Abhiroop Sanyal
a5sanyal@uwaterloo.ca

David R. Cheriton School of Computer Science
University of Waterloo

Attention, Learn to Solve Routing Problems!

Work By: Wouter Kool, Herke Van Hoof and Max Welling

ICLR, 2019

Outline

- Problem Definition: Traveling Salesman Problem
- REINFORCE with rollout baseline
- Recap: Attention
- Attention for TSP
- Experimental Results

Traveling Salesman Problem

Problem definition

Given a list of cities and the distances between each pair of cities (weighted complete graph $G = (V, E)$), compute the shortest route that visits every city exactly once and returns to the origin city (Hamiltonian Cycle of least weight).

Traveling Salesman Problem

Problem definition

Given a list of cities and the distances between each pair of cities (weighted complete graph $G = (V, E)$), compute the shortest route that visits every city exactly once and returns to the origin city (Hamiltonian cycle of least weight).

Status of Problem

- Decision version (given D determine if map has route of length at most D): NP-complete
- Search version: NP-hard

Traveling Salesman Problem

Metric TSP

Given any 3 vertices u, v, w the distances between them obey the triangle inequality:

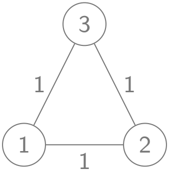
$$d_{uv} \leq d(uw) + d_{wv}$$

Traveling Salesman Problem

Metric TSP

Given any 3 vertices u, v, w the distances between them obey the triangle inequality:

$$d_{uv} \leq d_{uw} + d_{wv}$$



Traveling Salesman Problem

Metric TSP

Given any 3 vertices u, v, w the distances between them obey the triangle inequality:

$$d_{uv} \leq d_{uw} + d_{wv}$$

Euclidean distance metric: Still NP-hard.

Traveling Salesman Problem

Metric TSP

Given any 3 vertices u, v, w the distances between them obey the triangle inequality:

$$d_{uv} \leq d_{uw} + d_{wv}$$

Euclidean distance metric: Still NP-hard.

Given $\epsilon > 0$, $\exists$ deterministic algorithm that finds solutions at most $(1 + \epsilon)$ times the optimal solution in time $O\left((n \log n)^{O(1/\epsilon)}\right)$ (PTAS) [Arora, 1998].

REINFORCE

Assume we are in the Euclidean TSP framework.

REINFORCE

Assume we are in the Euclidean TSP framework.

- Input: $s = ((x_1, y_1), \cdots, (x_n, y_n))$
- Output: Reordering $\pi = (\pi_1, \cdots, \pi_n)$ of nodes of length $L(\pi)$.
- Objective: Minimize $L(\pi)$.

REINFORCE

- Input: $s = \left((x_1, y_1), \cdots, (x_n, y_n)\right)$
- Output: Reordering $\pi = (\pi_1, \cdots, \pi_n)$ of nodes of length $L(\pi)$.
- Objective: Minimize $L(\pi)$.

Additional Assumption: Given any instance s, suppose we have a probability distribution $P_\theta\left(\pi | s\right)$ from which we can sample to get tour $(\pi | s)$.

REINFORCE

Additional Assumption: Given any instance s, suppose we have a probability distribution $P_\theta\left(\pi|s\right)$ from which we can sample to get tour $(\pi|s)$.

Factor model

The above distribution obeys the property:

$$P_\theta(\pi|s) = \prod_{i=1}^{n} P_\theta\left(\pi_i|s, \pi_{<i}\right)$$

REINFORCE

Factor model

The above distribution obeys the property:

$$P_\theta(\pi|s) = \prod_{i=1}^{n} P_\theta\left(\pi_i|s, \pi_{<i}\right)$$

At step t, Sample $\pi_t \sim P_\theta\left(\pi_t|s, \pi_{<t}\right)$.

REINFORCE

Factor model

The above distribution obeys the property:

$$P_\theta(\pi|s) = \prod_{i=1}^{n} P_\theta\left(\pi_i | s, \pi_{<i}\right)$$

At step t, Sample $\pi_t \sim P_\theta\left(\pi_t | s, \pi_{<t}\right)$.

Loss function:= Expected cost of the randomized algorithm:

$$\mathcal{L}\left(\theta | s\right) = \mathbb{E}_{P_\theta(\pi|s)}[L(\pi)]$$

REINFORCE

Factor model

The above distribution obeys the property:

$$P_\theta(\pi|s) = \prod_{i=1}^{n} P_\theta\left(\pi_i|s, \pi_{<i}\right)$$

At step t, Sample $\pi_t \sim P_\theta\left(\pi_t|s, \pi_{<t}\right)$.

