Stanford CS224W: Graph Transformers

CS224W: Machine Learning with Graphs

Joshua Robinson, Stanford University

http://cs224w.stanford.edu

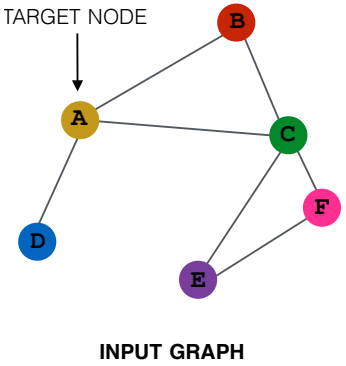
Announcements

- **Homework 3** due this Thursday
  - Late submissions accepted until end of day Monday, 11/20
- **Regrade request deadlines**
  - **Homework 2**: Saturday, 11/18
    - Solutions and statistics released on Ed

Announcements

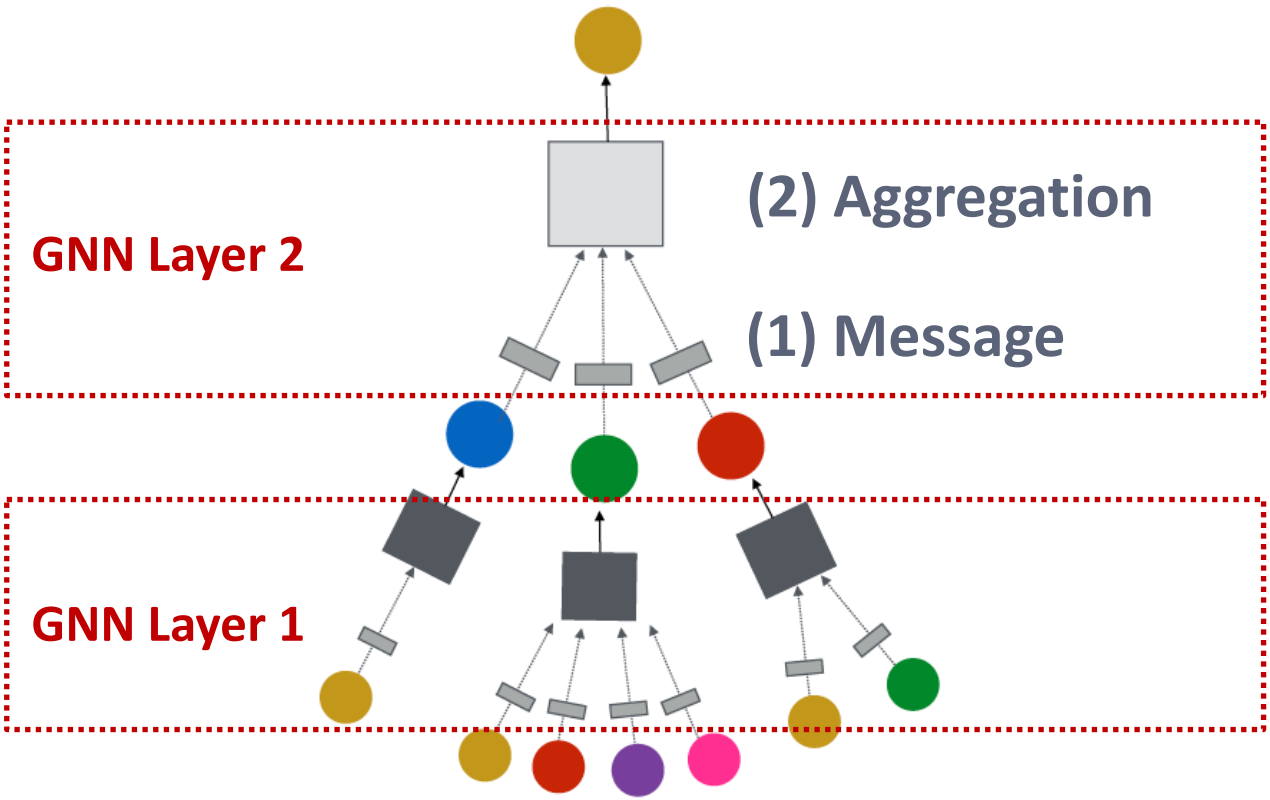
- **Thanksgivings office hours**
  - Wednesday, 11/22
    - 2pm-4pm, Thorton Center 207
    - **Also on Zoom** (link found under the Office Hours tab on the course website)
- **We are collecting feedback at https://tinyurl.com/cs224w-feedback**

Recap: A General GNN Framework



INPUT GRAPH

(5) Learning objective



(3) Layer connectivity

(4) Graph augmentation

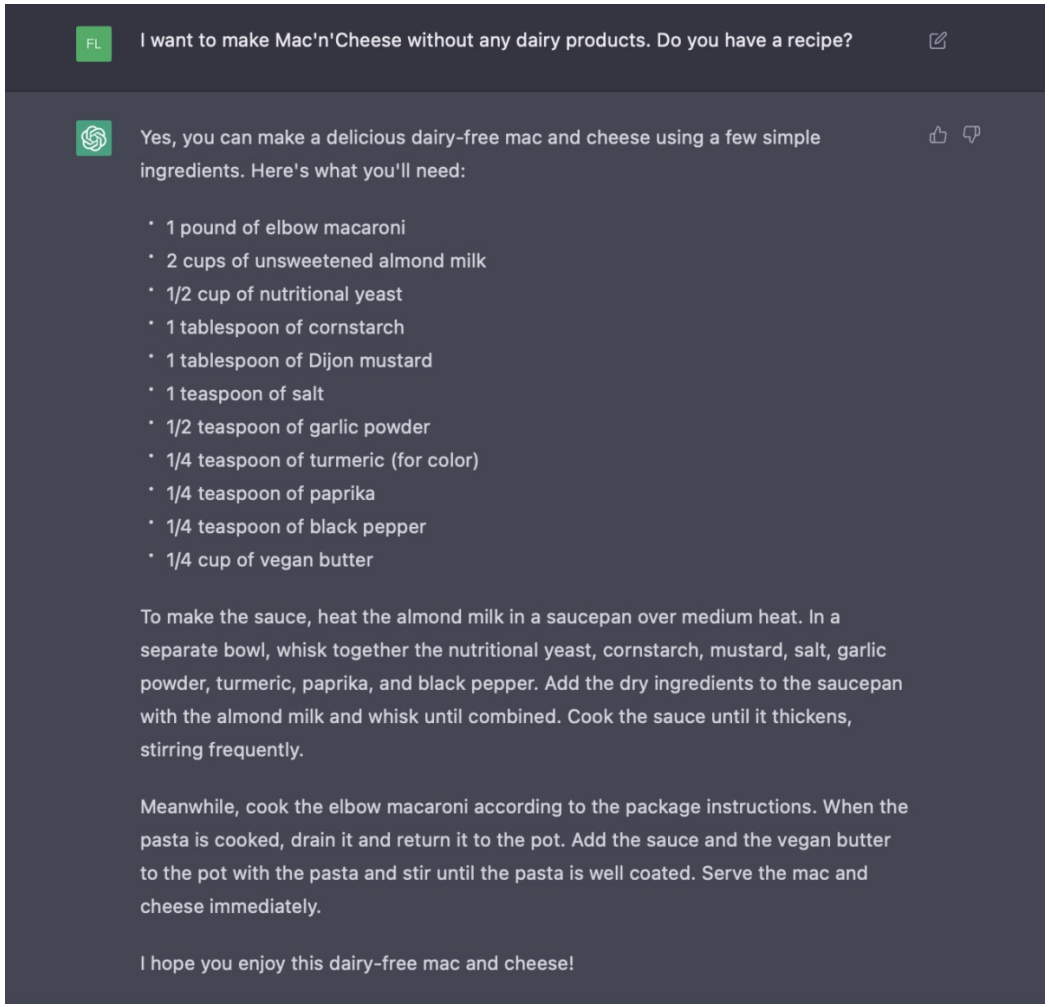
Stanford CS224W: Intro to Transformers

CS224W: Machine Learning with Graphs

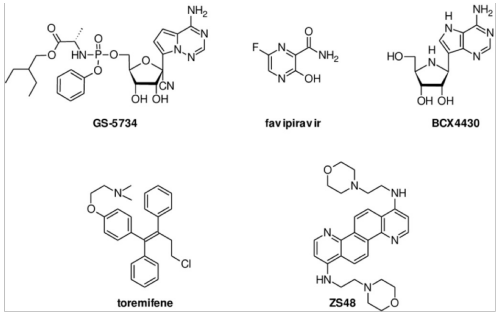
Joshua Robinson, Stanford University

http://cs224w.stanford.edu

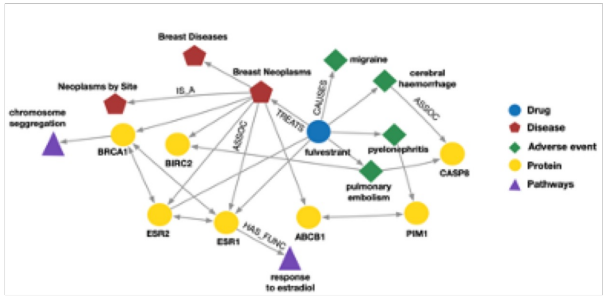
The Backbone of Generative AI



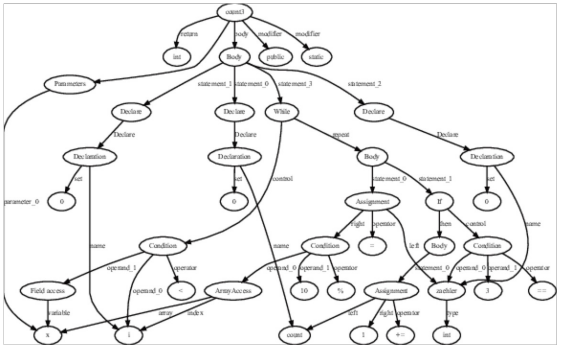
Goal: Bring Power of Transformers to Graphs



Small molecules



Knowledge graphs



Code graphs

Image credit: ResearchGate

Communication networks

Image credit: Lumen Learning

x

- **There is lots of multi-billon node/graph scale data to learn from**

Plan for Today

- **Part 1:**
  - Introducing Transformers
  - Relation to message passing GNNs
- **Part 2:**
  - A new design landscape for graph Transformers
- **Part 3:**
  - Sign invariant Laplacian positional encodings for graph Transformers

Stanford CS224W: Transformers

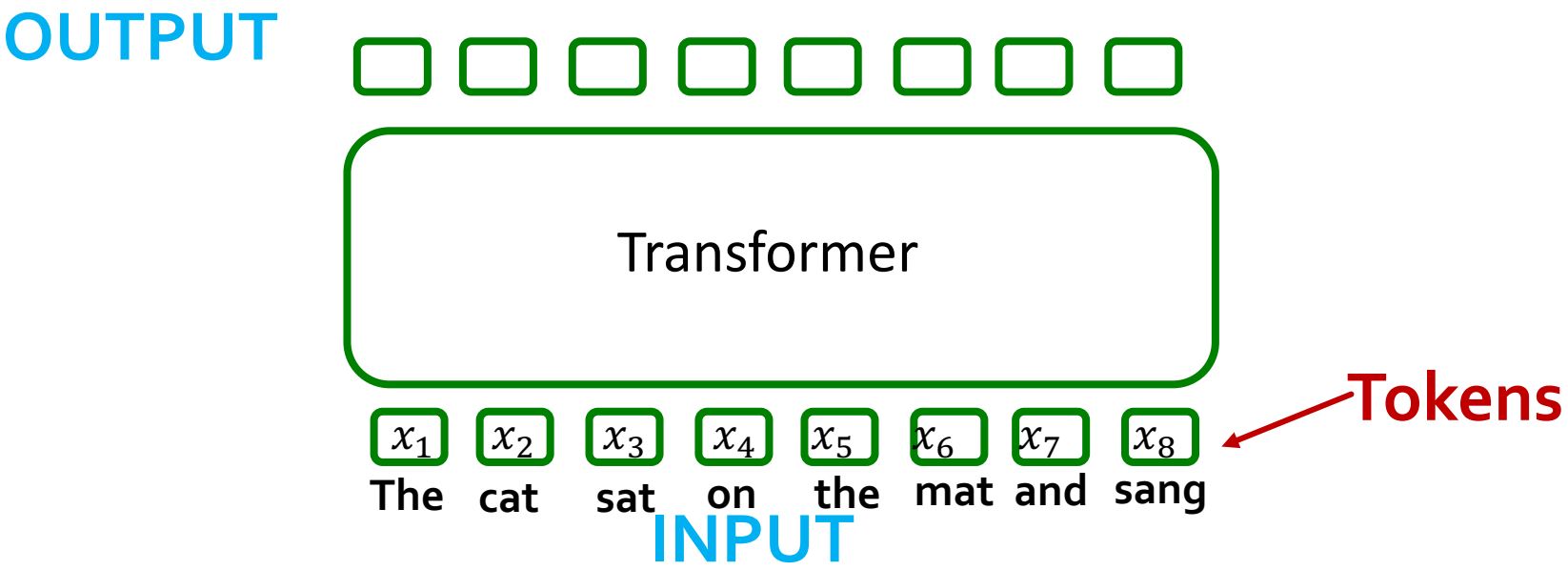
CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

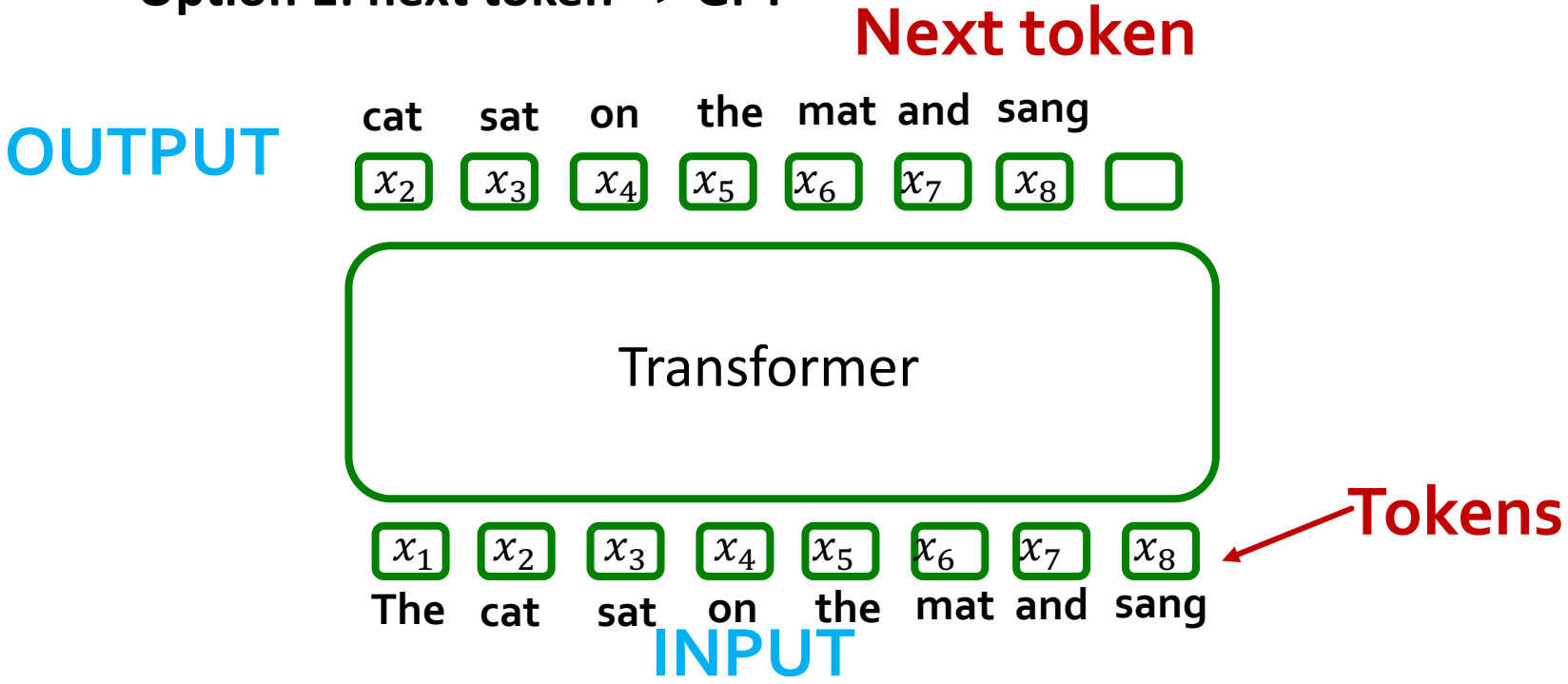
Transformers Ingest Tokens

- **Transformers map 1D sequences of vectors to 1D sequences of vectors**



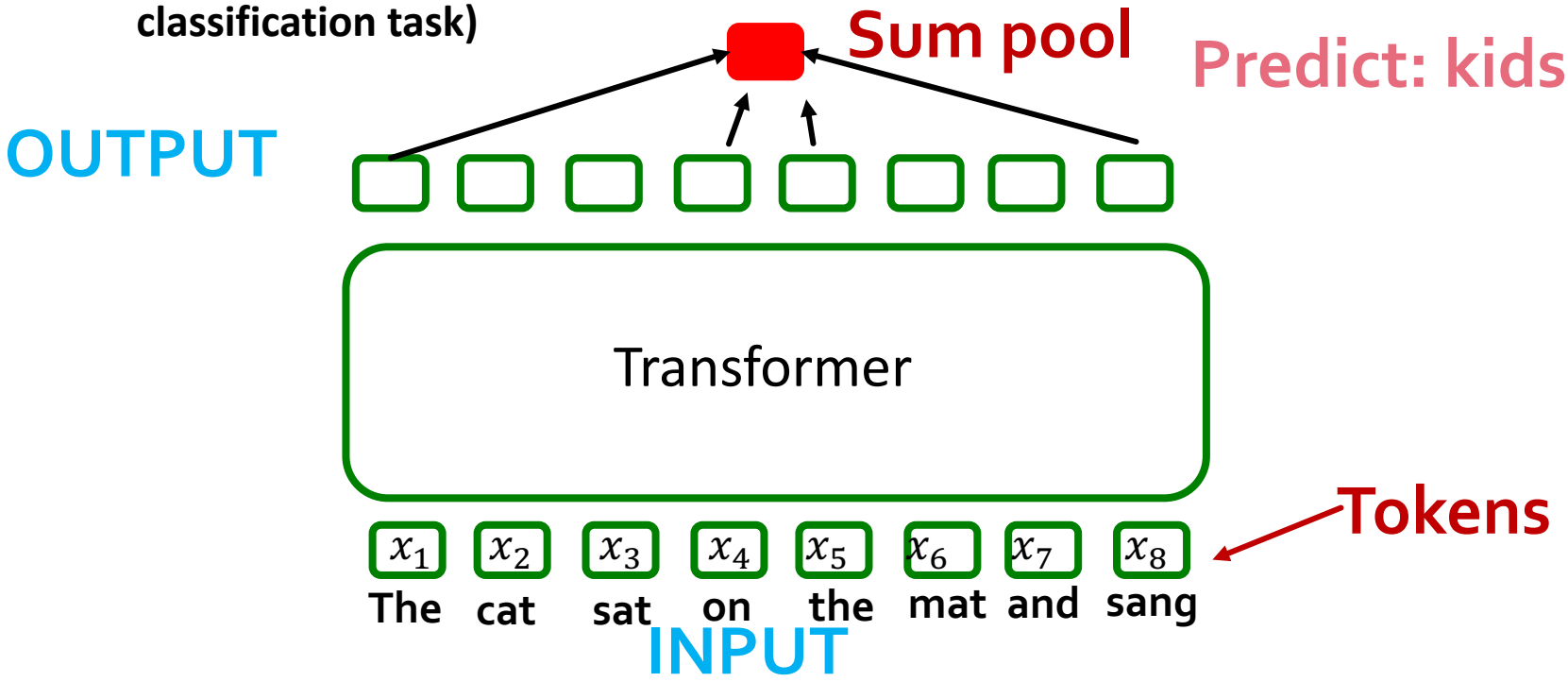
Transformers Ingest Tokens

- **Transformers map 1D sequences of vectors to 1D sequences of vectors known as tokens**
  - **Tokens describe a ”piece” of data – e.g., a word**
- **What output sequence?**
  - **Option 1: next token => GPT**



