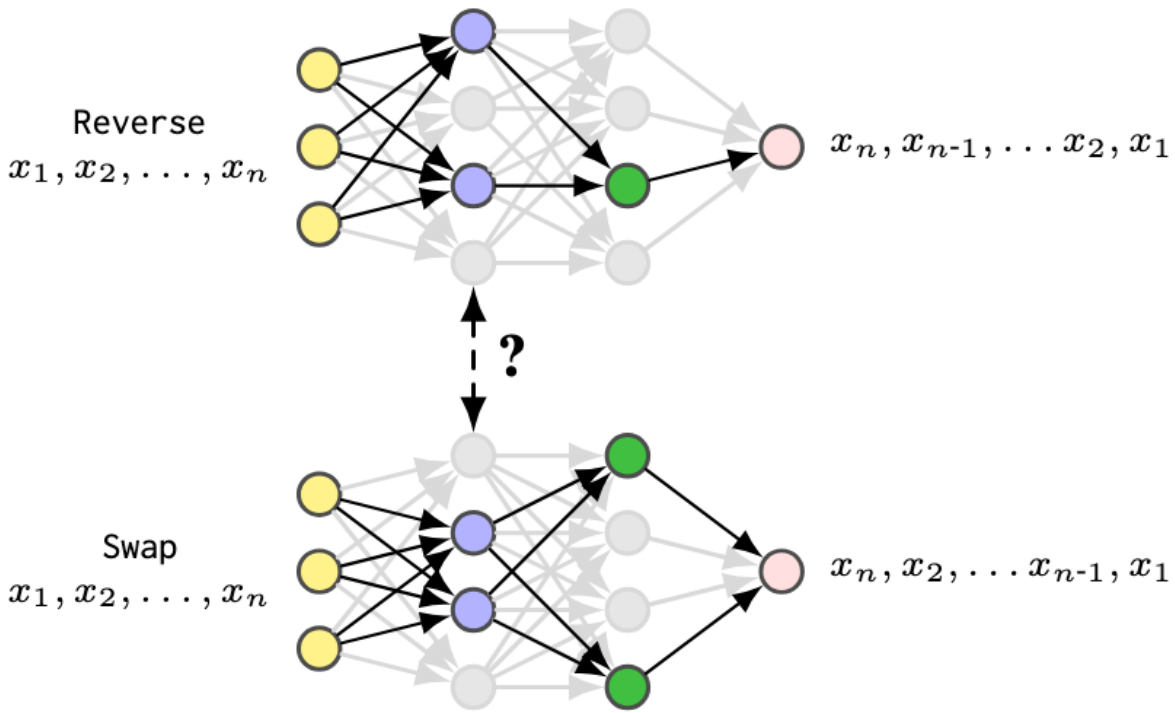
We need to **decompose**.

In this lecture:

1. What do we know about neural networks and LLMs?
   - Theoretical limitations
2. Mechanistic interpretability
3. The circuits framework

Circuits

One way to decompose is to identify so-called *circuits*: a computational subgraph responsible for some behaviour.



Circuits

Circuits can, admittedly with some anthropomorphism, explain high-level behaviour, such as (in GPT-2):

Circuits

Circuits can, admittedly with some anthropomorphism, explain high-level behaviour, such as (in GPT-2):

- Indirect object identification (Wang et al., 2022)
  - When Mary and John went to the store, John gave a drink to _

Circuits

Circuits can, admittedly with some anthropomorphism, explain high-level behaviour, such as (in GPT-2):

- Indirect object identification (Wang et al., 2022)
  - When Mary and John went to the store, John gave a drink to _
- Greater-than relations Hanna et al. (2023)
  - The war lasted from the year 1732 to the year 17x ($x > 32$)

Circuits

Circuits can, admittedly with some anthropomorphism, explain high-level behaviour, such as (in GPT-2):

- Indirect object identification (Wang et al., 2022)
  - When Mary and John went to the store, John gave a drink to _
- Greater-than relations Hanna et al. (2023)
  - The war lasted from the year 1732 to the year 17x ($x > 32$)
- Causal mediation analysis/patching.

Circuits

But we can do even better if we:

Circuits

But we can do even better if we:

1. Scale down our models **a lot**.

Circuits

But we can do even better if we:

1. Scale down our models **a lot**.

2. View a Transformer as a series of decoupled transformations.

Circuits

But we can do even better if we:

1. Scale down our models **a lot**.
2. View a Transformer as a series of decoupled transformations.
3. Start by understanding very basic principles.