Loss function:= Expected cost of the randomized algorithm:

$$\mathcal{L}\left(\theta|s\right) = \mathbb{E}_{P_\theta(\pi|s)}[L(\pi)]$$

Question: How do you optimize for θ?

REINFORCE

Idea: Suppose we have a probability distribution $P_\theta(\cdot|s)$.

- Sample $\pi \sim P_\theta(\cdot|s)$.

REINFORCE

Idea: Suppose we have a probability distribution $P_\theta(\cdot|s)$.

- Sample $\pi \sim P_\theta(\cdot|s)$.
- Compute $L[\pi|s]$.

REINFORCE

Idea: Suppose we have a probability distribution $P_\theta(\cdot|s)$.

- Sample $\pi \sim P_\theta(\cdot|s)$.
- Compute $L[\pi|s]$.
- Compare $L[\pi|s]$ to some cleverly selected baseline $b(s)$.

REINFORCE

Idea: Suppose we have a probability distribution $P_\theta(\cdot|s)$.

- Sample $\pi \sim P_\theta(\cdot|s)$.
- Compute $L[\pi|s]$.
- Compare $L[\pi|s]$ to some cleverly selected baseline $b(s)$.
- Adjust $P_\theta[\pi|s]$ proportional to $L(\pi|s) - L\left(\pi^{b(s)}|s\right)$

REINFORCE

Question: How is the baseline selected?

REINFORCE

Question: How is the baseline selected?

- Cost of a solution from a deterministic greedy rollout of the best policy.

REINFORCE

Question: How is the baseline selected?

- Cost of a solution from a deterministic greedy rollout of the best policy.
- Made deterministic by greedily picking the action with maximum probability.

REINFORCE

Question: How is the baseline selected?

- Cost of a solution from a deterministic greedy rollout of the best policy.
- Made deterministic by greedily picking the action with maximum probability.
- In each epoch, the baseline is fixed.

REINFORCE

Question: How is the baseline selected?

- Cost of a solution from a deterministic greedy rollout of the best policy.
- Made deterministic by greedily picking the action with maximum probability.
- In each epoch, the baseline is fixed.
- At the end of every epoch, compare training policy to baseline using a paired t-test on 10,000 separate evaluation instances.

REINFORCE

Question: How is the baseline selected?

- Cost of a solution from a deterministic greedy rollout of the best policy.
- Made deterministic by greedily picking the action with maximum probability.
- In each epoch, the baseline is fixed.
- At the end of every epoch, compare training policy to baseline using a paired t-test on 10,000 separate evaluation instances.
- In case of significant improvement, update baseline and sample fresh evaluation instances (to avoid overfitting).

REINFORCE

Algorithm 1 REINFORCE with Rollout Baseline

```
Input: number of epochs E, steps per epoch T, batch size B, significance α
Init θ, θ^BL ← θ
for epoch = 1, ..., E do
    for step = 1, ..., T do
        s_i ← RandomInstance() ∀i ∈ {1, ..., B}
        π_i ← SampleRollout(s_i, p_θ) ∀i ∈ {1, ..., B}
        π_i^BL ← GreedyRollout(s_i, p_θBL) ∀i ∈ {1, ..., B}
        ∇L ← Σ_{i=1}^{B} (L(π_i) − L(π_i^BL)) ∇_θ log p_θ(π_i)
        θ ← Adam(θ, ∇L)
    end for
    if OneSidedPairedTTest(p_θ, p_θBL) < α then
        θ^BL ← θ
    end if
end for
```

Where:

- $\theta^{\mathrm{BL}} \leftarrow \theta$
- $s_i \leftarrow \mathrm{RandomInstance}()\ \forall i \in \{1, \ldots, B\}$
- $\boldsymbol{\pi}_i \leftarrow \mathrm{SampleRollout}(s_i, p_{\boldsymbol{\theta}})\ \forall i \in \{1, \ldots, B\}$
- $\boldsymbol{\pi}_i^{\mathrm{BL}} \leftarrow \mathrm{GreedyRollout}(s_i, p_{\boldsymbol{\theta}^{\mathrm{BL}}})\ \forall i \in \{1, \ldots, B\}$
- $\nabla \mathcal{L} \leftarrow \sum_{i=1}^{B} \left(L(\boldsymbol{\pi}_i) - L(\boldsymbol{\pi}_i^{\mathrm{BL}})\right) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi}_i)$
- $\boldsymbol{\theta} \leftarrow \mathrm{Adam}(\boldsymbol{\theta}, \nabla \mathcal{L})$
- $\mathrm{OneSidedPairedTTest}(p_{\boldsymbol{\theta}}, p_{\boldsymbol{\theta}^{\mathrm{BL}}}) < \alpha$

Attention Mechanism

Question: How do we get the probability distribution?