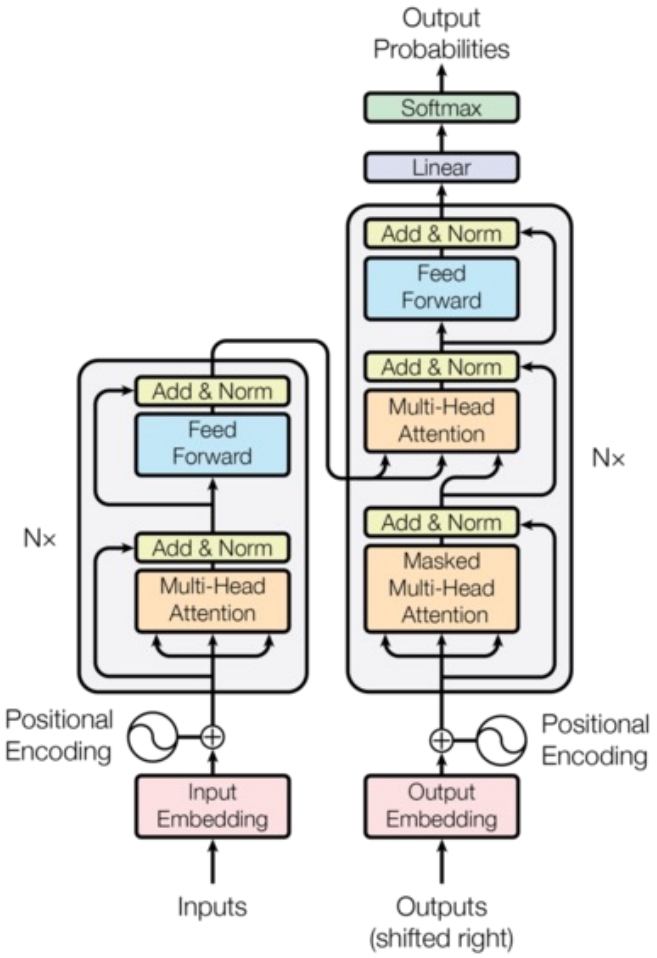
Transformers Ingest Tokens

- **Transformers map 1D sequences of vectors to 1D sequences of vectors known as tokens**
  - **Tokens describe a ”piece” of data – e.g., a word**
- **What output sequence?**
  - **Option 1: next token => GPT**
  - **Option 2: pool (e.g., sum-pool) to get sequence level-embedding (e.g., for classification task)**



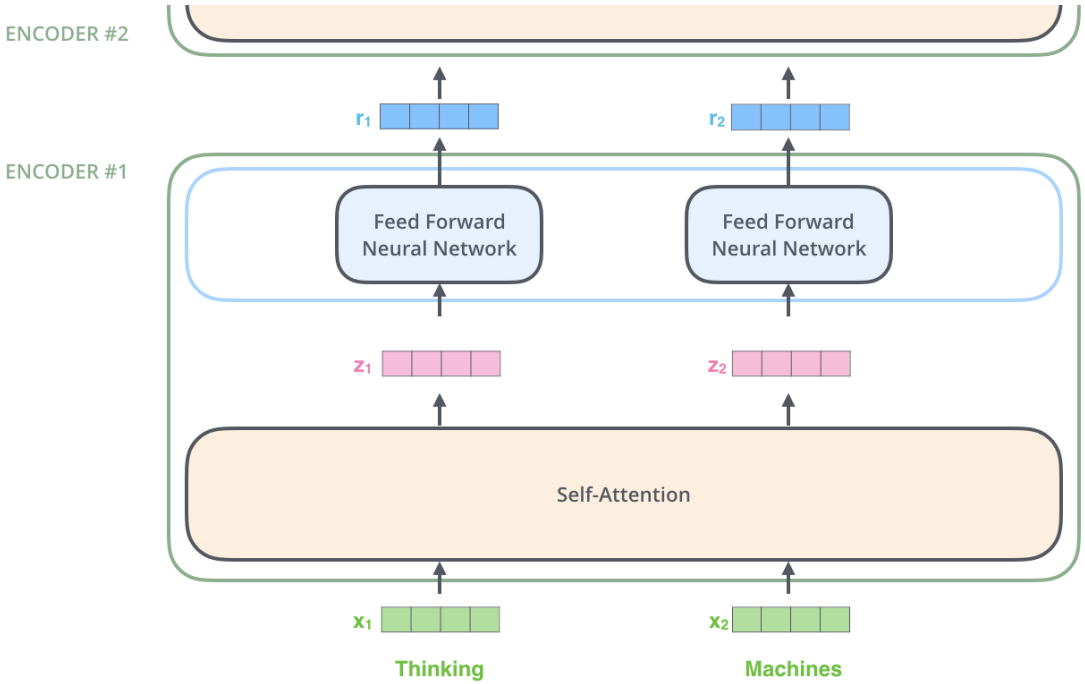
Transformer Blueprint

- How are tokens processed?
- Lots of components
  - Normalization
  - Feed forward networks
  - Positional encoding (more later)
  - **Multi-head self-attention**
- **What does self-attention block do?**



Self-attention

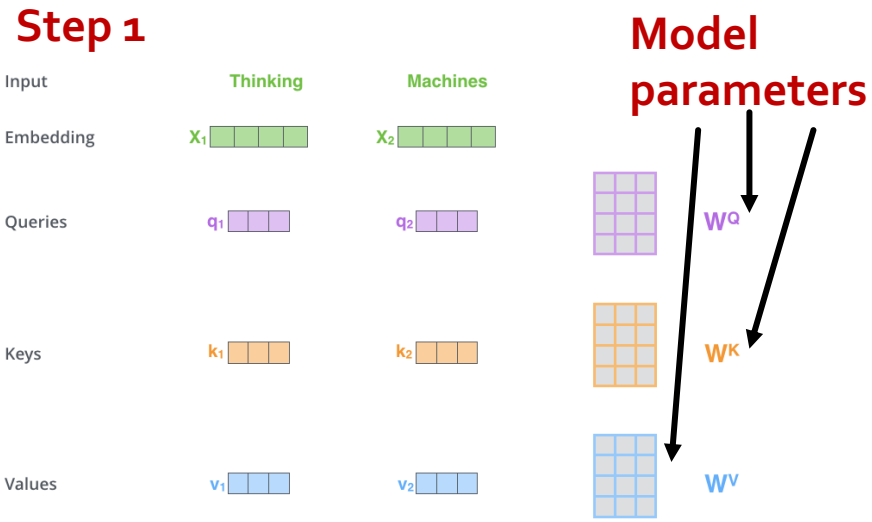
- **Before "multi-head" self-attention, what is "single head" self-attention?**



See: Illustrated Transformer tutorial, https://jalammar.github.io/illustrated-transformer/

Self-attention

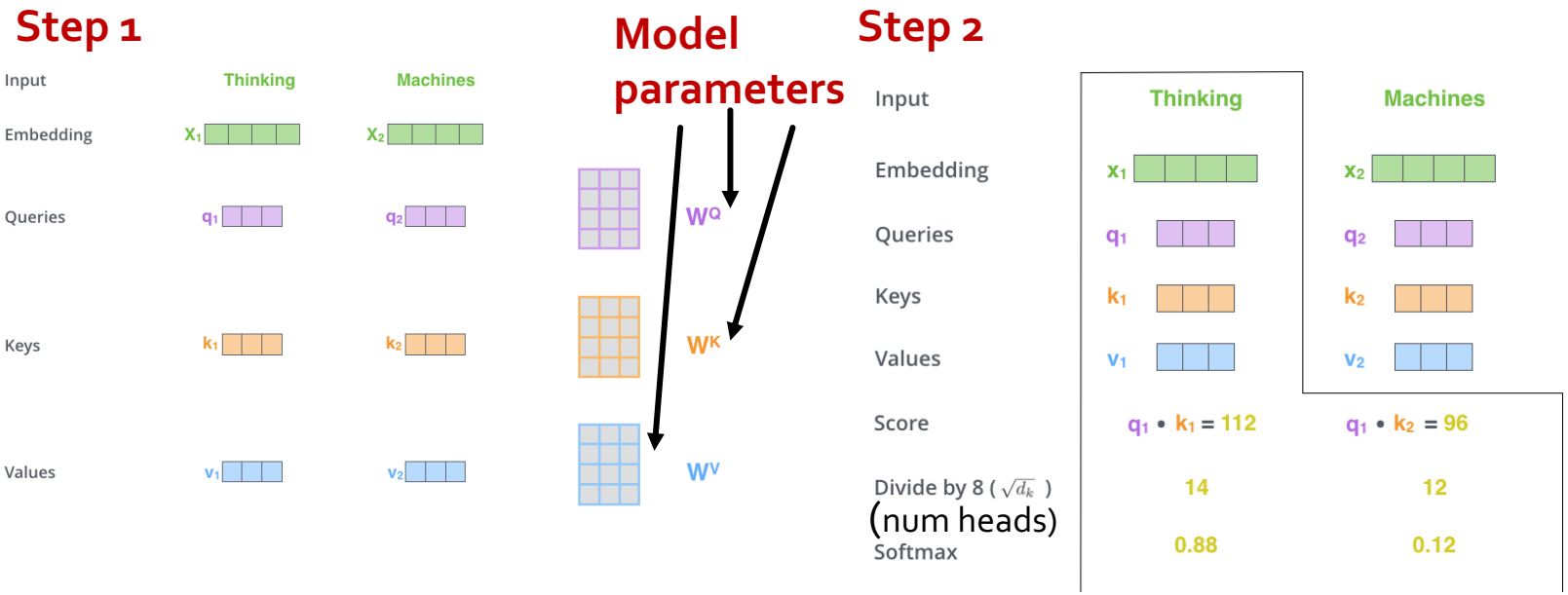
- **Step 1:** compute "key, value, query" for each input



See: Illustrated Transformer tutorial, https://jalammar.github.io/illustrated-transformer/

Self-attention

- **Step 1:** compute "key, value, query" for each input
- **Step 2 (just for x_1):** compute scores between pairs, turn into probabilities (same for x_2)

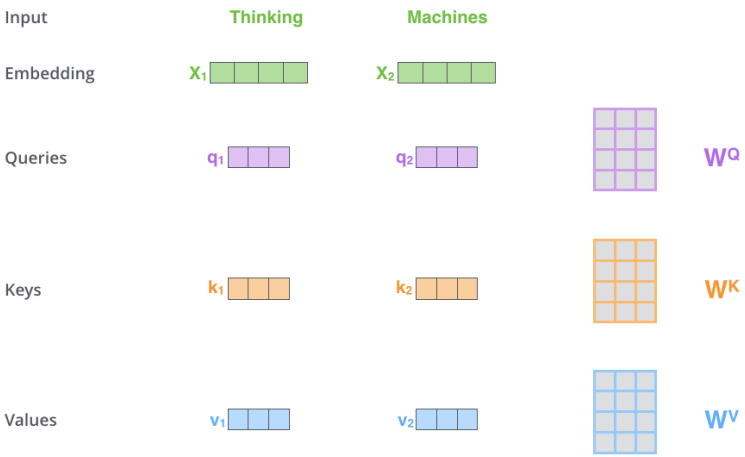


See: Illustrated Transformer tutorial, https://jalammar.github.io/illustrated-transformer/

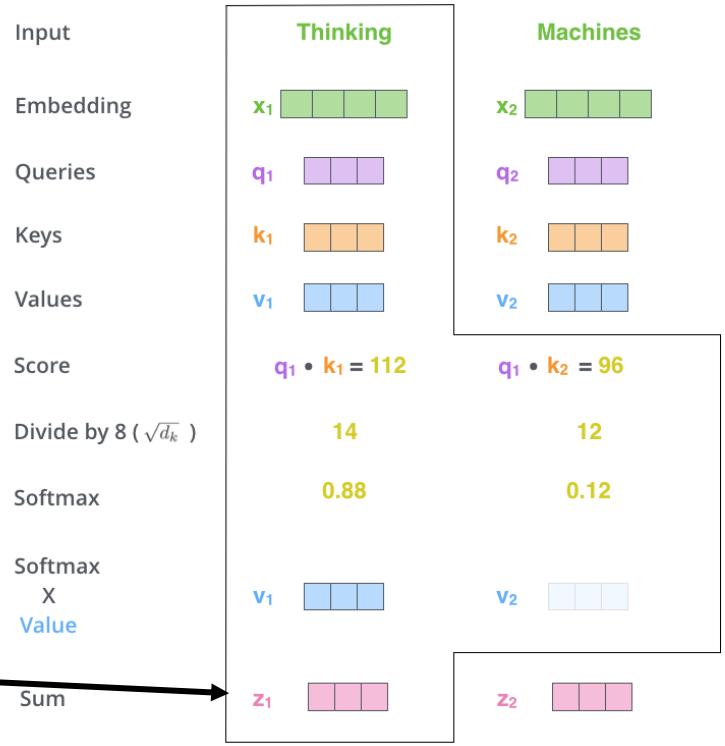
Self-attention

- **Step 1:** compute "key, value, query" for each input
- **Step 2 (just for x_1):** compute scores between pairs, turn into probabilities (same for x_2)
- **Step 3:** get new embedding z_1 by weighted sum of v_1, v_2

Step 1



Step 2



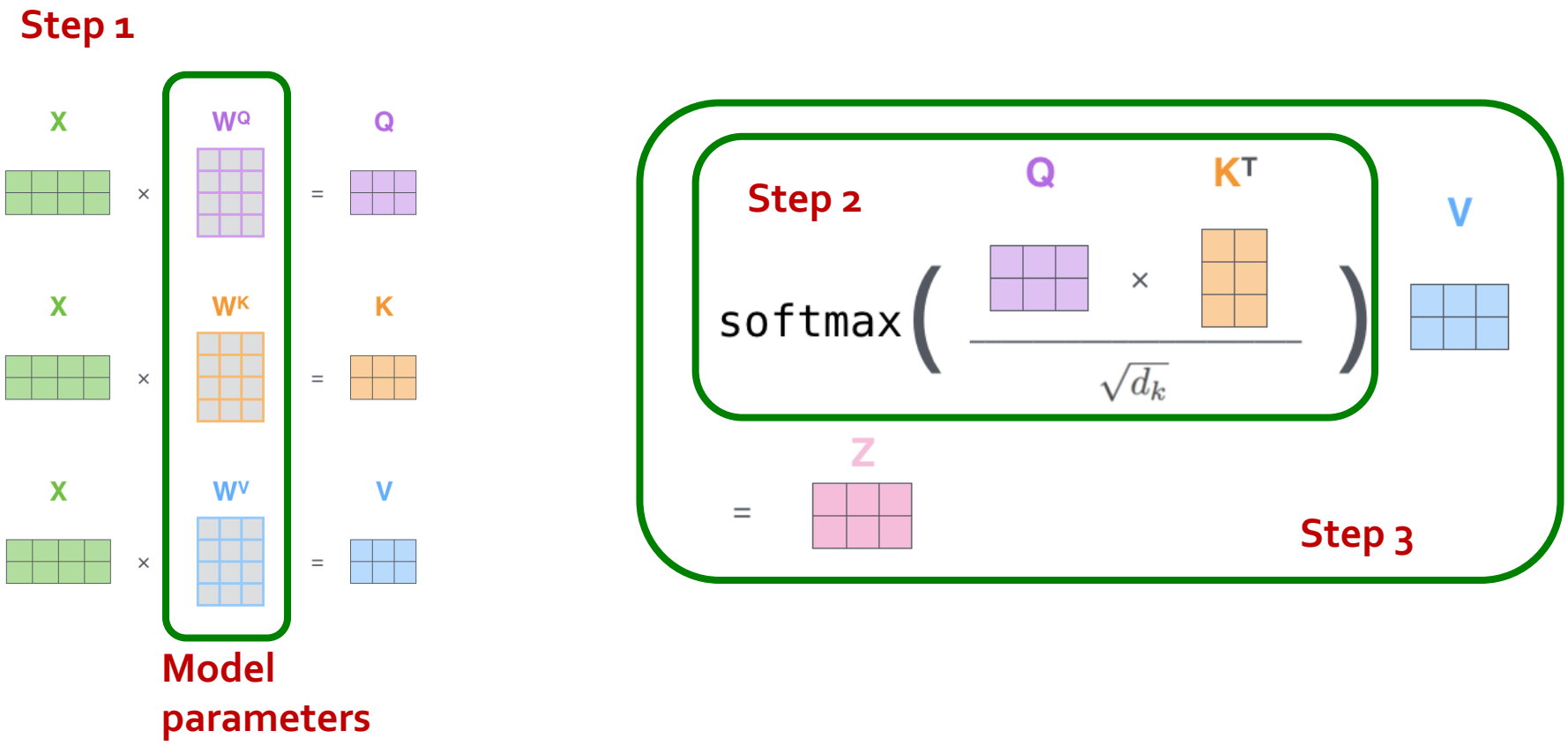
Step 3

$$z_1 = 0.88v_1 + 0.12v_2$$

See: Illustrated Transformer tutorial, https://jalammar.github.io/illustrated-transformer/