A Mathematical Framework for Transformer Circuits



Elhage et al. (2021) proposes a more principled way of studying Transformers, centered around the idea of the *residual stream*:

A Mathematical Framework for Transformer Circuits



Elhage et al. (2021) proposes a more principled way of studying Transformers, centered around the idea of the *residual stream*:

```
    def TransformerBlockForward(x: Tensor) -> Tensor:
      x = x + Attention(x)
      x = LayerNorm(x)
      x = x + MLP(x)
      return x
```

A Mathematical Framework for Transformer Circuits

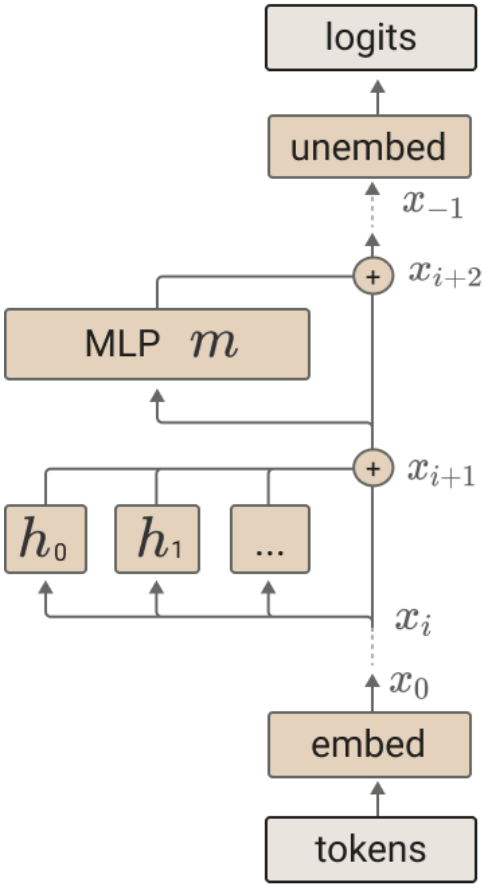


Elhage et al. (2021) proposes a more principled way of studying Transformers, centered around the idea of the *residual stream*:

```
    def TransformerBlockForward(x: Tensor) -> Tensor:
      x = x + Attention(x)
      x = LayerNorm(x)
      x = x + MLP(x)
      return x
```

The output of the Attention and the MLP is *added* back into the "stream".

A Transformer with 1 block



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_{-1}$$

An MLP layer, m, is run and added to the residual stream.

$$x_{i+2} = x_{i+1} + m(x_{i+1})$$

Each attention head, h, is run and added to the residual stream.

$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$$

One residual block

Token embedding.

$$x_0 = W_E t$$

Figure: Illustration taken from Elhage et al. (2021)

Attention heads

In this course, we implemented multi-head attention like this, following the original paper:

```
    def MHAttention(x: Tensor) -> Tensor:
      queries = self.query(x)
      queries = queries.view(batch_size,
        seq_len,
        self.heads,
        hidden // self.heads)
      .
      .
      .
      scaled_values = attention_probabilities @ values
      # Concatenate all the heads (as per the paper)
      # Shape: [B, H, T_q, D_h] -> [B, T_q, H, D_h]
          -> [B, T_q, D]
      concat_heads = scaled_values.transpose(1,
          2).flatten(2, 3)
      return self.output(concat_heads)
```

Attention heads

The concatenation of the heads makes for more efficient compute, but we can also view heads as independent operations:

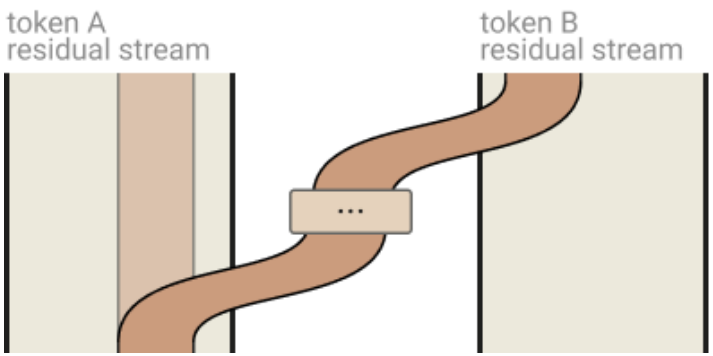
$$W_O^H \begin{bmatrix} r^{h1} \\ r^{h2} \\ \dots \end{bmatrix} = \left[W_O^{h1}, W_O^{h2}, \dots \right] \begin{bmatrix} r^{h1} \\ r^{h2} \\ \dots \end{bmatrix} = \sum_i W_O^{h_i} r^{h_i}$$

Consequently, heads can be understood as *independently additive*.

What does each head do?

Attention heads **move information**:

- W_q and W_k compute the attention pattern, which moves information between tokens.
- W_v and W_o copy information from a source token to a destination token in the residual stream.



Attention heads copy information from the residual stream of one token to the residual stream of another. They typically write to a different subspace than they read from.

Figure: Illustration from Elhage et al. (2021)

Induction heads

- Because W_q and W_k always appear together, and the same for W_v and W_o, these can be seen as the QK and VO **circuits**.

Induction heads

- Because W_q and W_k always appear together, and the same for W_v and W_o, these can be seen as the QK and VO **circuits**.
- Together, these circuits form what is know as *induction heads*.