Attention Mechanism

Question: How do we get the probability distribution?

Answer: Attention + Graph Convolutions.

Attention Mechanism

Question: How do we get the probability distribution?

Answer: Attention + Graph Convolutions.

The authors interpret the attention mechanism (Vaswani et al. 2017) as weighted message-passing between nodes.

Attention Mechanism: Recap

Basic Self-Attention

- Input: Words $x_1, ..., x_n$ as vectors in $\mathbb{R}^k$.

Attention Mechanism: Recap

Basic Self-Attention

- Input: Words $x_1, ..., x_n$ as vectors in $\mathbb{R}^k$.
- Output: $y_1, ..., y_n$, where y_i are weighted average over the vectors x_i i.e $y_i = \sum_{j=1}^{n} w_{ij} x_j$.

Attention Mechanism: Recap

Basic Self-Attention

- Input: Words $x_1, ..., x_n$ as vectors in $\mathbb{R}^k$.
- Output: $y_1, ..., y_n$, where y_i are weighted average over the vectors x_i i.e $y_i = \sum_{j=1}^{n} w_{ij} x_j$.

Simplest choice of weight: $w_{ij} = x_i^T x_j$.

Attention Mechanism: Recap

Basic Self-Attention

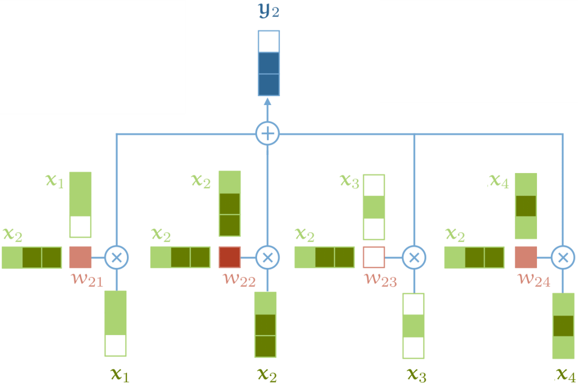
- Input: Words $x_1, ..., x_n$ as vectors in $\mathbb{R}^k$.
- Output: $y_1, ..., y_n$, where y_i are weighted average over the vectors x_i i.e $y_i = \sum_{j=1}^{n} w_{ij} x_j$.

Simplest choice of weight: $w_{ij}^0 = x_i^T x_j$.

Since $w_{ij} \in (-\infty, \infty)$, we apply softmax to normalize:

$$w_{ij} = \frac{exp\left(w_{ij}^0\right)}{\sum_{j=1}^{n} exp\left(w_{ij}^0\right)}$$

Attention Mechanism: Recap



A visual illustration of basic self-attention. Note that the softmax operation over the weights is not illustrated.

Attention Mechanism: Recap

Self-Attention

Consider learnable matrices W_q, W_k and $W_v \in \mathbb{R}^{k \times k}$.

Attention Mechanism: Recap

Self-Attention

Consider learnable matrices W_q, W_k and $W_v \in \mathbb{R}^{k \times k}$. Compute:

$$q_i = W_q x_i \ \ \text{(queries)}, \ \ k_i = W_k x_i \ \ \text{(keys)}, \ \ v_i = W_v x_i \ \ \text{(values)}$$

Attention Mechanism: Recap

Self-Attention

Consider learnable matrices W_q, W_k and $W_v \in \mathbb{R}^{k \times k}$. Compute:

$$q_i = W_q x_i \ \text{(queries)}, \ k_i = W_k x_i \ \text{(keys)}, \ v_i = W_v x_i \ \text{(values)}$$