Self-attention

- Same calculation in matrix form

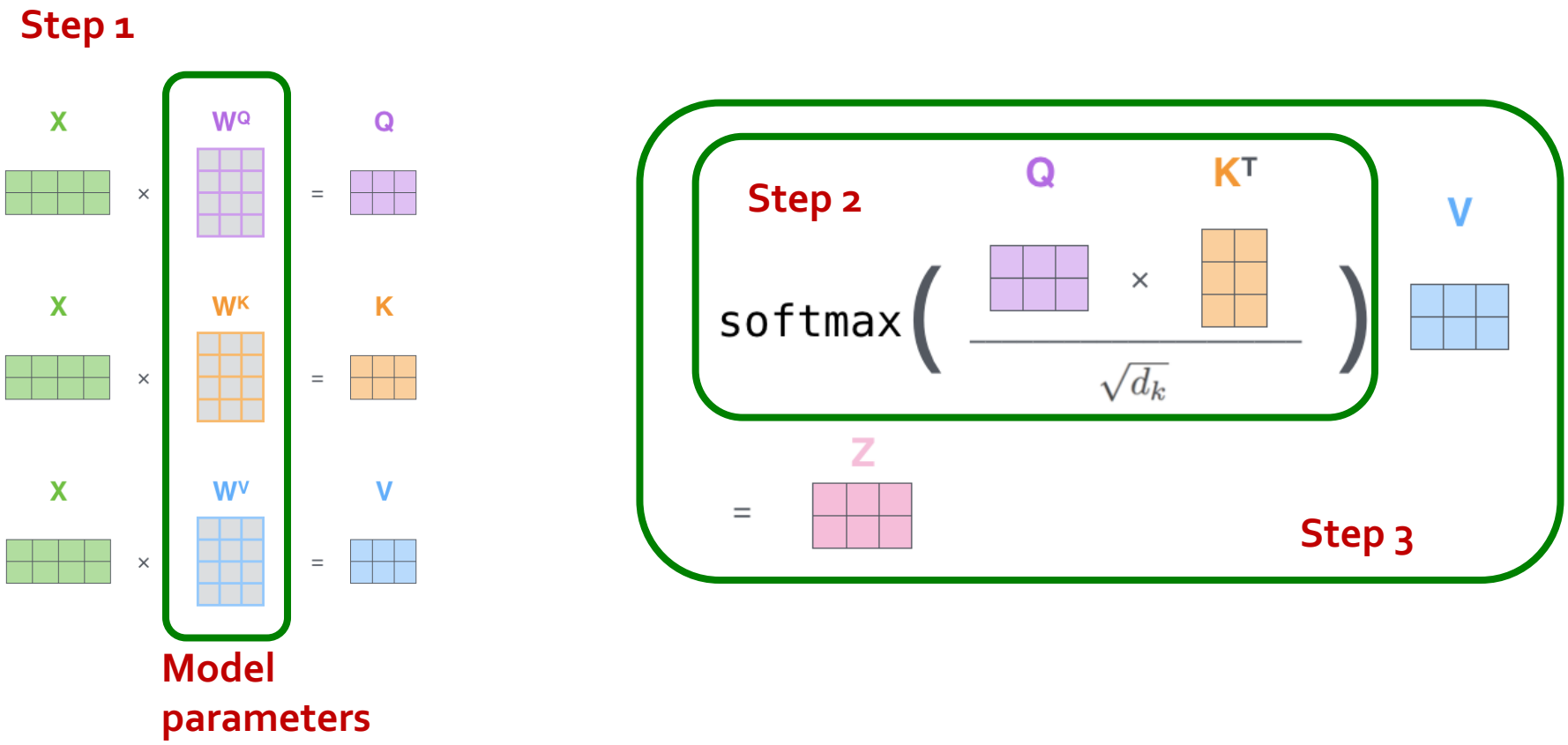


See: Illustrated Transformer tutorial, https://jalammar.github.io/illustrated-transformer/

Self-attention

- Same calculation in matrix form

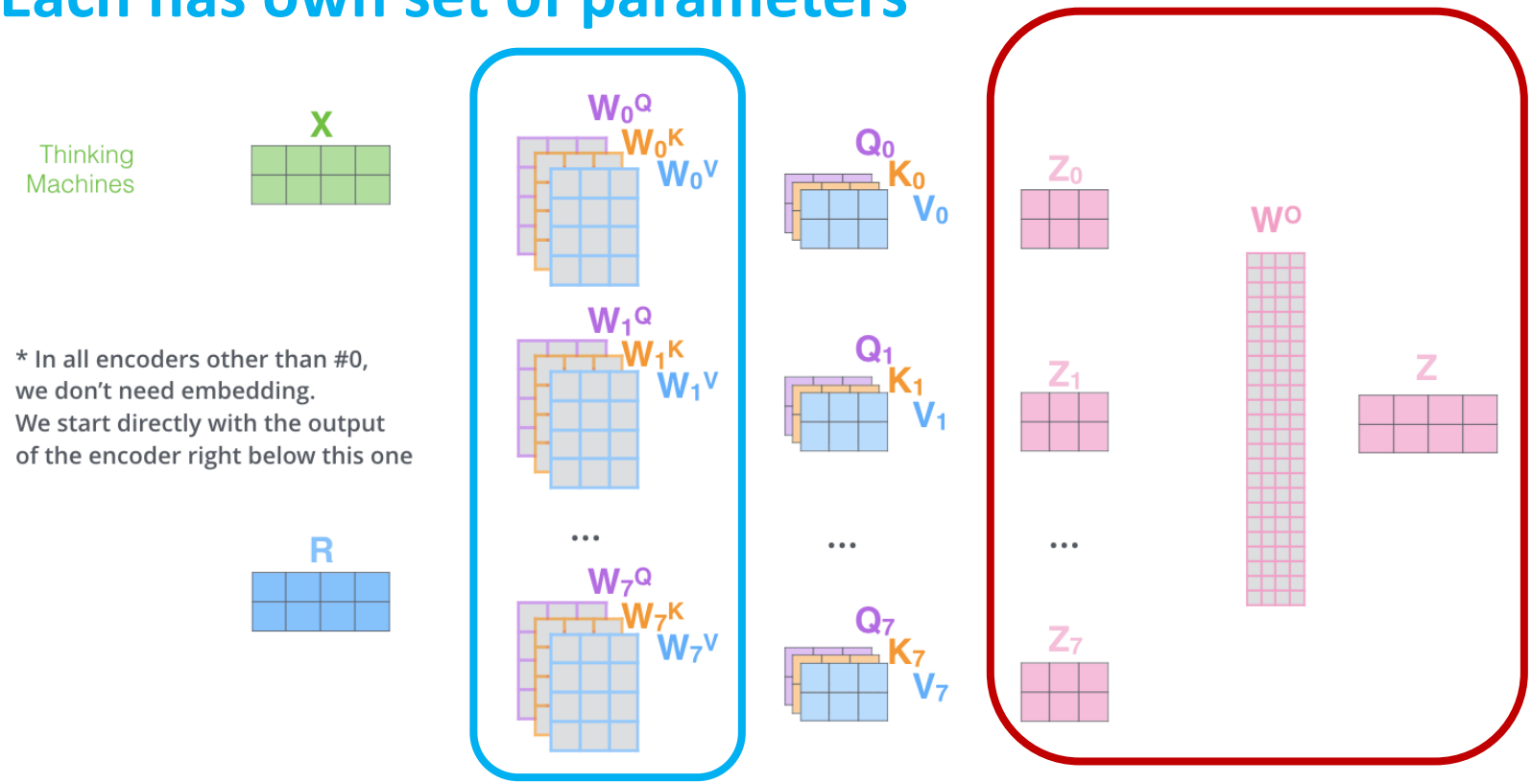
Jure: This slide is duplicated. Probably delete.



See: Illustrated Transformer tutorial, https://jalammar.github.io/illustrated-transformer/

Multi-head self-attention

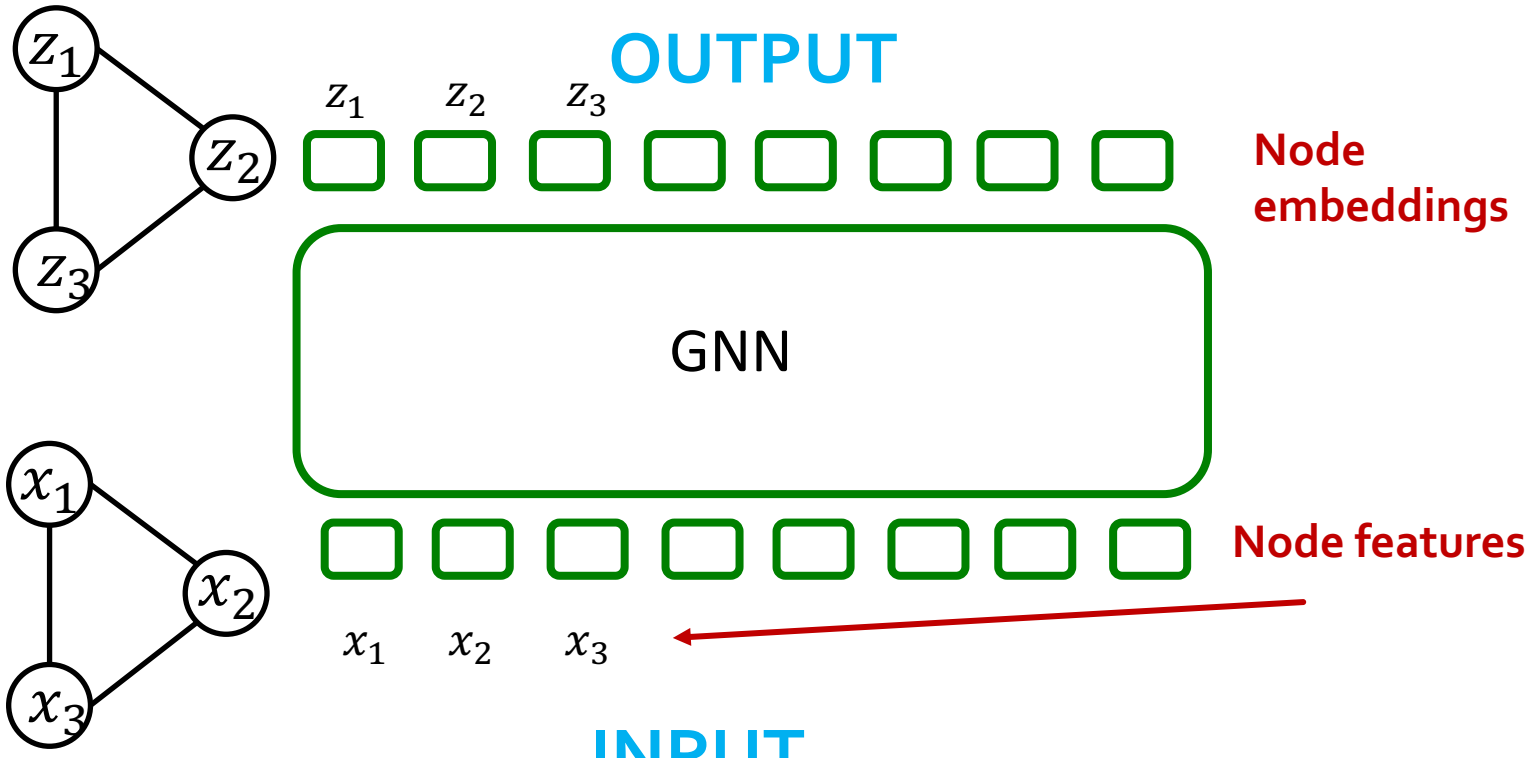
- Do many self-attentions in parallel, and combine
- Different heads can learn different "similarities" between inputs
- Each has own set of parameters



See: Illustrated Transformer tutorial, https://jalammar.github.io/illustrated-transformer/

Comparing Transformers and GNN

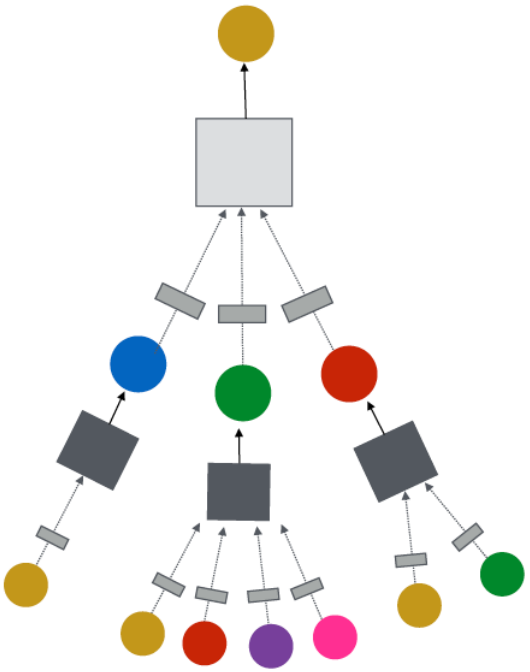
- **Similarity:** GNNs also take in a sequence of vectors (in no particular order) and output a sequence of embeddings
- **Difference:** GNNs use **message passing**, Transformer uses **self-attention**

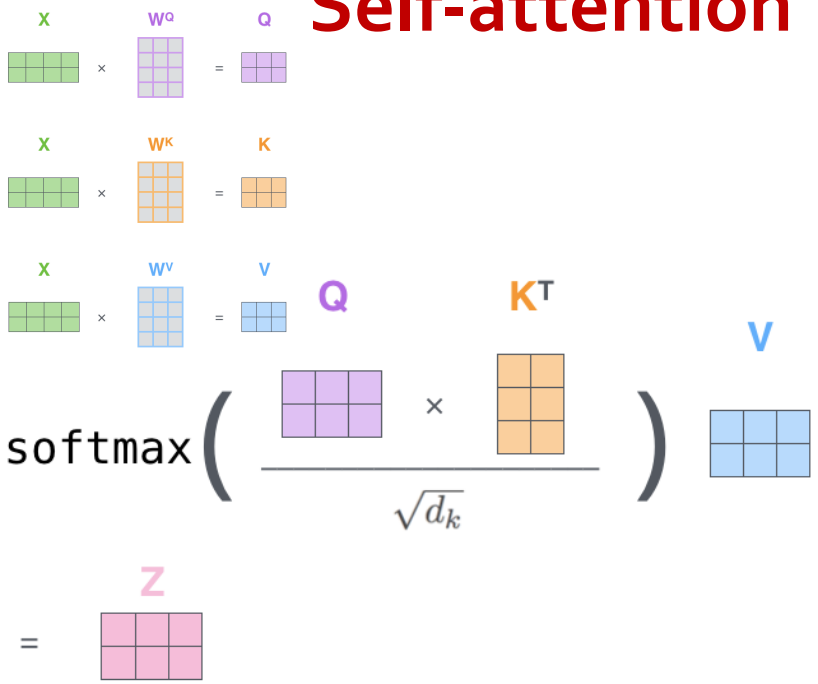


Comparing Transformers and GNN

- **Difference:** GNNs use **message passing**, Transformer uses **self-attention**
- **Are self-attention and message passing really different?**

Message Passing **Vs.** **Self-attention**





Stanford CS224W: Self-attention vs. message passing

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

Interpreting the Self-Attention Update

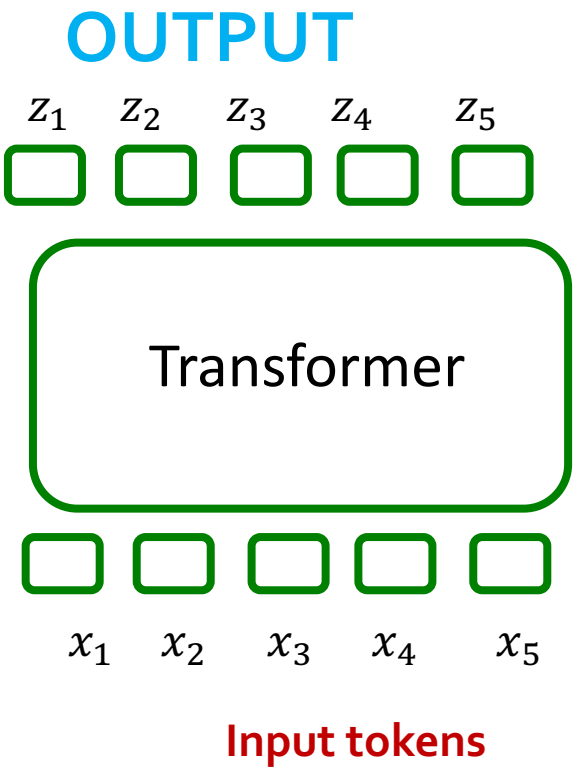
- Recall Formula for attention update:

$$Att(X) = softmax(K^T Q)V$$

$$= softmax\Big((XW^K)^T(XW^Q)\Big)XW^V$$

Inputs stored row-wise

$$X = \begin{bmatrix} — & x_i & — \end{bmatrix}$$



Interpreting the Self-Attention Update

- Recall Formula for attention update:

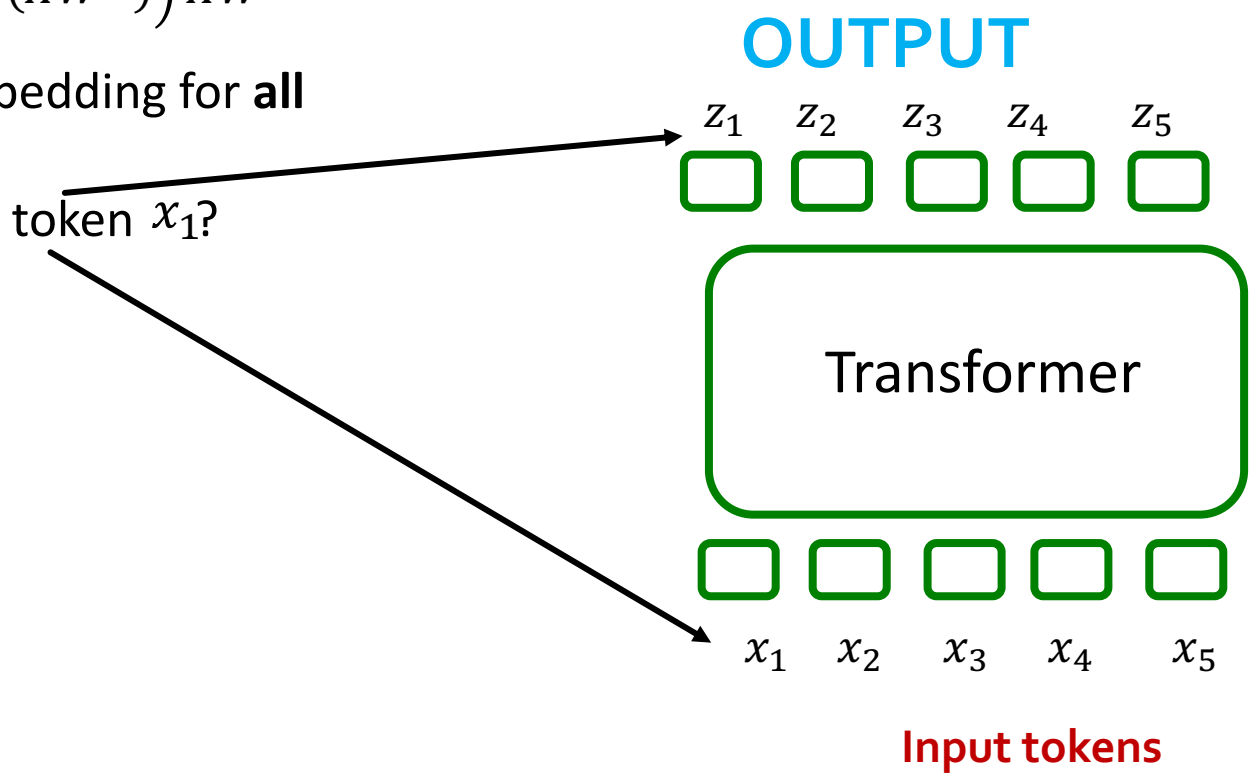
$$Att(X) = softmax(K^T Q)V$$

$$= softmax\left((XW^K)^T(XW^Q)\right)XW^V$$

Inputs stored row-wise

$$X = \begin{bmatrix} — & x_i & — \end{bmatrix}$$

- This formula gives the embedding for **all tokens** simultaneously
- What if we simplify to just token x_1?