Induction heads

- Because W_q and W_k always appear together, and the same for W_v and W_o, these can be seen as the QK and VO **circuits**.
- Together, these circuits form what is know as *induction heads*.
- Induction heads look for previous instances of the same token in the sequence/context:
  - If they find it, they copy information from the next token.
  - If not, they look at the first token (which is often a special token).

Induction heads

- Because W_q and W_k always appear together, and the same for W_v and W_o, these can be seen as the QK and VO **circuits**.
- Together, these circuits form what is know as *induction heads*.
- Induction heads look for previous instances of the same token in the sequence/context:
  - If they find it, they copy information from the next token.
  - If not, they look at the first token (which is often a special token).
- This is a general algorithm for **in-context learning**.

Induction head

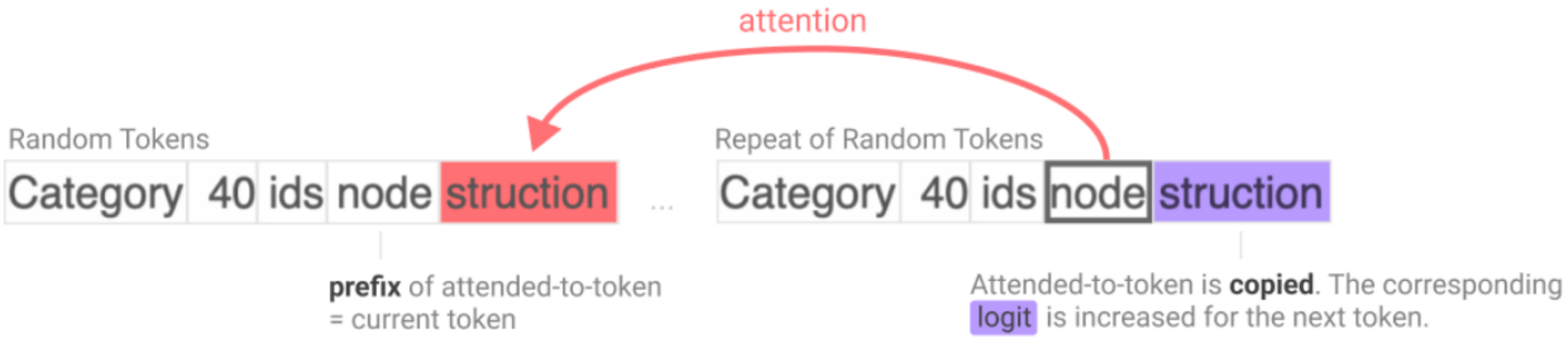


Figure: Illustration taken from Olsson et al. (2022)

Conclusion

- LLMs, as neural networks, suffer from inherent limitations.

Conclusion

- LLMs, as neural networks, suffer from inherent limitations.
- We can study Transformers as general function approximators or through computability theory.

Conclusion

- LLMs, as neural networks, suffer from inherent limitations.
- We can study Transformers as general function approximators or through computability theory.
- If we downscale our models, we can actually get a mechanistic understanding of how Transformers function, at least on a small scale.

Conclusion

- LLMs, as neural networks, suffer from inherent limitations.
- We can study Transformers as general function approximators or through computability theory.
- If we downscale our models, we can actually get a mechanistic understanding of how Transformers function, at least on a small scale.
- By decoupling and rewriting attention heads, we have discovered a general algorithm that explains in-context learning.

References I

Barbero, F., Banino, A., Kapturowski, S., Kumaran, D., Madeira Araújo, J., Vitvitskyi, O., Pascanu, R., and Veličković, P. (2024). Transformers need glasses! information over-squashing in language tasks. Advances in Neural Information Processing Systems, 37:98111–98142.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. (2021). Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. arXiv preprint arXiv:2104.13478.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. (2021). A mathematical framework for transformer circuits. Transformer Circuits Thread. https://transformer-circuits.pub/2021/framework/index.html.

References II

Hahn, M. (2020). Theoretical limitations of self-attention in neural sequence models. Transactions of the Association for Computational Linguistics, 8:156–171.

Hanna, M., Liu, O., and Variengien, A. (2023). How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. Advances in Neural Information Processing Systems, 36:76033–76060.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2):251–257.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366.

References III

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. (2022). In-context learning and induction heads. Transformer Circuits Thread. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Pérez, J., Marinković, J., and Barceló, P. (2019). On the turing completeness of modern neural network architectures. In International Conference on Learning Representations.

Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. In Proceedings of the fifth annual workshop on Computational learning theory, pages 440–449.

References IV

Wang, K. R., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. (2022). Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. In The Eleventh International Conference on Learning Representations.

Weiss, G., Goldberg, Y., and Yahav, E. (2021). Thinking like transformers. In Meila, M. and Zhang, T., editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 11080–11090. PMLR.