Weights are computed as:

$$w_{ij}^{(1)} = \frac{q_i^T k_j}{\sqrt{k}}$$

$$w_{ij} = \text{softmax}\left(w_{ij}^{(1)}\right)$$

Attention Mechanism: Recap

Self-Attention

Consider learnable matrices W_q, W_k and $W_v \in \mathbb{R}^{k \times k}$. Compute:

$$q_i = W_q x_i \ \text{(queries)}, \ k_i = W_k x_i \ \text{(keys)}, \ v_i = W_v x_i \ \text{(values)}$$

Weights are computed as:

$$w_{ij}^{(1)} = \frac{q_i^T k_j}{\sqrt{k}}$$

$$w_{ij} = \text{softmax}\left(w_{ij}^{(1)}\right)$$

Output:

$$y_i = \sum_{j=1}^{n} w_{ij} v_j$$

Attention Mechanism: Recap

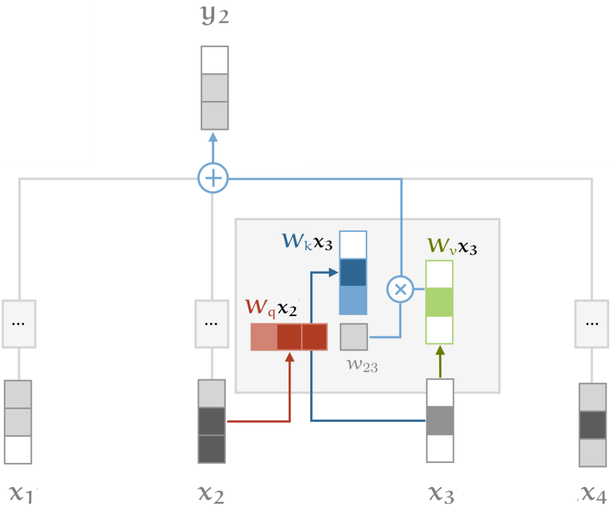


Illustration of the self-attention with key, query and value transformations.

Attention Mechanism: Recap

Multi-head attention

Consider matrices W_q^t, W_k^t and W_v^t for self-attention heads labeled by $1 \leq t \leq M$.

Attention Mechanism: Recap

Multi-head attention

Consider matrices W_q^t, W_k^t and W_v^t for self-attention heads labeled by $1 \leq t \leq M$. For each x_i, we get a vector y_i^t corresponding to head t.

Attention Mechanism: Recap

Multi-head attention

Consider matrices W_q^t, W_k^t and W_v^t for self-attention heads labeled by $1 \leq t \leq M$. For each x_i, we get a vector y_i^t corresponding to head t.

Output: Concatenation of the vectors y_i^t multiplied by a matrix to reduce dimension back to k.

Attention Mechanism: Recap

Efficient Implementation of MHA

With M heads the above operation would seem M time slower.

Efficient Implementation of MHA

With M heads the above operation would seem M time slower. Solution: Pass low-dimensional keys, queries, and values to the heads.

Attention Mechanism: Recap

Efficient Implementation of MHA

With M heads the above operation would seem M time slower. Solution: Pass low-dimensional keys, queries, and values to the heads.

Given M heads, consider $3M$ learnable matrices W_k^t, W_q^t, $W_v^t \in \mathbb{R}^{k \times \frac{k}{M}}$ corresponding to keys, queries and values.

Attention Mechanism: Recap

Efficient Implementation of MHA

With M heads the above operation would seem M time slower. Solution: Pass low-dimensional keys, queries, and values to the heads.

Given M heads, consider $3M$ learnable matrices W_k^t, W_q^t, $W_v^t \in \mathbb{R}^{k \times \frac{k}{M}}$ corresponding to keys, queries and values.

The input to each of the M heads now has dimension k/M.

Attention Mechanism: Recap

Efficient Implementation of MHA

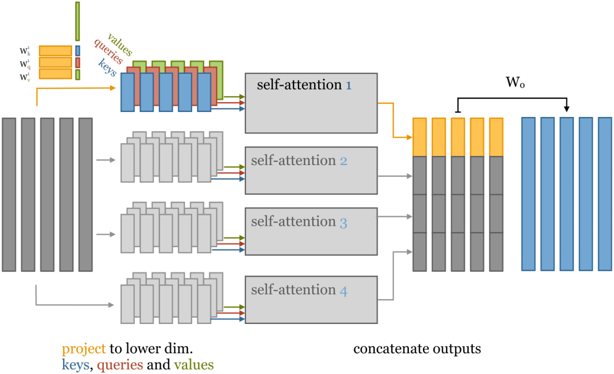
With M heads the above operation would seem M time slower. Solution: Pass low-dimensional keys, queries, and values to the heads.