Interpreting the Self-Attention Update

- Recall Formula for attention update:

$$Att(X) = softmax(K^T Q)V$$

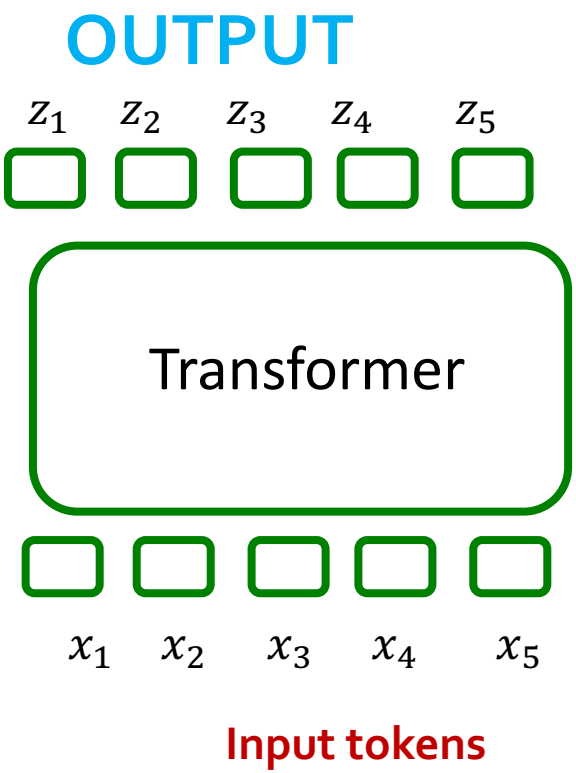
$$= softmax\left((XW^K)^T(XW^Q)\right)XW^V$$

Inputs stored row-wise

$$X = \begin{bmatrix} — & x_i & — \end{bmatrix}$$

- This formula gives the embedding for **all tokens** simultaneously
- What if we simplify to just token x_1?

$$z_1 = \sum_{j=1}^{5} softmax_j(q_1^T k_j)v_j$$ **How to interpret this?**



Interpreting the Self-Attention Update

$$Att(X) = softmax(K^T Q)V$$

$$= softmax\left((XW^K)^T(XW^Q)\right)XW^V$$

Inputs stored row-wise

$$X = \begin{bmatrix} — & x_i & — \end{bmatrix}$$

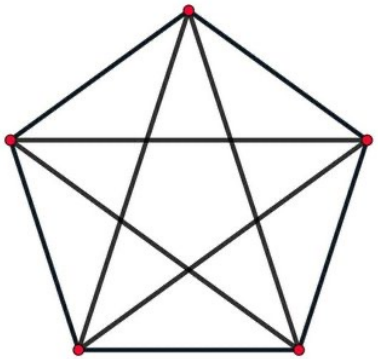
- This formula gives the embedding for **all tokens** simultaneously
- If we simplify to just token x_1 what does the update look like?

$$z_1 = \sum_{j=1}^{5} softmax_j(q_1^T k_j)v_j$$ **How to interpret this?**

- Steps for computing new embedding for token 1:
  - **1. Compute message from j:** $MSG(x_j) = (v_j, \ k_j) = (W^V x_j, W^K x_j)$
  - **2. Compute query for 1:** $q_1 = W^Q x_1$
  - **3. Aggregate all messages:** $Agg(\{MSG(x_j): j\}, q_1) = \sum_{j=1}^{n} softmax_j(q_1^T k_j)v_j$

Self-Attention as Message Passing

- Takeaway: **Self-attention can be written as message + aggregation – i.e., it is a GNN!**
- But so far there is no graph – just tokens.
  - **So what graph is this a GNN on?**
- Clearly tokens = nodes, but what are the edges?
- **Key observation:**
  - Token 1 depends on (receives "messages" from) all other tokens
  - ➔ **the graph is fully connected!**
- **Alternatively: if you only sum over $j \in N(i)$ you get ~GAT**

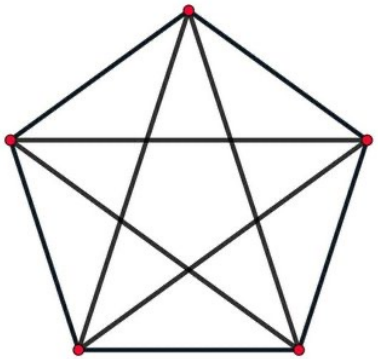


$$z_1 = \sum_{j=1}^{5} softmax_j(q_1^T k_j) v_j$$

- Steps for computing new embedding for token 1:
  - **1. Compute message from j:** $MSG(x_j) = (v_j, \ k_j) = (W^V x_j, W^K x_j)$
  - **2. Compute query for 1:** $q_1 = W^Q x_1$
  - **3. Aggregate all messages:** $Agg(\{MSG(x_j): j\}, q_1) = \sum_{j=1}^{n} softmax_j(q_1^T k_j) v_j$

Self-Attention as Message Passing

- **Takeaway 1:** Self-attention is a special case of message passing



- **Takeaway 2:** It is message passing on the fully connected graph

- **Takeaway 3:** Given a graph G, if you constrain the self-attention softmax to only be over ***j* adjacent to *i*** nodes, you get ~GAT!

- Steps for computing new embedding for token 1:
  - **1. Compute message from j:** $MSG(x_j) = (v_j, \ k_j) = (W^V x_j, W^K x_j)$
  - **2. Compute query for 1:** $q_1 = W^Q x_1$
  - **3. Aggregate all messages:** $Agg(\{MSG(x_j): j\}, q_1) = \sum_{j=1}^{n} softmax_j(q_1^T k_j) v_j$

Plan for Today

- **Part 1:**
  - Introducing Transformers
  - Relation to message passing GNNs
- **Part 2:**
  - A new design landscape for graph Transformers
- **Part 3:**
  - Sign invariant Laplacian positional encodings for graph Transformers

Stanford CS224W: A New Design Landscape for Graph Transformers

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

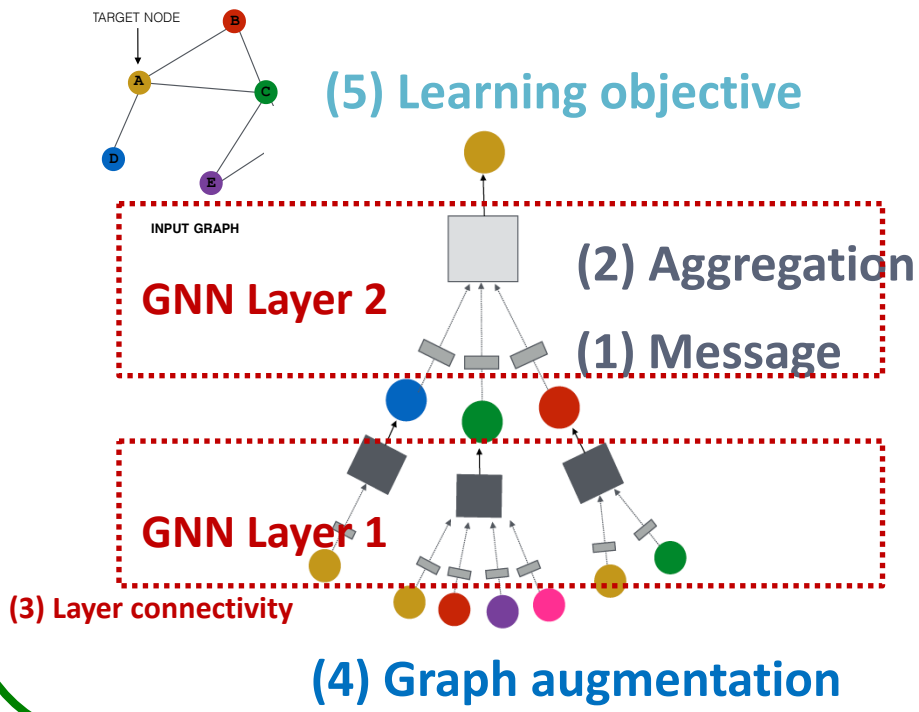
http://cs224w.stanford.edu

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020c

Recap: A General GNN Framework

- We know a lot about the design space of GNNs
- **What does the corresponding design space for Graph Transformers look like?**

GNN design space

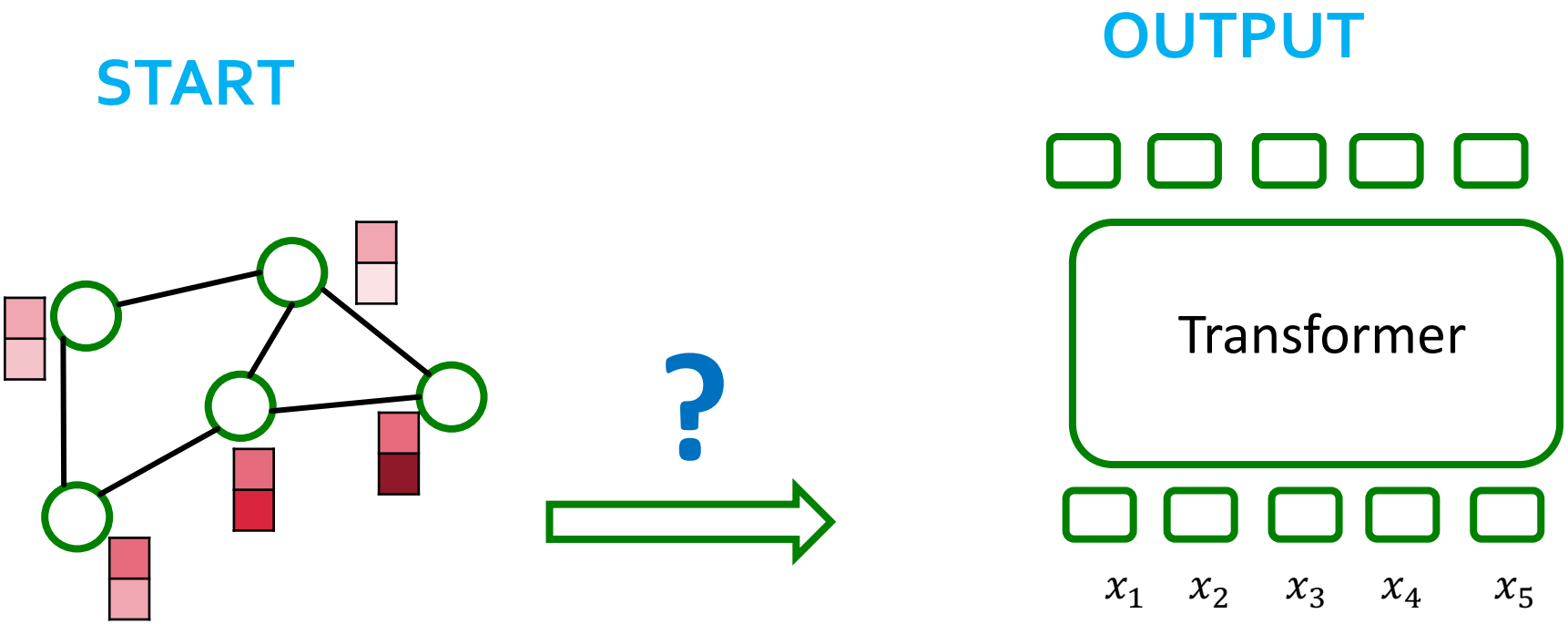


Graph Transformer design space

?

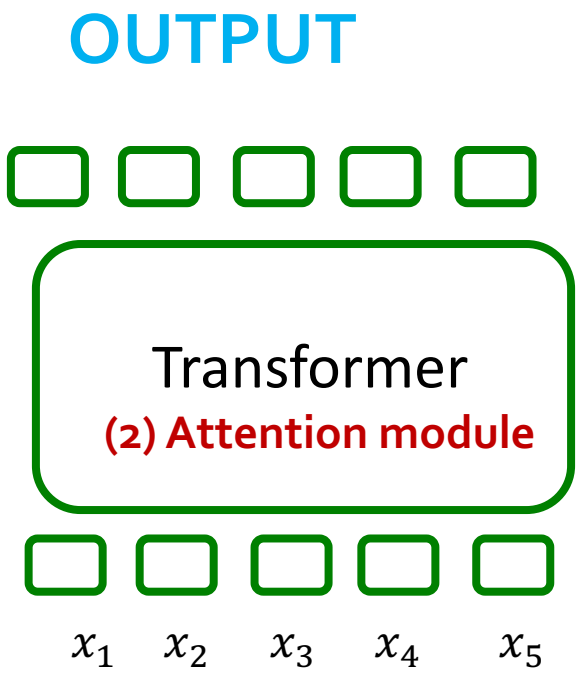
Processing Graphs with Transformers

- We start with graph(s)
- How to input a graph into a Transformer?



Components of a Transformer

- To understand how to process graphs with Transformers we must:
  - Understand the key components of the Transformer. Seen already:
    - 1) tokenizing,
    - 2) self-attention
  - Decide how to make suitable **graph versions** of each



A final key piece: token ordering

- There is one other key missing piece we have not yet discussed…

A final key piece: token ordering

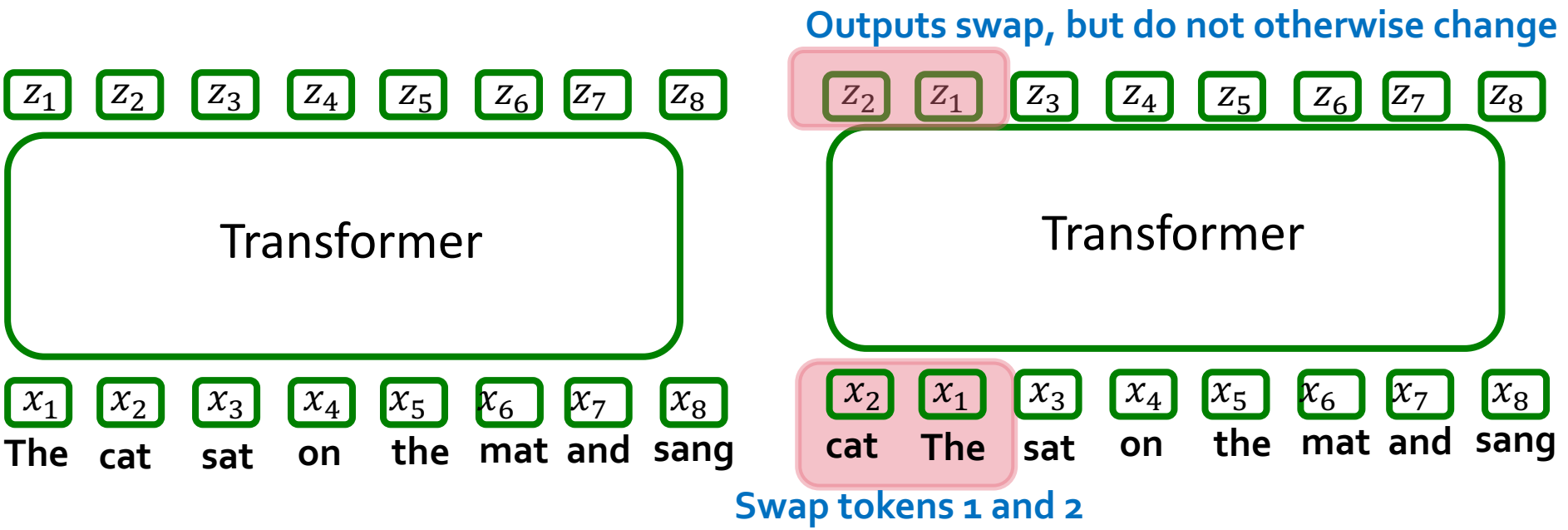
- There is one other key missing piece we have not yet discussed …
- **First recall update formula** $z_1 = \sum_{j=1}^{5} softmax_j(q_1^T k_j) v_j$
- **Key Observation: order of tokens does not matter!!!**

A final key piece: token ordering

- There is one other key missing piece we have not yet discussed …
- **First recall update formula**

$$z_1 = \sum_{j=1}^{5} softmax_j(q_1^T k_j) v_j$$

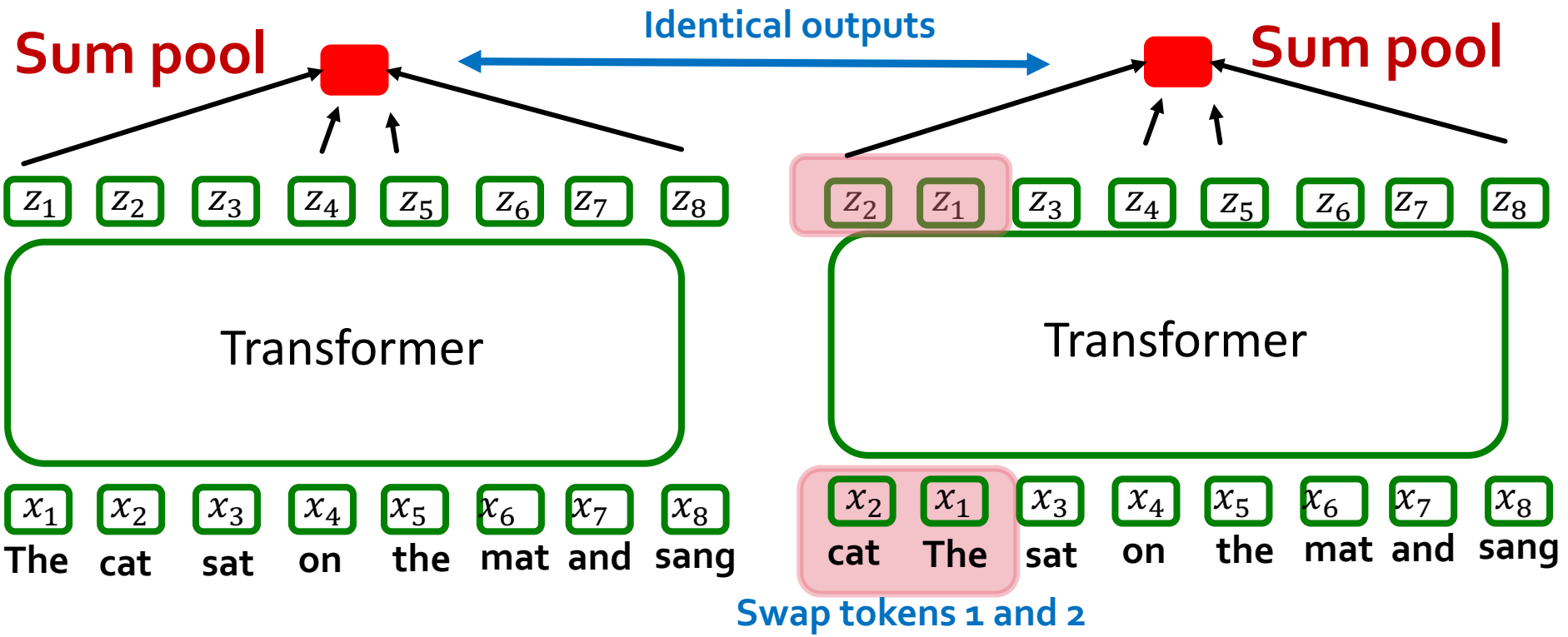
- **Key Observation: order of tokens does not matter!!!**



A final key piece: Token ordering

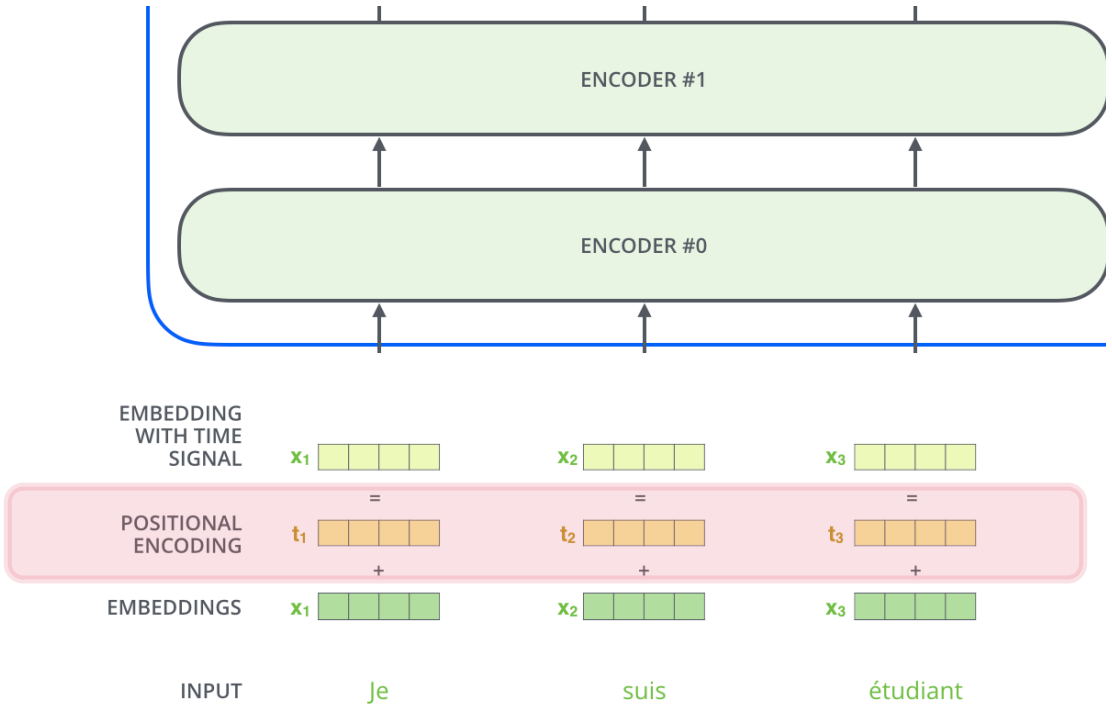
- **This is a problem**
- **Same predictions** no matter what order the words are in!

