南京航空航天大學

Nanjing University of Aeronautics and Astronautics

SEQUENCE LEVEL TRAINING WITH RECURRENT NEURAL NETWORKS

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, Wojciech Zaremba
Facebook AI Research
{ranzato, spchopra, michealauli, wojciech}@fb.com

ICLR 2016

introduce

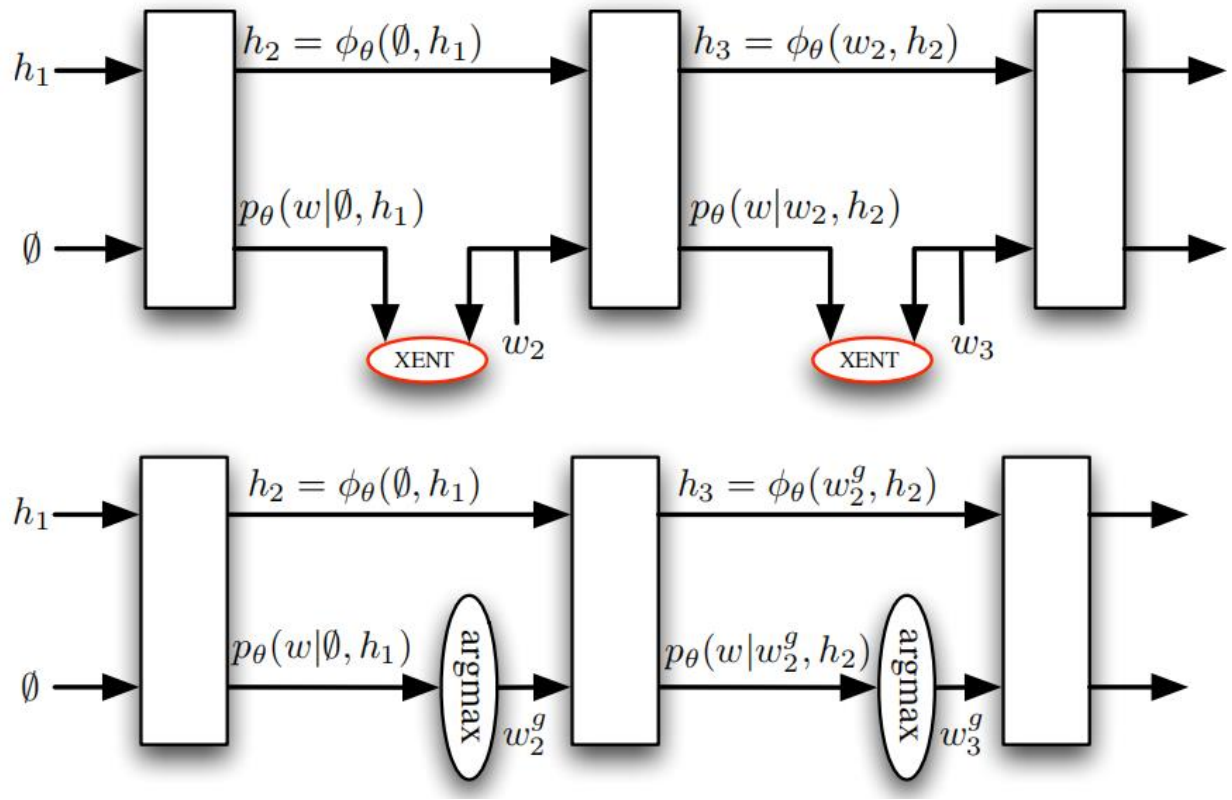


Figure 1: RNN training using XENT (top), and how it is used at test time for generation (bottom). The RNN is unfolded for three time steps in this example. The red oval is a module computing a loss, while the rectangles represent the computation done by the RNN at one step. At the first step, all inputs are given. In the remaining steps, the input words are clamped to ground truth at training time, while they are clamped to model predictions (denoted by w_t^g) at test time. Predictions are produced by either taking the argmax or by sampling from the distribution over words.

introduce

Conventional training with XENT suffers from exposure bias since training uses ground truth words as opposed to model predictions