Given M heads, consider $3M$ learnable matrices W_k^t, W_q^t, $W_v^t \in \mathbb{R}^{k\times \frac{k}{M}}$ corresponding to keys, queries and values.

The input to each of the M heads now has dimension k/M.

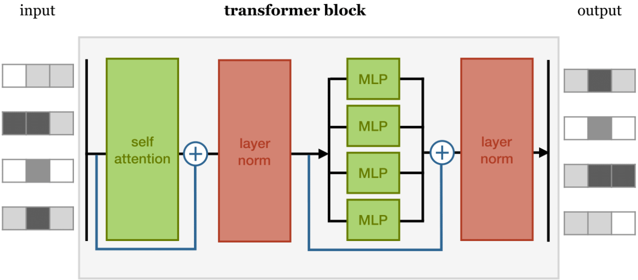
Number of parameters $= 3M \times k \times \frac{k}{M} = 3k^2 = O(k^2)$.

Attention Mechanism: Recap



The basic idea of multi-head self-attention with 4 heads. To get our keys, queries and values, we project the input down to vector sequences of smaller dimension.

Transformer Architecture



Attention as Message-Passing Mechanism for TSP

The parameter matrices W_q, $W_k \in \mathbb{R}^{d_k \times d_h}$ and $W_v \in \mathbb{R}^{d_v \times d_h}$.

We have $q_i = W^Q h_i$ where h_i is d_h-dimensional embedding of i-th node (similarly k_i and v_i).

Attention Mechanism for TSP

Compatibility between nodes i and j is computed as:

$$u_{ij} := \begin{cases} \frac{q_i^T k_j}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are adjacent} \\ -\infty & \text{otherwise} \end{cases}$$

Attention as Message-Passing Mechanism for TSP

Compatibility between nodes i and j is computed as:

$$u_{ij} := \begin{cases} \frac{q_i^T k_j}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are adjacent} \\ -\infty & \text{otherwise} \end{cases}$$

Setting compatibility to $-\infty$ between non-adjacent nodes ensures no message passing between these nodes.

Attention as Message-Passing Mechanism for TSP

Compatibility between nodes i and j is computed as:

$$u_{ij} := \begin{cases} \frac{q_i^T k_j}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are adjacent} \\ -\infty & \text{otherwise} \end{cases}$$

The attention weights a_{ij} are computed as:

$$a_{ij} = softmax(u_{ij})$$

Message $h_i^{'}$ received by node i is:

$$h_i' = \sum_j a_{ij} v_j$$

Attention as Message-Passing Mechanism for TSP

Since using Multi-headed Attention is beneficial, h_i' is computed $M = 8$ times using parameters $d_k = d_v = \frac{d_h}{M} = 16$.

Attention as Message-Passing Mechanism for TSP

Since using Multi-headed Attention is beneficial, h'_i is computed $M = 8$ times using parameters $d_k = d_v = \frac{d_h}{M} = 16$.

The result is projected back to a d_h dimensional vector using using $d_h \times d_v$ parameter matrices:

$$\mathrm{MHA}_i\left(h_1, ..., h_n\right) = \sum_{m=1}^{M} W_m h'_{im}$$

Model: Attention based encoder-decoder

Objective: Use attention-based encoder-decoder model to give stochastic policy $P_\theta(\pi|s)$ for selecting solution π given problem instance s.

Model: Attention based encoder-decoder

Objective: Use attention-based encoder-decoder model to give stochastic policy $P_\theta(\pi|s)$ for selecting solution π given problem instance s.

- Encoder: Produces embeddings of all input nodes.
- Decoder: Takes as input encoder embeddings and a problem-specific context and mask. Outputs the sequence π.

Attention based encoder

Each input coordinate x_i is 2-dimensional. Produce $d_h = 128$ dimensional initial embeddings using affine projections: $h_i^{(0)} = W^X x_i + b_X$.

Attention based encoder

Each input coordinate x_i is 2-dimensional. Produce $d_h = 128$ dimensional initial embeddings using affine projections: $h_i^{(0)} = W^X x_i + b_X$.

- Embeddings updated using N attention layers. Let $h_i^{(l)}$ be the embedding produced at layer l.