  (A "bag of words" prediction model)...
  - How to fix?



Positional Encodings

- Transformer doesn't know order of inputs
- Extra **positional** features needed so it knows that
  - Je = word 1,
  - suis = word 2
  - etc.
- For NLP, positional encoding vectors are learnable parameters

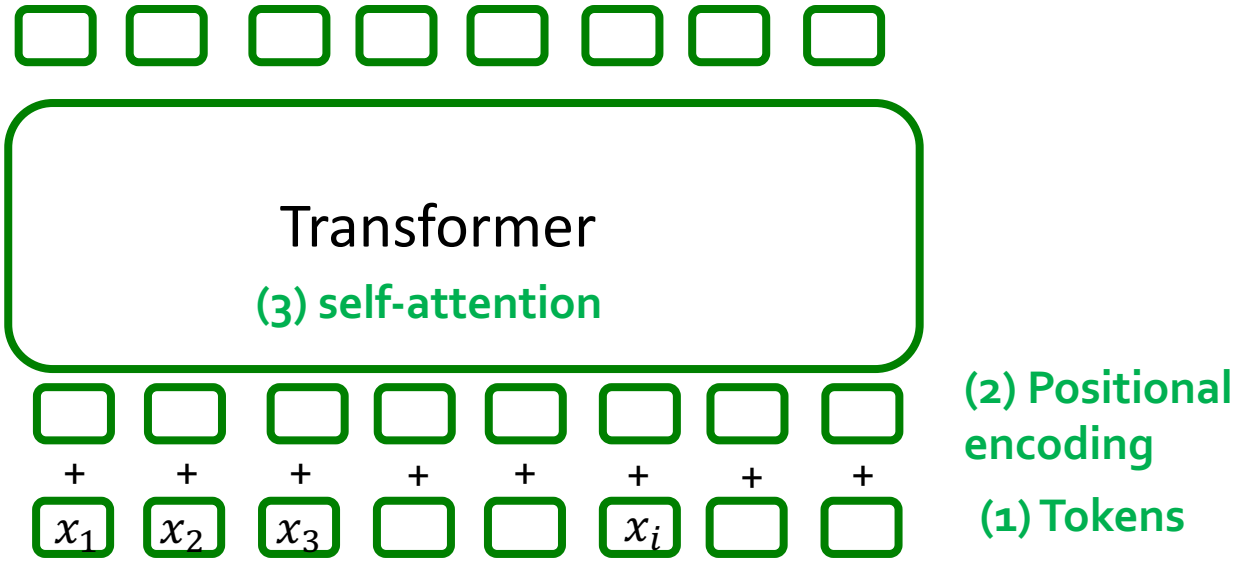


Components of a Transformer

- Key components of Transformer
  - (1) tokenizing
  - (2) positional encoding
  - (3) self-attention

How to chose these for graph data?

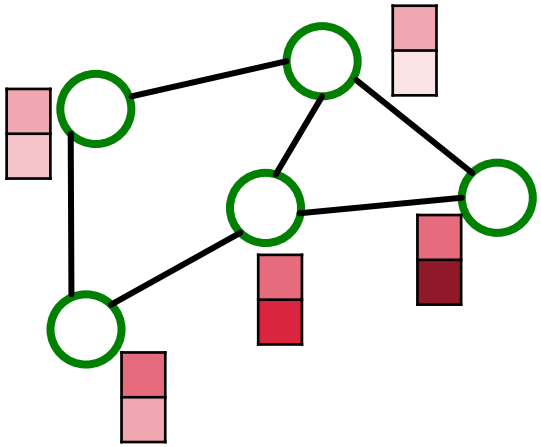
- **Key question:** What should these be for a graph input?



J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020c

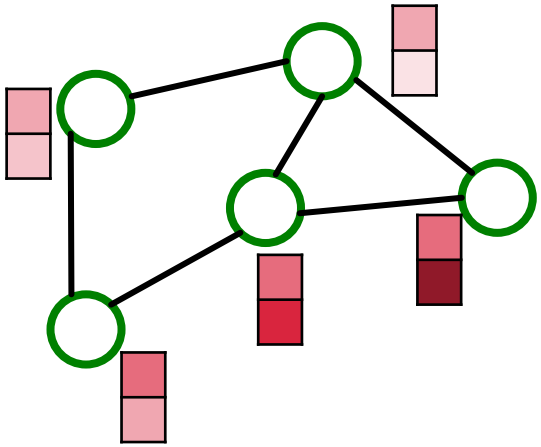
Processing Graphs with Transformers

- A graph Transformer must take the following inputs:
  - **(1) Node features?**
  - **(2) Adjacency information?**
  - **(3) Edge features?**
- Key components of Transformer
  - **(1) tokenizing**
  - **(2) positional encoding**
  - **(3) self-attention**



Processing Graphs with Transformers

- A graph Transformer must take the following inputs:
  - **(1) Node features?**
  - **(2) Adjacency information?**
  - **(3) Edge features?**
- Key components of Transformer
  - **(1) tokenizing**
  - **(2) positional encoding**
  - **(3) self-attention**

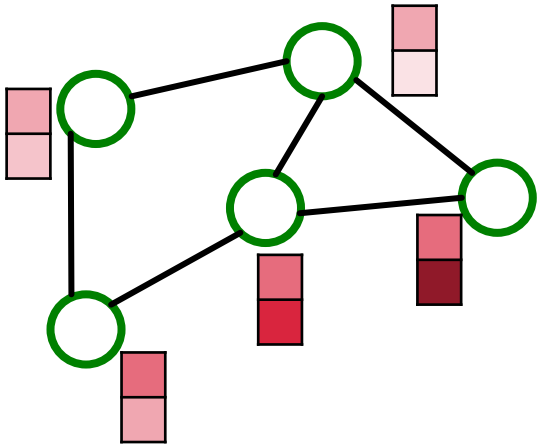


- There are many ways to do this
- Different approaches correspond to different "matchings" between graph inputs **(1), (2), (3)** transformer components **(1), (2), (3)**

Processing Graphs with Transformers

- A graph Transformer must take the following inputs:
  - **(1) Node features?**
  - **(2) Adjacency information?**
  - **(3) Edge features?**
- Key components of Transformer
  - **(1) tokenizing**
  - **(2) positional encoding**
  - **(3) self-attention**

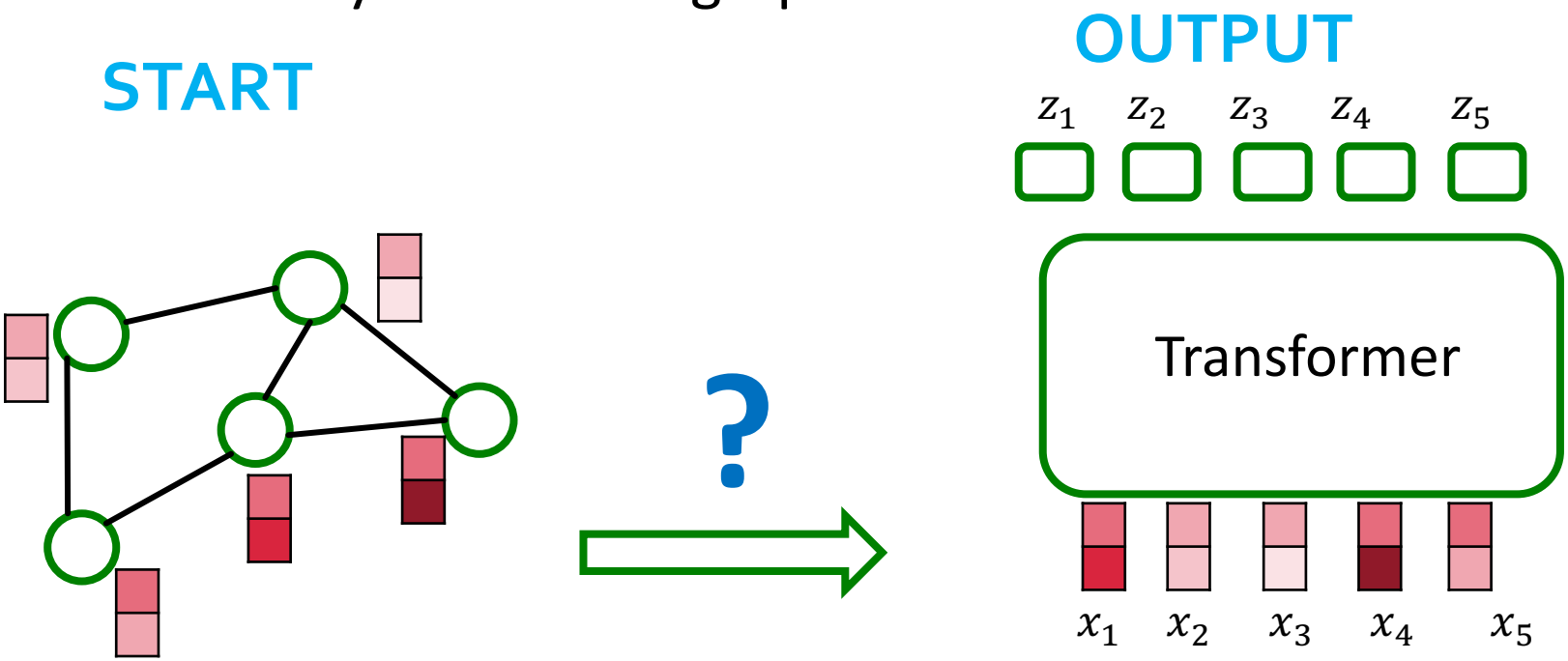
Today



- There are many ways to do this
- Different approaches correspond to different "matchings" between graph inputs **(1), (2), (3)** transformer components **(1), (2), (3)**

Nodes as Tokens

- **Q1: what should our tokens be?**
- **Sensible Idea:** node features = input tokens
- This matches the setting for the "attention is message passing on the fully connected graph" observation

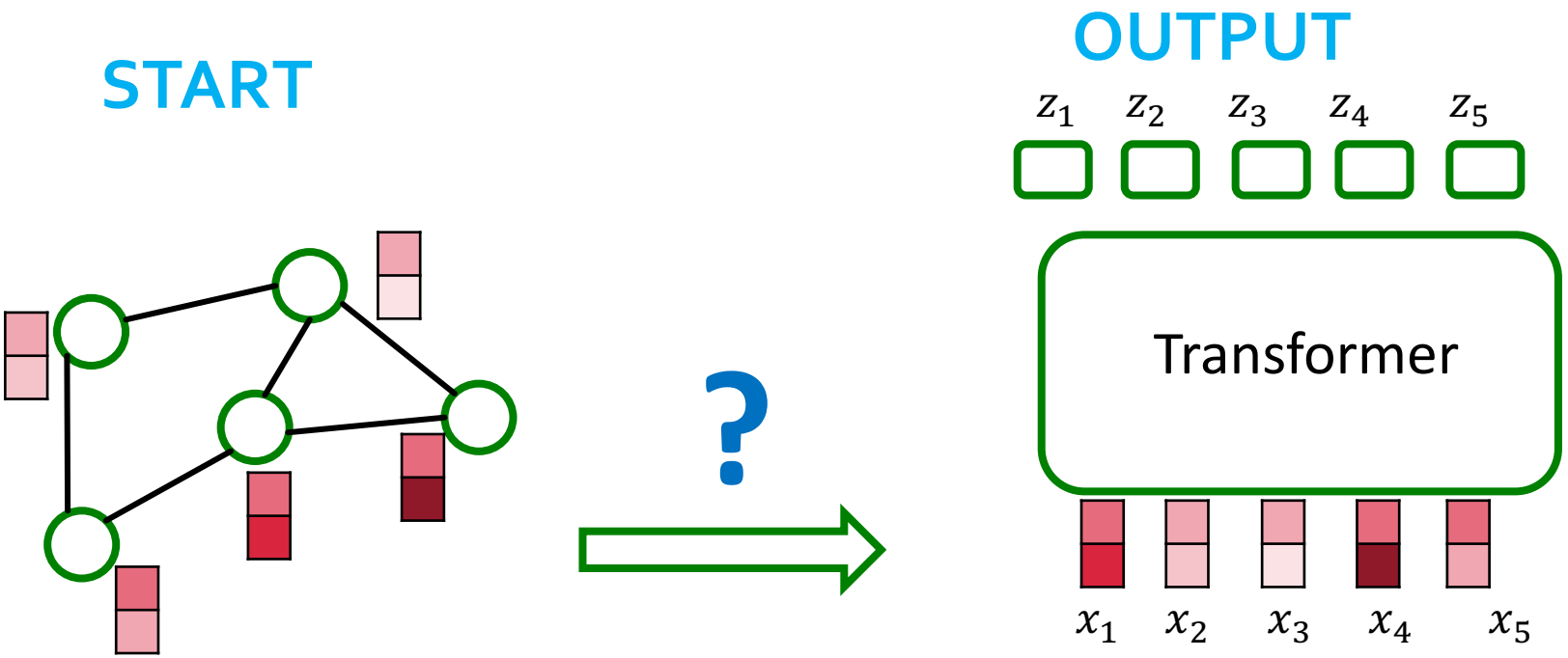


(1) Input tokens = Node features

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020c

Processing Graphs with Transformers

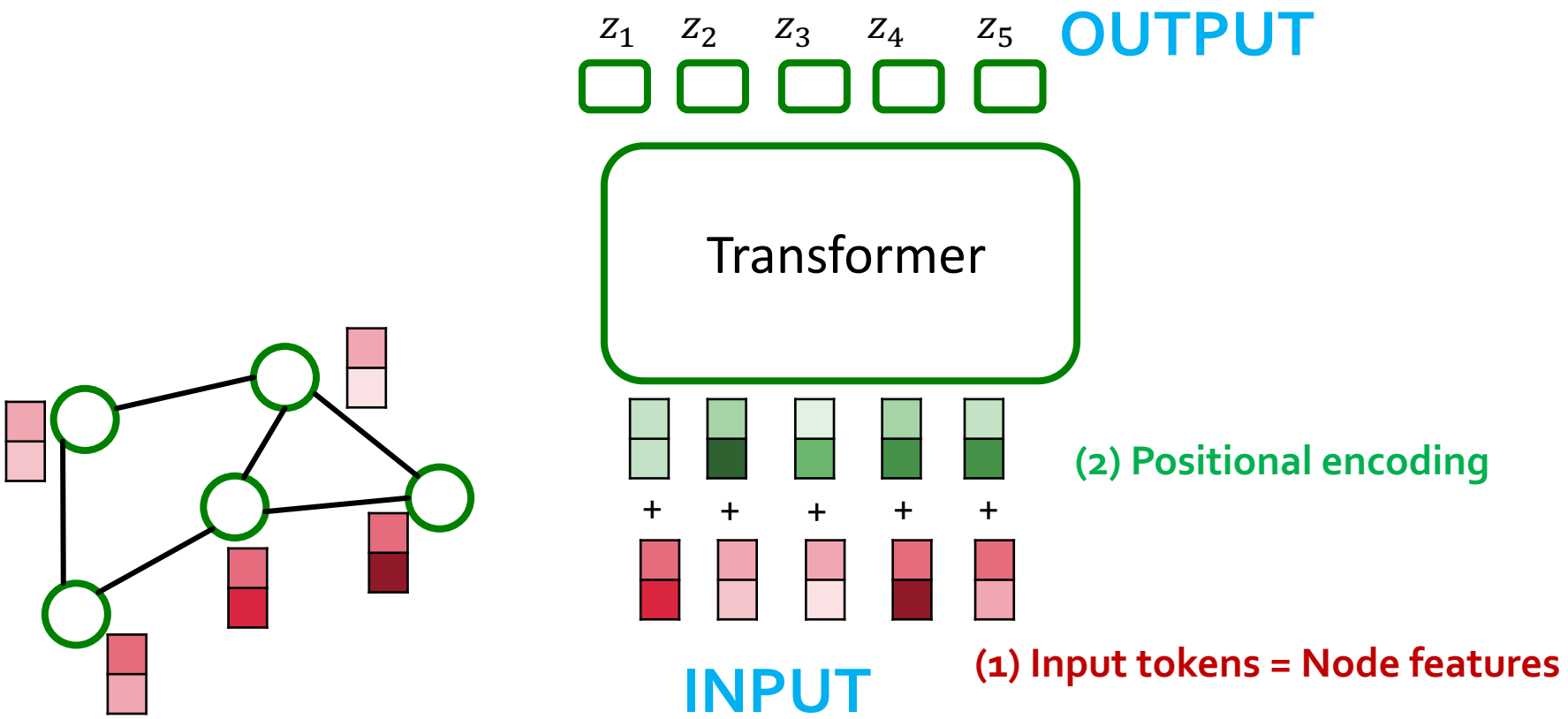
- **Problem? We completely lose adjacency info!**
- **How to also inject adjacency information?**



(1) Input tokens = Node features

How to Add Back Adjacency Info?

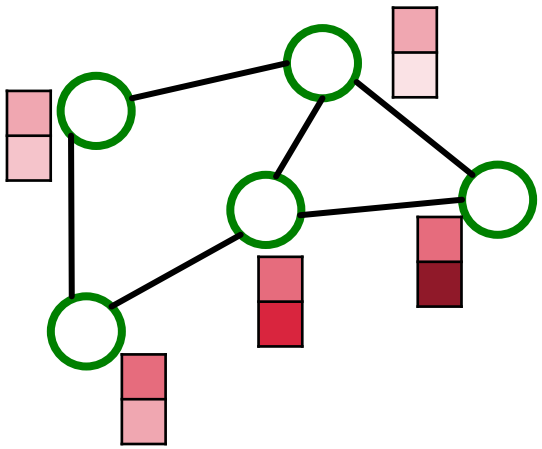
- **Idea:** Encode adjacency info in the **positional encoding** for each node
- Positional encoding describes **where** a node is in the graph

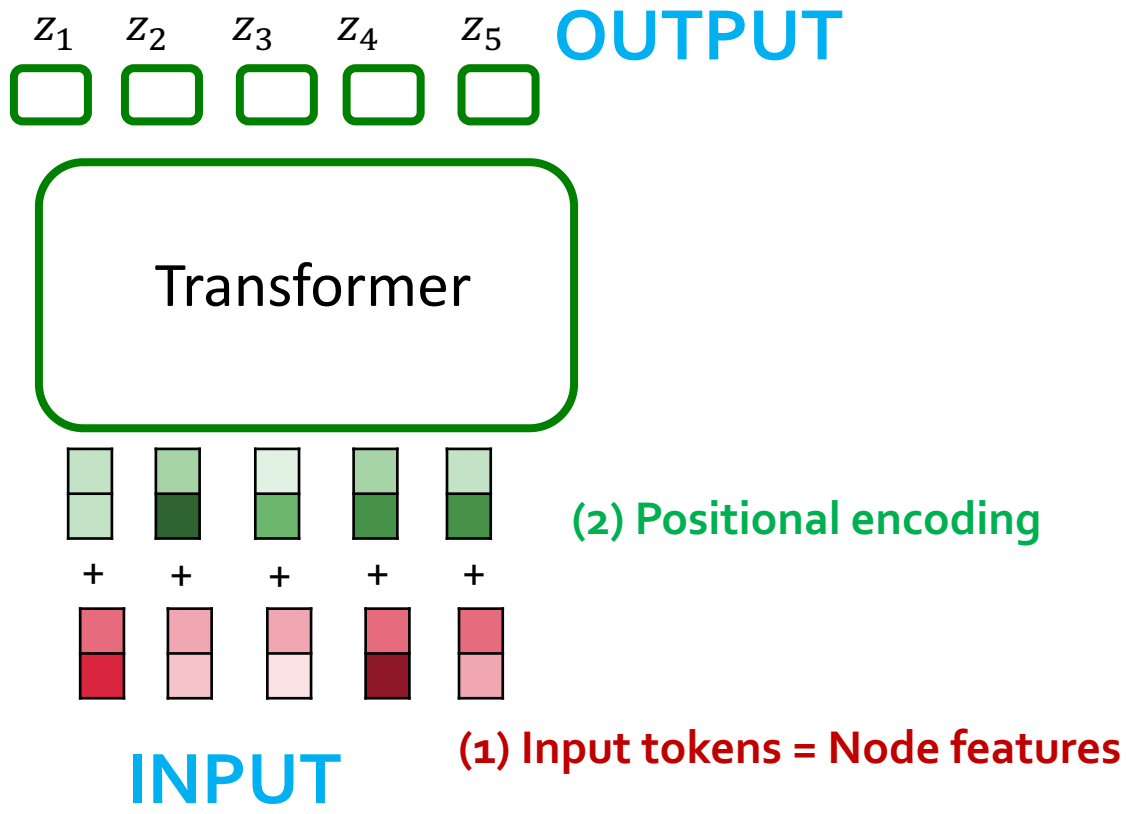


How to Add Back Adjacency Info?

- **Idea:** Encode adjacency info in the **positional encoding** for each node
- Positional encoding describes **where** a node is in the graph

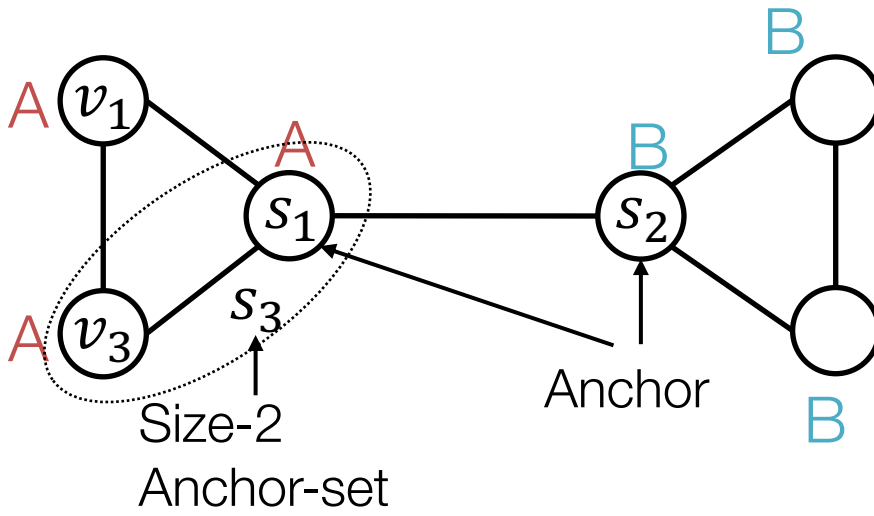
Q2: How to design a good positional encoding?





Option 1: relative distances

- **Last lecture:** positional encoding based on relative distances
- Similar methods based on **random walks**
- This is a good idea! It works well in many cases
- Especially strong for tasks that require counting cycles



Positional encoding for node v_1 =

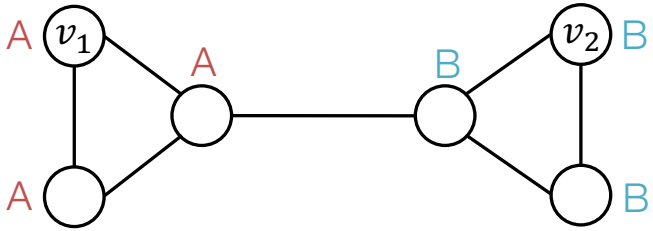
Relative Distances

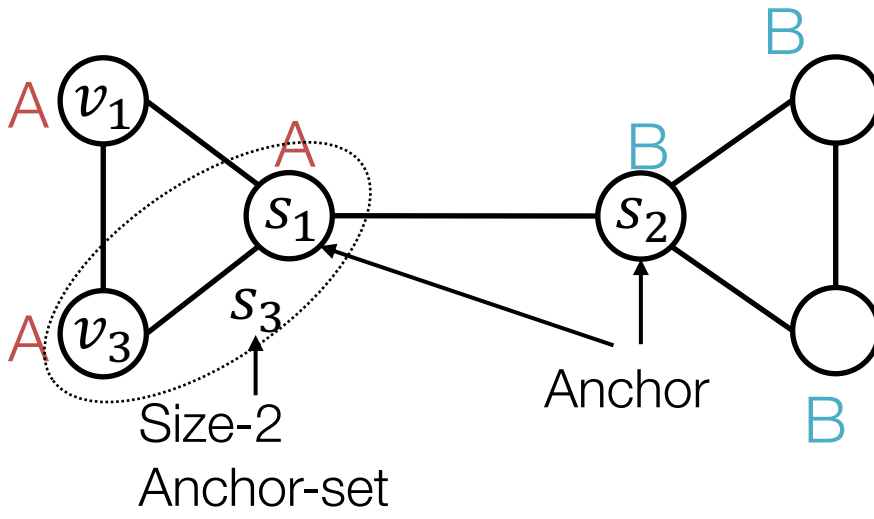
	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

Anchor s_1, s_2 cannot differentiate node v_1, v_3, but anchor-set s_3 can

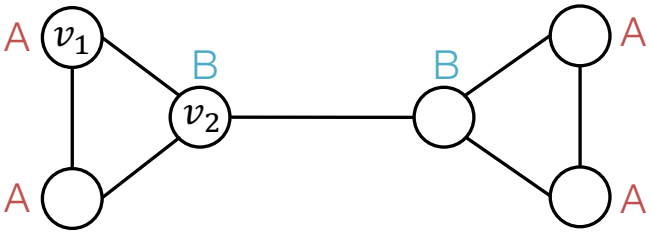
Option 1: Relative distances

- **Last lecture:** Relative distances useful for position-aware task





- But not suited to structure-aware tasks



Positional encoding for node v_1 =

Relative Distances

	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

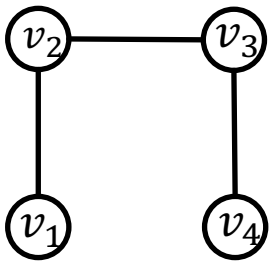
Anchor s_1, s_2 cannot differentiate node v_1, v_3, but anchor-set s_3 can

Option 2: Laplacian Eigenvector Positional Encodings

- What other ways to make positional encoding?

Laplacian Eigenvector Positional Encodings

- **What other ways to make positional encoding?**
- Draw on knowledge of **Graph Theory** (many useful and powerful tools)
- **Key object:** Laplacian Matrix **L = Degrees - Adjacency**
  - Each graph has its own Laplacian matrix
  - Laplacian encodes the graph structure
  - Several Laplacian variants that add degree information differently



$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Degree of each node **Adjacency**

Laplacian Eigenvector Positional Encodings

- Laplacian matrix captures graph structure
- **Its eigenvectors inherit this structure**
- This is important because eigenvectors are vectors (!) and so can be fed into a Transformer
- Eigenvectors with small eigenvalue = local structure, large eigenvalue = global symmetries

Refresher

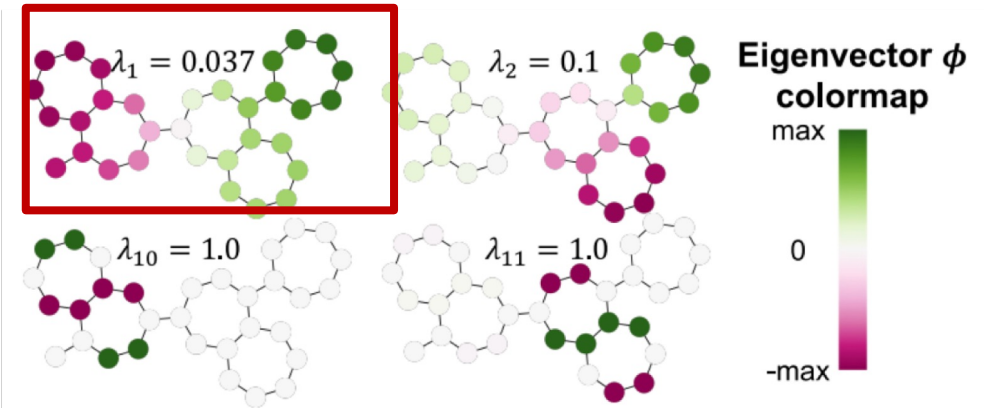
Eigenvector: v such that $Lv = \lambda v$

$L: n \times n$ matrix

$v: n$ dimensional vector

λ: Scalar eigenvalue

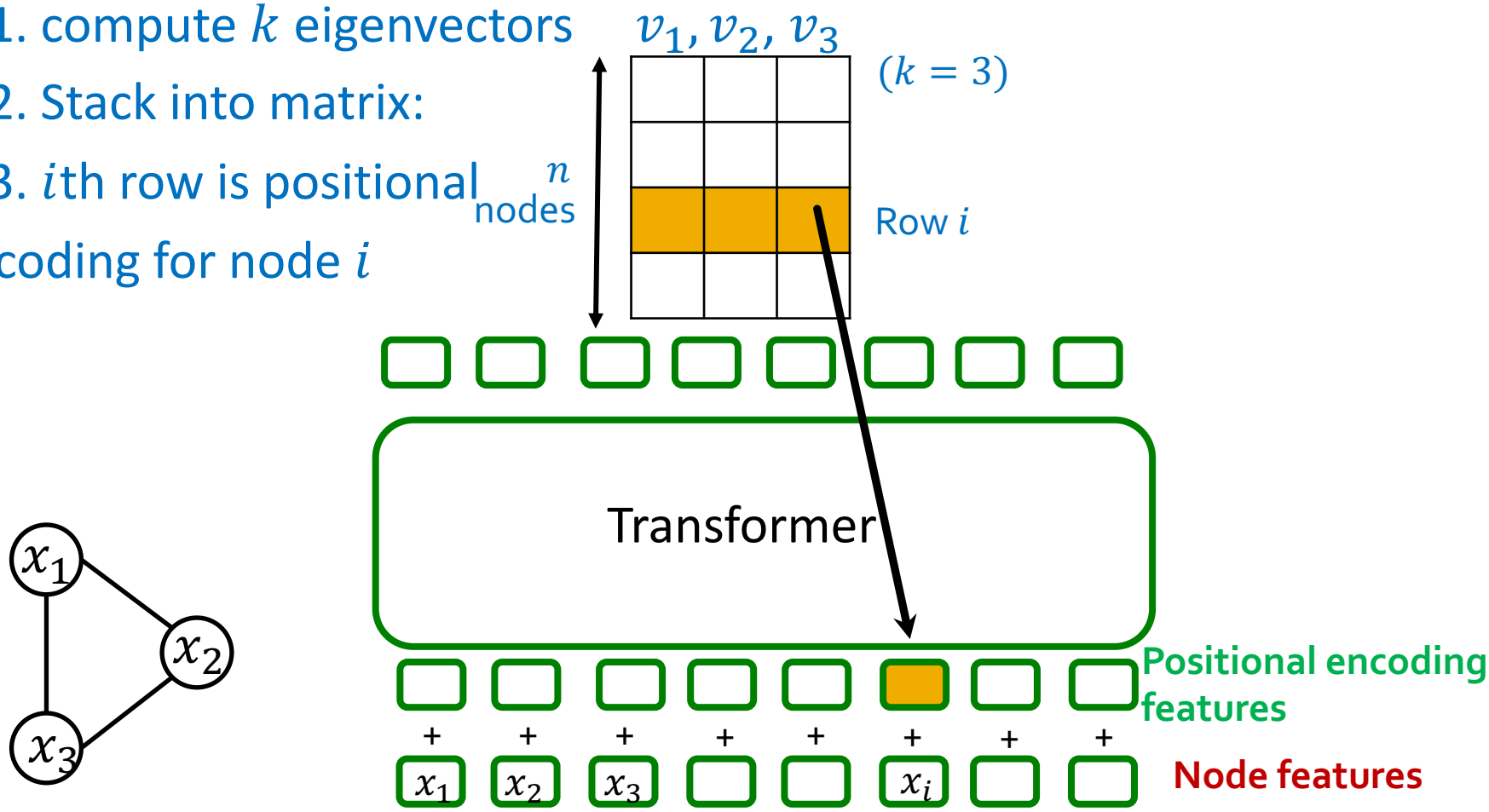
Visualize one eigenvector



(Figure from Kreuzer and Beaini* et al. 2021)*

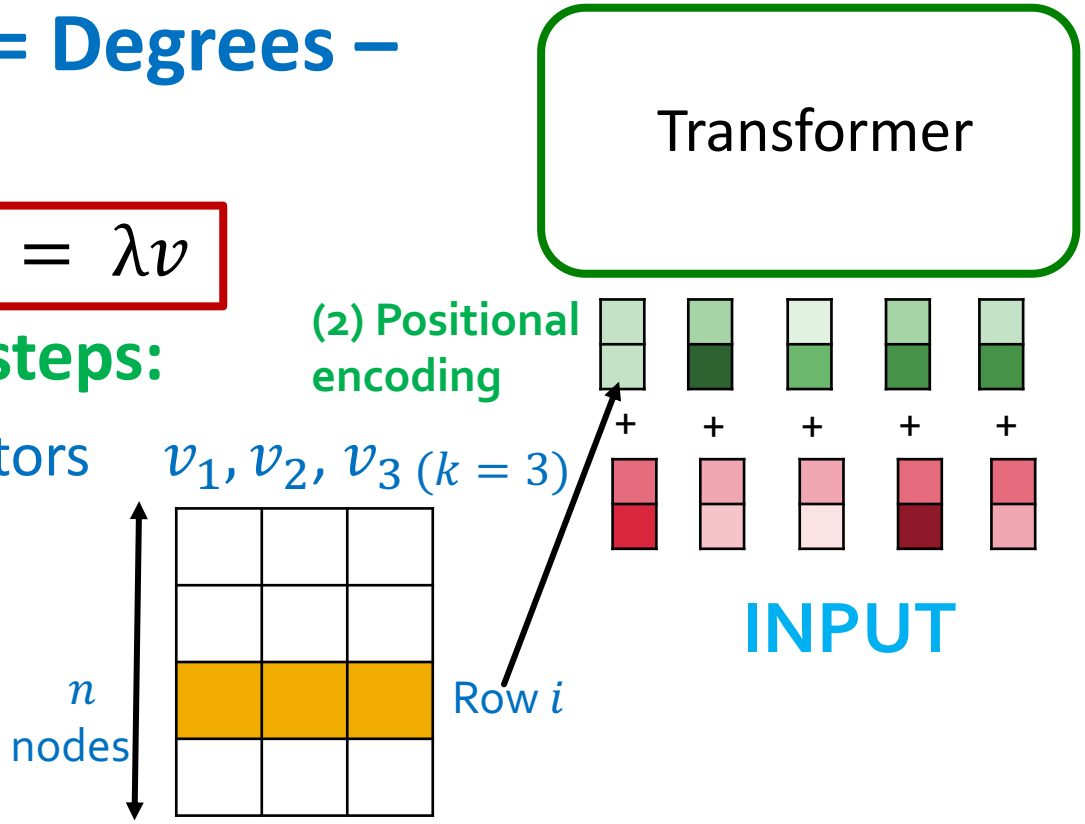
Laplacian Eigenvector Positional Encodings

- **Positional encoding steps:**
  - 1. compute k eigenvectors
  - 2. Stack into matrix:
  - 3. ith row is positional encoding for node i



Summay: Laplacian Eigenvector Positional Encodings

- Laplacian Matrix **L = Degrees – Adjacency**
- Eigenvector: v such that $Lv = \lambda v$
- **Positional encoding steps:**
  - 1. compute k eigenvectors
  - 2. Stack into matrix:
  - 3. ith row is positional encoding for node i



- Laplacian Eigenvector positional encodings can also be used with message-passing GNNs
  - **This helps for same reasons as relative-distance based positional encodings in previous lecture**

Laplacian Eigenvectors in Practice

- Task: given a graph, predict YES if it has a cycle, NO otherwise
- Recall, message-passing cannot solve this task!