Attention based encoder

Each input coordinate x_i is 2-dimensional. Produce $d_h = 128$ dimensional initial embeddings using affine projections: $h_i^{(0)} = W^X x_i + b_X$.

- Embeddings updated using N attention layers. Let $h_i^{(l)}$ be the embedding produced at layer l. The final graph embedding is given by:

$$h_G^{(N)} = \frac{1}{n} \sum_{i=1}^{n} h_i^N$$

Attention based encoder

- Embeddings updated using N attention layers. Let $h_i^{(l)}$ be the embedding produced at layer l. The final graph embedding is given by:

$$h_G^{(N)} = \frac{1}{n}\sum_{i=1}^{n} h_i^N$$

- Each attention layer is as follows:

$$\hat{h_i} = \mathsf{BN}^l\left(h_i^{(l-1)} + \mathsf{MHA}\left(h_1^{(l-1)}, ..., h_n^{(l-1)}\right)\right)$$

$$h_i^{(l)} = BN^l\left(\hat{h_i} + FF^l\left(\hat{h_i}\right)\right)$$

Attention-based encoder

- Each attention layer is as follows:

$$\hat{h_i} = \mathsf{BN}^l \left(h_i^{(l-1)} + \mathsf{MHA} \left(h_1^{(l-1)}, ..., h_n^{(l-1)} \right) \right)$$

$$h_i^{(l)} = BN^l \left(\hat{h_i} + FF^l \left(\hat{h_i} \right) \right)$$

The layers do not share parameters. MHA has M heads with dimensionality $\frac{d_h}{M} = 8$.

Attention-based encoder

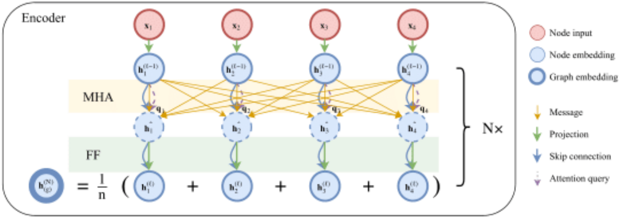


Figure 1: Attention based encoder. Input nodes are embedded and processed by N sequential layers, each consisting of a multi-head attention (MHA) and node-wise feed-forward (FF) sub-layer. The graph embedding is computed as the mean of node embeddings. Best viewed in color.

Attention-based decoder

Idea: Suppose a partial tour is constructed. The goal is to find a tour from the last visited node to the first node, through all the unvisited nodes.

Attention-based decoder

The tour is constructed one node at a time and at timestep t the decoder outputs π_t depending on embeddings of the encoder and the outputs $\pi_{t'}$ $\forall$ $t' < t$.

Attention-based decoder

Idea: Attention layer on top of encode with only messages to a special context node c.

Attention-based decoder

Idea: Attention layer on top of encode with only messages to a special context node c.

Context embedding:

$$h_c^{(N)} = \begin{cases} \left[h_G^{(N)}; h_{\pi_{t-1}}^{(N)}; h_{\pi_1}^{N}\right] & \forall t > 1 \\ \left[h_G^{(N)}, v^1, v^2\right] & t > 1 \end{cases}$$

Attention-based decoder

Compute $h_c^{(N+1)}$ using the attention mechanism described before, the difference being:

Attention-based decoder

Compute $h_c^{(N+1)}$ using the attention mechanism described before, the difference being:

$$q_c = W_q h_c \quad k_i = W_k h_i \quad v_i = W_v h_i$$

Attention-based decoder

Compute $h_c^{(N+1)}$ using the attention mechanism described before, the difference being:

$$q_c = W_q h_c \quad k_i = W_k h_i \quad v_i = W_v h_i$$

Compatibility between c and j:

$$u_{cj} = \begin{cases} \frac{q_c^T k_j}{\sqrt{d_k}} & j \neq \pi_{t'} \quad \forall t' < t \\ -\infty & \text{otherwise} \end{cases}$$

Attention-based decoder

Compute $h_c^{(N+1)}$ using the attention mechanism described before, the difference being:

$$q_c = W_q h_c \;\; k_i = W_k h_i \;\; v_i = W_v h_i$$

Compatibility between c and j:

$$u_{cj} = \begin{cases} \frac{q_c^T k_j}{\sqrt{d_k}} & j \neq \pi_{t'} \;\; \forall t' < t \\ -\infty & \text{otherwise} \end{cases}$$

This makes sure that the nodes visited already have been masked.