- "PE" indicates using Laplacian Eigenvector Pos. Enc.

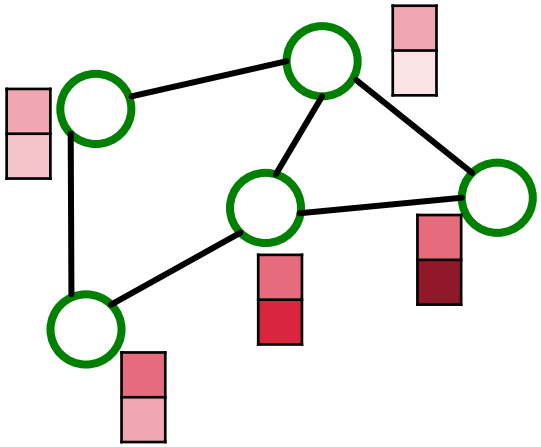
Train samples →			200	500	1000	5000
Model	L	**#Param**	**Test Acc±s.d.**			
GIN	4	100774	70.585±0.636	74.995±1.226	78.083±1.083	86.130±1.140
GIN-PE	4	102864	**86.720±3.376**	**95.960±0.393**	**97.998±0.300**	**99.570±0.089**
GatedGCN	4	103933	50.000±0.000	50.000±0.000	50.000±0.000	50.000±0.000
GatedGCN-PE	4	105263	**95.082±0.346**	**96.700±0.381**	**98.230±0.473**	**99.725±0.027**

Processing Graphs with Transformers

- A graph Transformer must take the following inputs:
  - (1) Node features?
  - (2) Adjacency information?
  - (3) Edge features?

- Key components of Transformer
  - (1) tokenizing
  - (2) positional encoding
  - (3) self-attention

So far

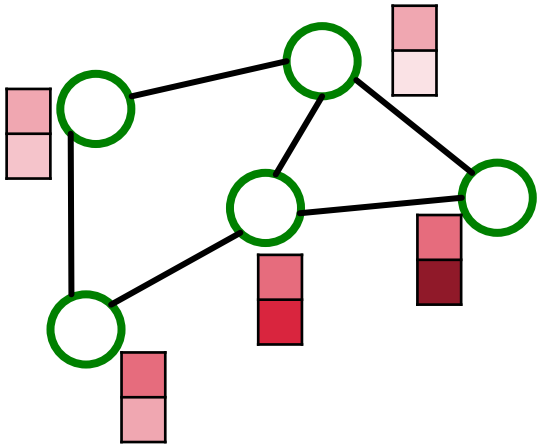


- There are many ways to do this
- Different approaches correspond to different "matchings" between graph inputs **(1), (2), (3)** transformer components **(1), (2), (3)**

Processing Graphs with Transformers

- A graph Transformer must take the following inputs:
  - (1) Node features?
  - (2) Adjacency information?
  - (3) Edge features?
- Key components of Transformer
  - (1) tokenizing
  - (2) positional encoding
  - (3) self-attention

Left to do



- There are many ways to do this
- Different approaches correspond to different "matchings" between graph inputs (1), (2), (3) transformer components (1), (2), (3)

Edge Features in Self-Attention

- Not clear how to add edge features in the tokens or positional encoding
- How about in the attention? $Att(X) = softmax(K^T Q)V$
- $[k_{ij}] = K^T Q$ is an n x n matrix. Entry k_{ij} describes "how much" token j contributes to the update of token i

Do Transformers Really Perform Bad for Graph Representation? Ying et al. NeurIPS 2021

Edge Features in Self-Attention

- Not clear how to add edge features in the tokens or positional encoding
- How about in the attention? $Att(X) = softmax(K^T Q)V$
- $[k_{ij}] = K^T Q$ is an n x n matrix. Entry k_{ij} describes "how much" token j contributes to the update of token i
- **Idea: adjust k_{ij} based on edge features. Replace with $k_{ij} + c_{ij}$ where c_{ij} depends on the edge features**

Do Transformers Really Perform Bad for Graph Representation? Ying et al. NeurIPS 2021

Edge Features in Self-Attention

- Not clear how to add edge features in the tokens or positional encoding
- How about in the attention? $Att(X) = softmax(K^T Q)V$
- $[k_{ij}] = K^T Q$ is an n x n matrix. Entry k_{ij} describes "how much" token j contributes to the update of token i
- **Idea: adjust k_{ij} based on edge features. Replace with $k_{ij} + c_{ij}$ where c_{ij} depends on the edge features**
- **Implementation:**

  Learned parameters w_1

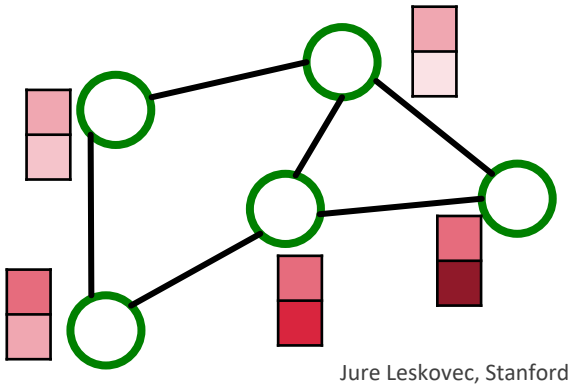
  - **If there is an edge between i and j with features e_{ij}, define $c_{ij} = w_1^T e_{ij}$**
  - **If there is no edge, find shortest edge path between i and j $(e^1, e^2, \ldots e^N)$ and define $c_{ij} = \sum_n w_n^T e^n$**

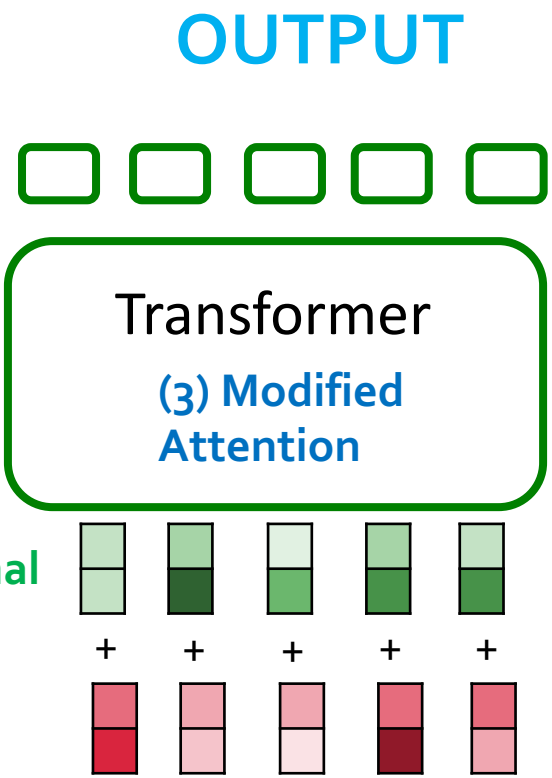
  Learned parameters $w_1, \ldots, w_N$

Do Transformers Really Perform Bad for Graph Representation? Ying et al. NeurIPS 2021

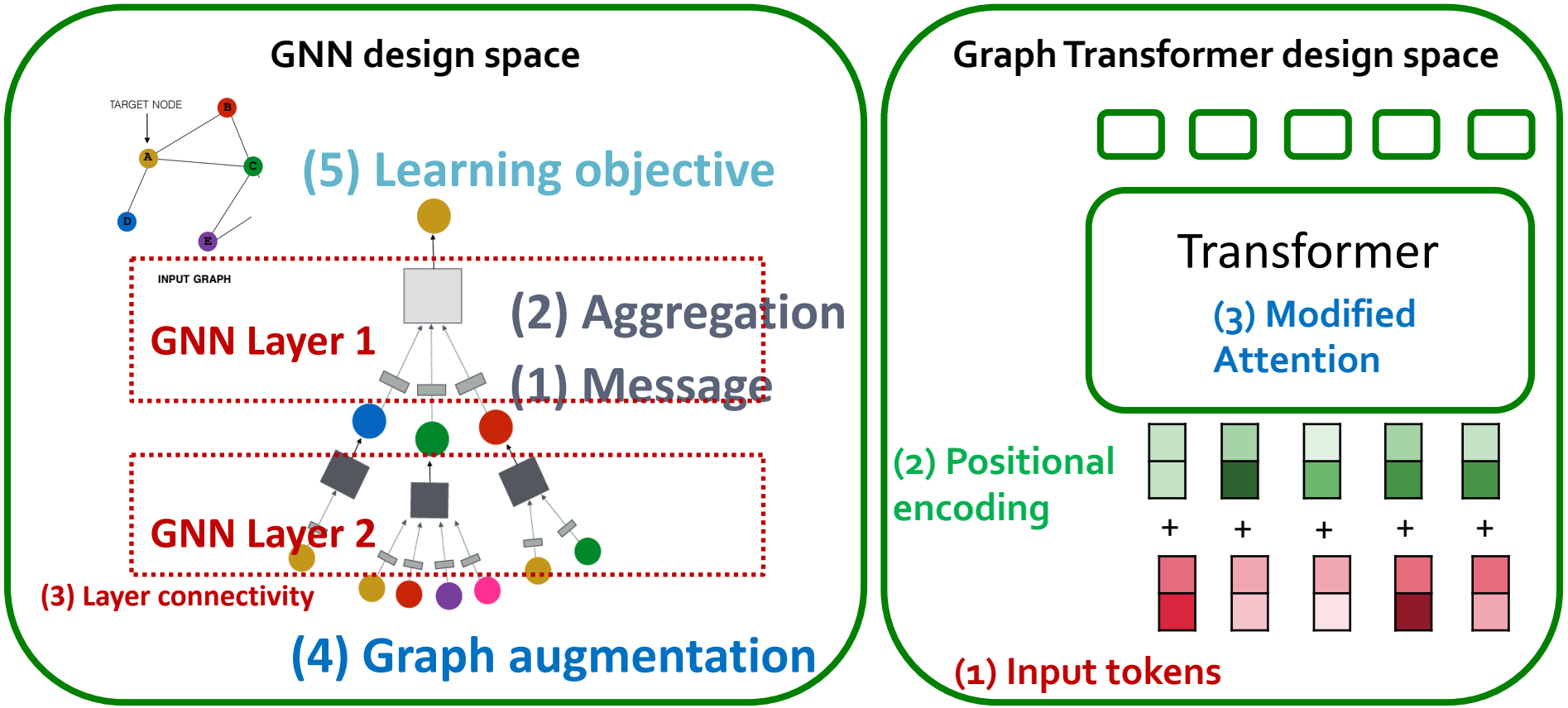
Summary: Graph Transformer Design Space

- **(1) Tokenization**
  - Usually node features
  - Other options, such as subgraphs, and node + edge features (not discussed today)
- **(2) Positional Encoding**
  - Relative distances, or Laplacian eigenvectors
  - Gives Transformer adjacency structure of graph
- **(3) Modified Attention**
  - Reweight attention using edge features





Summary: Graph Transformer Design Space



Plan for Today

- **Part 1:**
  - Introducing Transformers
  - Relation to message passing GNNs
- **Part 2:**
  - A new design landscape for graph Transformers
- **Part 3:**
  - Sign invariant Laplacian positional encodings for graph Transformers

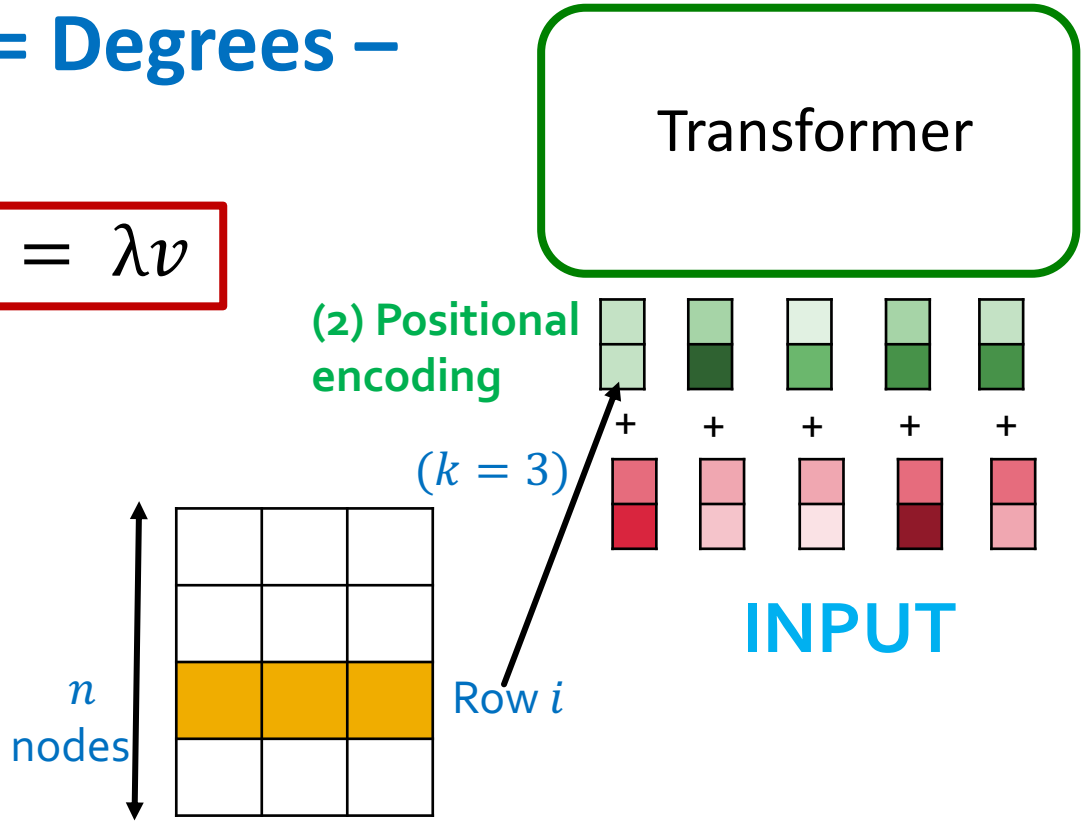
Jure:
The lecture is great! I like how nicely you explain the concepts and connect them!
My sense is that you have a lot of material. If you cover the material till here (finish the lecture here) it will be plenty.

Stanford [illegible]ful Positional Encodings for Graph Transformers

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

Recall: Laplacian Eigenvector Positional Encodings

- Laplacian Matrix **L = Degrees – Adjacency**
- Eigenvector: v such that $Lv = \lambda v$



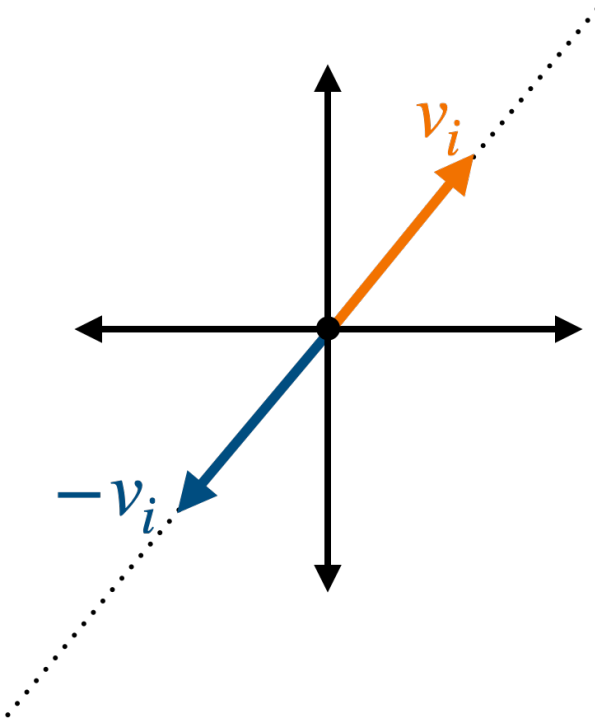
Laplacian Eigenvector Positional Encodings

- Laplacian Eigenvector positional encodings work!
- **But is this the best we can do?**
  - **Hint: no**
- Q: What is the problem with the current approach?
  - A1: Eigenvectors are **not** arbitrary vectors
  - A2: They have **special structure** that we have been ignoring!
- **To use eigenvectors properly we must account for their structure in our models**

Eigenvector Sign Ambiguity

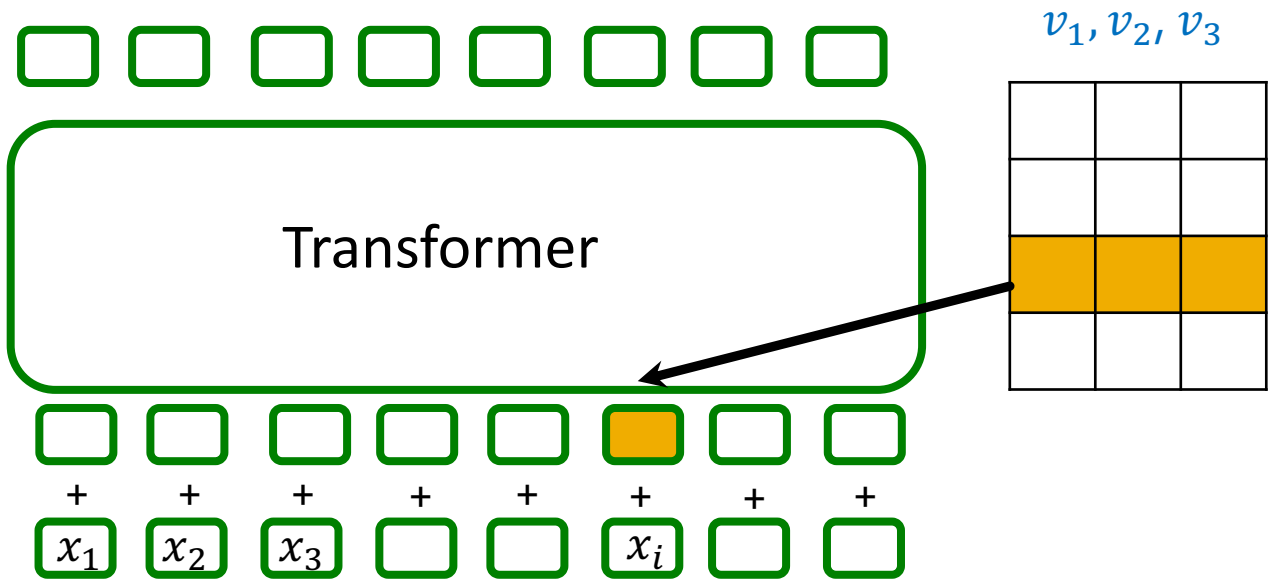
- Suppose v is a Laplacian eigenvector
  - So $Lv = \lambda v$
- But this means:
  - Also $L(-v) = \lambda(-v)$
- So $-v$ is also a Laplacian eigenvector

The choice of sign is arbitrary!