Attention based decoder

Final computation of probabilities: Use one final decoder layer with a single attention head ($M = 1$, $d_k = d_h$).

Attention based decoder

Final computation of probabilities: Use one final decoder layer with a single attention head ($M = 1$, $d_k = d_h$).

$$u_{cj} = \begin{cases} C \cdot \text{tanh}\left(\frac{q_c^T k_j}{\sqrt{d_k}}\right) & j \neq \pi_{t'} \quad \forall t' < t \\ -\infty & \text{otherwise} \end{cases}$$

where $C \in [-10, 10]$.

Attention based decoder

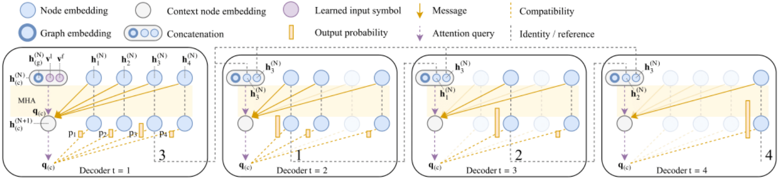
Final computation of probabilities: Use one final decoder layer with a single attention head ($M = 1$, $d_k = d_h$).

$$u_{cj} = \begin{cases} C \cdot \tanh\left(\frac{q_c^T k_j}{\sqrt{d_k}}\right) & j \neq \pi_{t'} \quad \forall t' < t \\ -\infty & \text{otherwise} \end{cases}$$

where $C \in [-10, 10]$.

$$p_i = P_\theta\left(\pi_t = i | s, \pi_{1:t-1}\right) = \frac{exp(u_{ci})}{\sum_j exp(u_{cj})}$$

Attention-based decoder



Experimental Results

Table 1: Attention Model (AM) vs baselines. The gap % is w.r.t. the best value across all methods.

	Method	$n = 20$ Obj.	Gap	Time	$n = 50$ Obj.	Gap	Time	$n = 100$ Obj.	Gap	Time
TSP	Concorde	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
	LKH3	3.84	0.00%	(18s)	5.70	0.00%	(5m)	7.76	0.00%	(21m)
	Gurobi	3.84	0.00%	(7s)	5.70	0.00%	(2m)	7.76	0.00%	(17m)
	Gurobi (1s)	3.84	0.00%	(8s)	5.70	0.00%	(2m)		-	
	Nearest Insertion	4.33	12.91%	(1s)	6.78	19.03%	(2s)	9.46	21.82%	(6s)
	Random Insertion	4.00	4.36%	(0s)	6.13	7.65%	(1s)	8.52	9.69%	(3s)
	Farthest Insertion	3.93	2.36%	(1s)	6.01	5.53%	(2s)	8.35	7.59%	(7s)
	Nearest Neighbor	4.50	17.23%	(0s)	7.00	22.94%	(0s)	9.68	24.73%	(0s)
	Vinyals et al. (gr.)	3.88	1.15%		7.66	34.48%			-	
	Bello et al. (gr.)	3.89	1.42%		5.95	4.46%		8.30	6.90%	
	Dai et al.	3.89	1.42%		5.99	5.16%		8.31	7.03%	
	Nowak et al.	3.93	2.46%			-			-	
	EAN (greedy)	3.86	0.66%	(2m)	5.92	3.98%	(5m)	8.42	8.41%	(8m)
	AM (greedy)	**3.85**	**0.34%**	(0s)	**5.80**	**1.76%**	(2s)	**8.12**	**4.53%**	(6s)
	OR Tools	3.85	0.37%		5.80	1.83%		7.99	2.90%	
	Chr.f. + 2OPT	3.85	0.37%		5.79	1.65%			-	
	Bello et al. (s.)		-		5.75	0.95%		8.00	3.03%	
	EAN (gr. + 2OPT)	3.85	0.42%	(4m)	5.85	2.77%	(26m)	8.17	5.21%	(3h)
	EAN (sampling)	3.84	0.11%	(5m)	5.77	1.28%	(17m)	8.75	12.70%	(56m)
	EAN (s. + 2OPT)	3.84	0.09%	(6m)	5.75	1.00%	(32m)	8.12	4.64%	(5h)
	AM (sampling)	**3.84**	**0.08%**	(5m)	**5.73**	**0.52%**	(24m)	**7.94**	**2.26%**	(1h)