Sign Ambiguity is a Problem

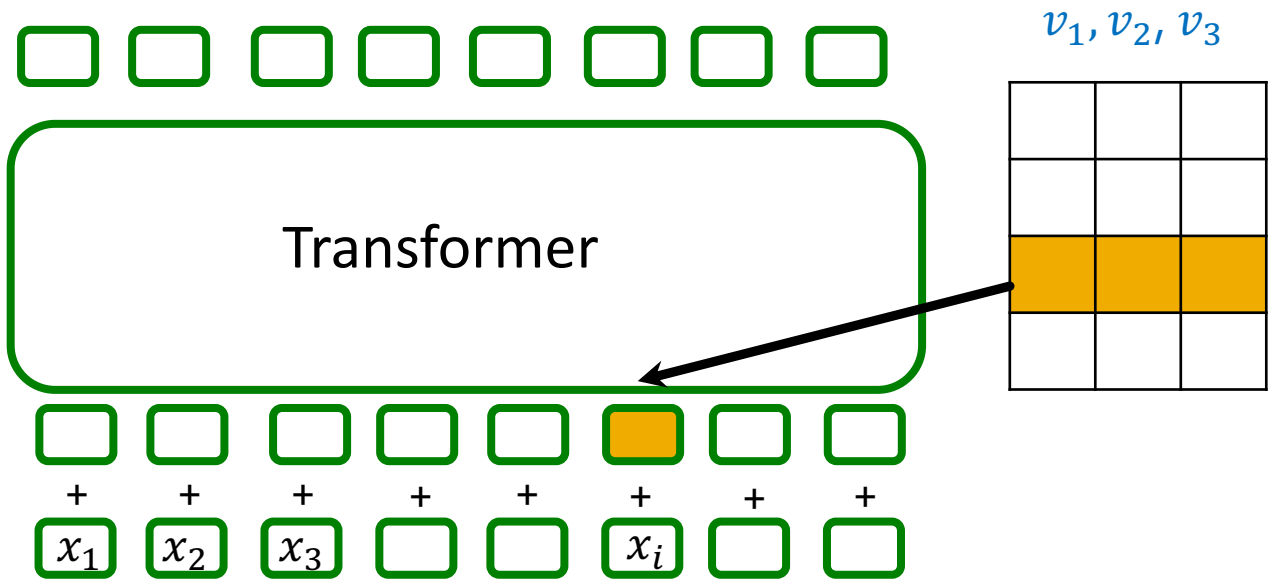
- Both v and $-v$ are eigenvectors
- But when we use them as positional encodings we **pick one arbitrarily**
- **Why does this matter for positional encodings?**



Sign Ambiguity is a Problem

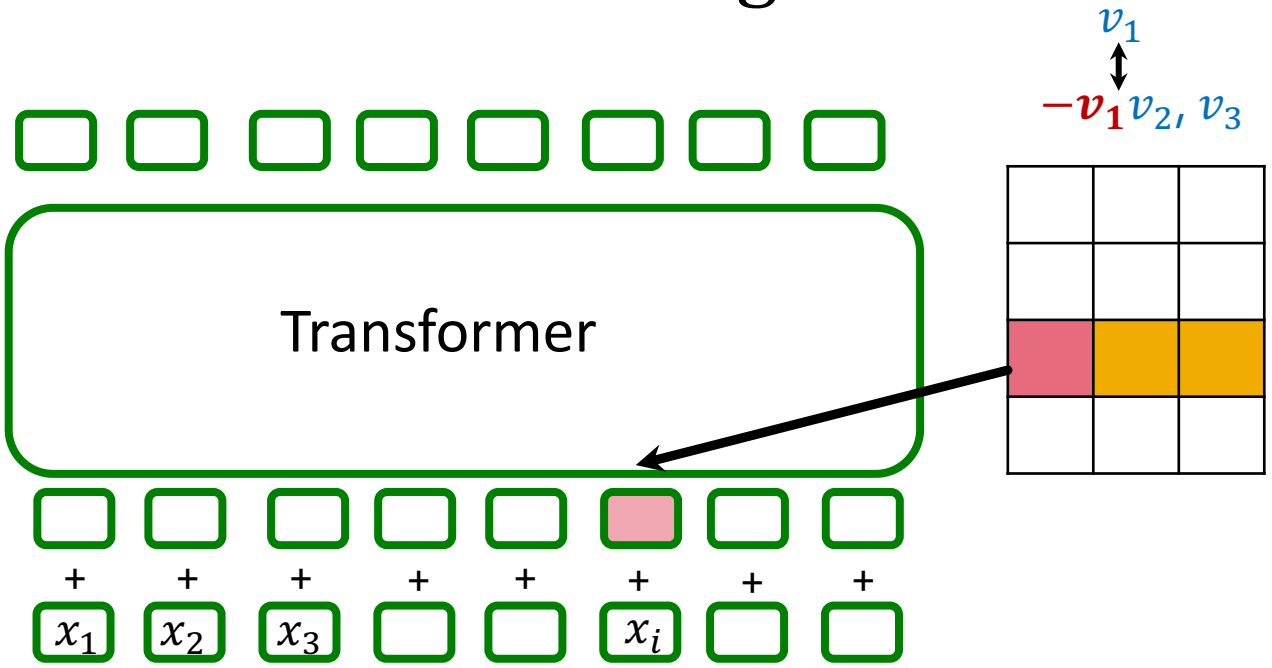
- Both v and $-v$ are eigenvectors
- But when we use them as positional encodings we **pick one arbitrarily**
- **Why does this matter for positional encodings?**

- **What if we picked the other sign?**



Sign Ambiguity is a Problem

- What if we picked the other sign choice?
- **Then the input PE changes**
- **=> The models predictions will change!**
- For k eigenvectors there are 2^k sign choices
  - 2^k different predictions for the same input graph!

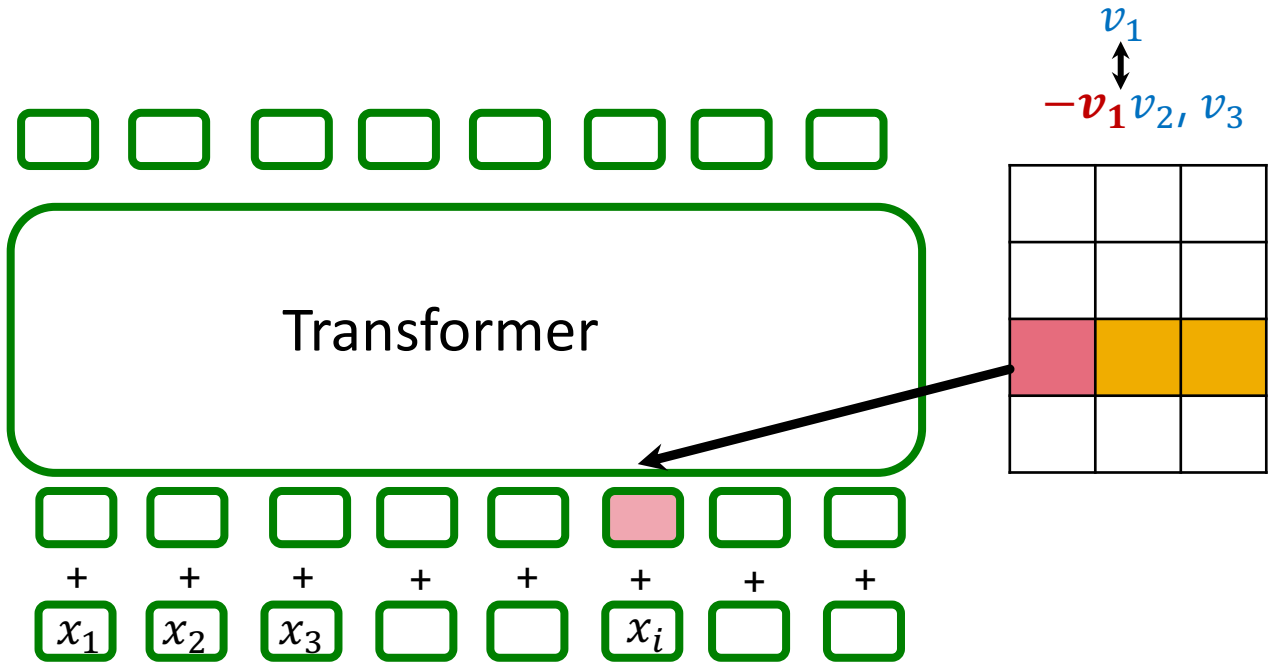


How to fix sign ambiguity

- **Simple Idea:** randomly flip the signs of eigenvectors during training
  - I.e., data augmentation
  - Model will learn to not use the sign information
  - **Issue:** exponentially many sign choices is very difficult to learn

How to fix sign ambiguity

- **Better Idea:** build a neural network that is **invariant** to sign choices!
  - Since it is invariant, the predictions will no longer depend on the sign choice



Sign Invariant Neural Networks

- **Goal:** design a neural network $f(v_1, v_2, \ldots v_k)$ such that:
  - $f(v_1, v_2, \ldots v_k) = f(\pm v_1, \pm v_2, \ldots \pm v_k)$ for all $\pm$ choices
  - f is "expressive": note that $f(v_1, v_2, \ldots v_k) = 0$ is sign invariant… but it's a terrible neural network architecture
- **Warmup: one eigenvector**
  - What about $f(v_1)$ such that $f(v_1) = f(-v_1)$?

Sign Invariant Neural Networks

- **Warmup: one eigenvector**
- **Goal:** design a neural network $f(v_1)$ such that

$$f(v_1) = f(-v_1)$$

Sign Invariant Neural Networks

- **Warmup: one eigenvector**
- **Goal:** design a neural network $f(v_1)$ such that

$$f(v_1) = f(-v_1)$$

- **Proposition:** f satisfies $f(v_1) = f(-v_1)$ if and only if there is a ϕ such that

$$f(v_1) = \phi(v_1) + \phi(-v_1)$$

Sign Invariant Neural Networks

- **Warmup: one eigenvector**
- **Goal:** design a neural network $f(v_1)$ such that

$$f(v_1) = f(-v_1)$$

- **Proposition:** f satisfies $f(v_1) = f(-v_1)$ if and only if there is a ϕ such that

$$f(v_1) = \phi(v_1) + \phi(-v_1)$$

Proof:

<=: If $f(v_1) = \phi(v_1) + \phi(-v_1)$, then $f(-v_1) = \phi(-v_1) + \phi(v_1) = f(v_1)$,

=>: If $f(v_1) = f(-v_1)$, define $\phi(v_1) = f(v_1)/2$.

Then $\phi(v_1) + \phi(-v_1) = f(v_1)/2 + f(-v_1)/2 = f(v_1)$

Sign Invariant Neural Networks

- **Warmup: one eigenvector**
- **Goal:** design a sign invariant neural network $f(v_1, v_2, \ldots v_k)$ in two steps:
  - **Step 1: sign invariant $f_i(v_i)$ for each i**
  - **Step 2: COMBINE individual eigenvector embeddings into one:**

$$f(v_1, v_2, \ldots v_k) = AGG(f_1(v_1), \ldots, f_k(v_k))$$

Sign Invariant Neural Networks

- **Warmup: one eigenvector**
- **Goal:** design a sign invariant neural network $f(v_1, v_2, \ldots v_k)$ in two steps:
  - **Step 1: sign invariant $f_i(v_i)$ for each i**
  - **Step 2: COMBINE individual eigenvector embeddings into one:**

$$f(v_1, v_2, \ldots v_k) = AGG(f_1(v_1), \ldots, f_k(v_k))$$

Use model for one eigenvector

$$f(v_1, v_2, \ldots v_k)$$
$$= AGG(\phi_1(v_1), +\phi_1(-v_1), \ldots, \phi_k(v_k), +\phi_k(-v_k))$$

Combine using another neural net $AGG = \rho$

Sign Invariant Neural Networks

- **Overall model:**

$$f(v_1, v_2, \dots v_k) = \rho(\phi_1(v_1), +\phi_1(-v_1), \dots, \phi_k(v_k), +\phi_k(-v_k))$$

- Introducing k distinct neural nets is costly…
- Let's minimize extra parameters by **sharing one ϕ:**

$$f(v_1, v_2, \dots v_k) = \rho(\phi(v_1), +\phi(-v_1), \dots, \phi(v_k), +\phi(-v_k))$$

ρ, ϕ = any neural network (MLP, GNN etc.)

SignNet

Sign Invariant Neural Networks

- **Recall Goal:** design a neural network $f(v_1, v_2, \ldots v_k)$ such that:
  - $f(v_1, v_2, \ldots v_k) = f(\pm v_1, \pm v_2, \ldots \pm v_k)$ for all $\pm$ choices
    - **SignNet is sign invariant.**
  - f is "expressive"
    - **Is SignNet expressive?**

$$f(v_1, v_2, \ldots v_k) = \rho(\phi(v_1), +\phi(-v_1), \ldots, \phi(v_k), +\phi(-v_k))$$

ρ, ϕ = any neural network (MLP, GNN etc.)

SignNet

Sign Invariant Neural Networks

- **Recall Goal:** design a neural network $f(v_1, v_2, \dots v_k)$ such that:
  - $f(v_1, v_2, \dots v_k) = f(\pm v_1, \pm v_2, \dots \pm v_k)$ for all $\pm$ choices
    - **SignNet is sign invariant.**
  - f is "expressive"
    - **Is SignNet expressive?**

Theorem: if f is sign invariant, then there exist functions ρ, ϕ such that

$$f(v_1, v_2, \dots v_k) = \rho(\phi(v_1), +\phi(-v_1), \dots, \phi(v_k), +\phi(-v_k))$$

SignNet can express all sign invariant functions!!

Sign Invariant Neural Networks

- **Many eigenvector case**
- **Strategy:** design a neural network $f(v_1)$ such that

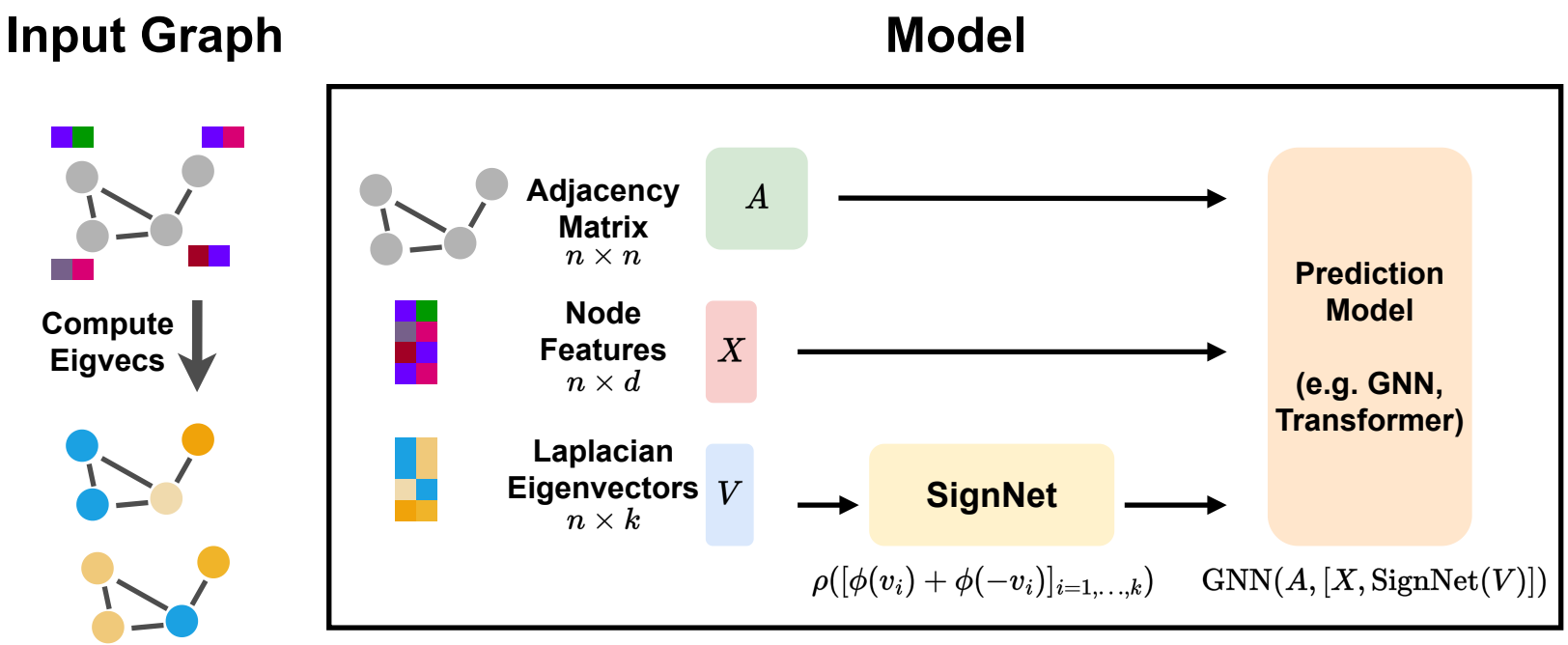
$$f(v_1) = f(-v_1)$$

- **Proposition:** f satisfies $f(v_1) = f(-v_1)$ if and only if there is a ϕ such that

$$f(v_1) = \phi(v_1) + \phi(-v_1)$$

SignNet in practice

- **How to use SignNet in practice?**
  - Step 1: Compute eigenvectors
  - Step 2: get eigenvector embeddings using SignNet
  - Step 3: concatenate SignNet embeddings with node features X
  - Step 4: pass through main GNN/Transformer as usual.
  - Step 5: Backpropagate gradients to train SignNet + Prediction model jointly.

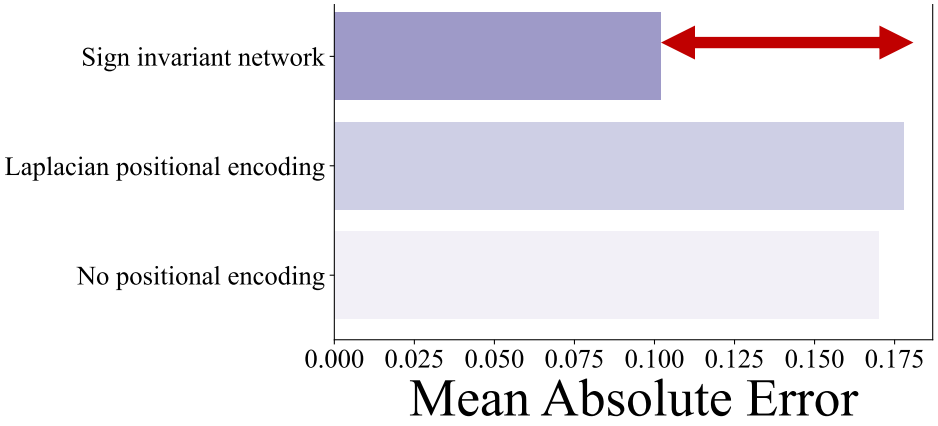


Small molecule property prediction with SignNet

- **Task:** given a small molecule, predict its **solubility**

$$f(\text{MBX-2982}) = \text{solubility}$$

50% reduction in test error



Plan for Today

- **Part 1:**
  - Transformers to message passing on fully connected graph
- **Part 2:**
  - New design landscape for graph Transformers
    - Tokenization
    - Positional encoding
    - Modified self-attention
- **Part 3:**
  - Sign invariant Laplacian positional encodings for graph Transformers

Summary: Graph Transformer Design Space

- New design space for graph Transformers