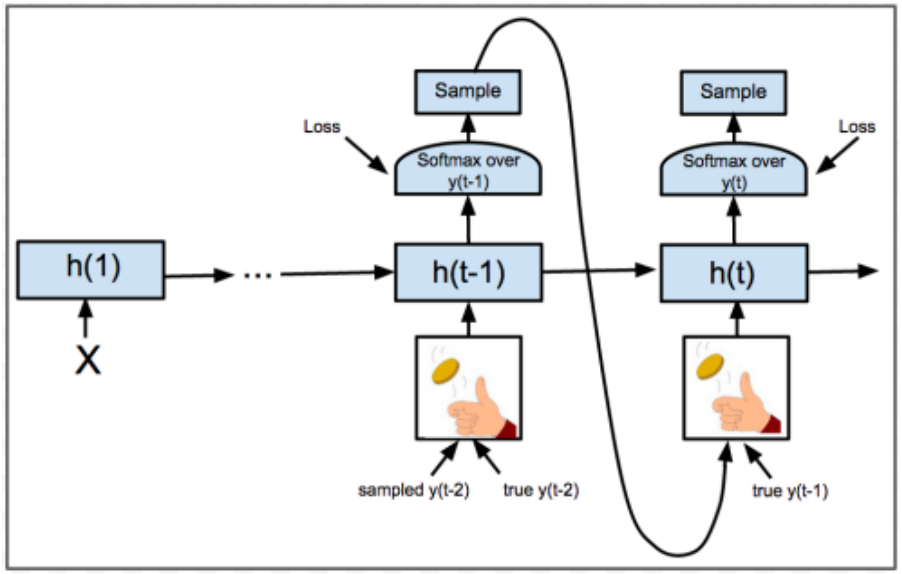


Figure 1: Illustration of the Scheduled Sampling approach, where one flips a coin at every time step to decide to use the true previous token or one sampled from the model itself.

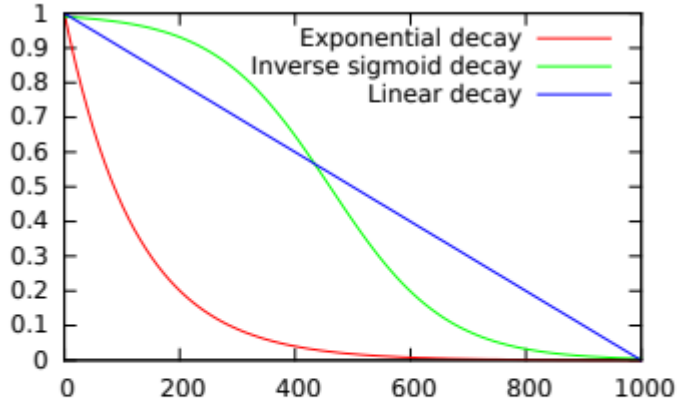


Figure 2: Examples of decay schedules.

introduce

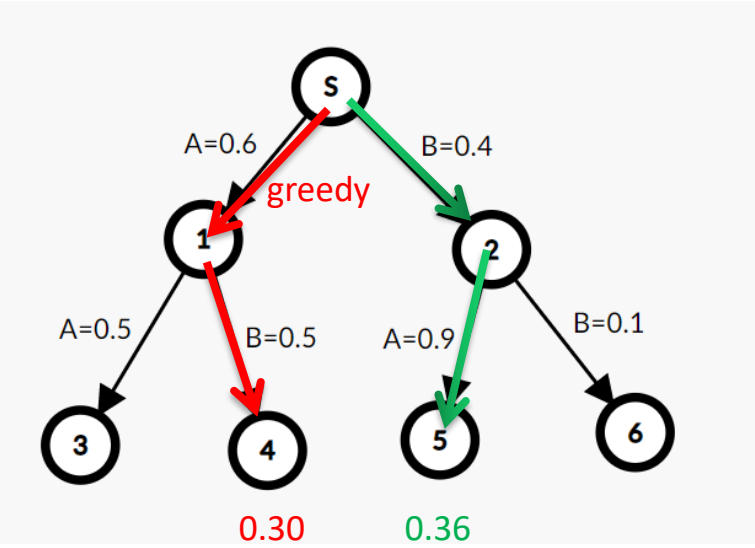
Cross-entropy loss (XENT) maximizes the probability of the observed sequence according to the model. If the target sequence is $[w_1, w_2, \ldots, w_T]$, then XENT training involves minimizing:

$$L = -\log p(w_1, \ldots, w_T) = -\log \prod_{t=1}^{T} p(w_t | w_1, \ldots, w_{t-1}) = -\sum_{t=1}^{T} \log p(w_t | w_1, \ldots, w_{t-1}). \quad (6)$$

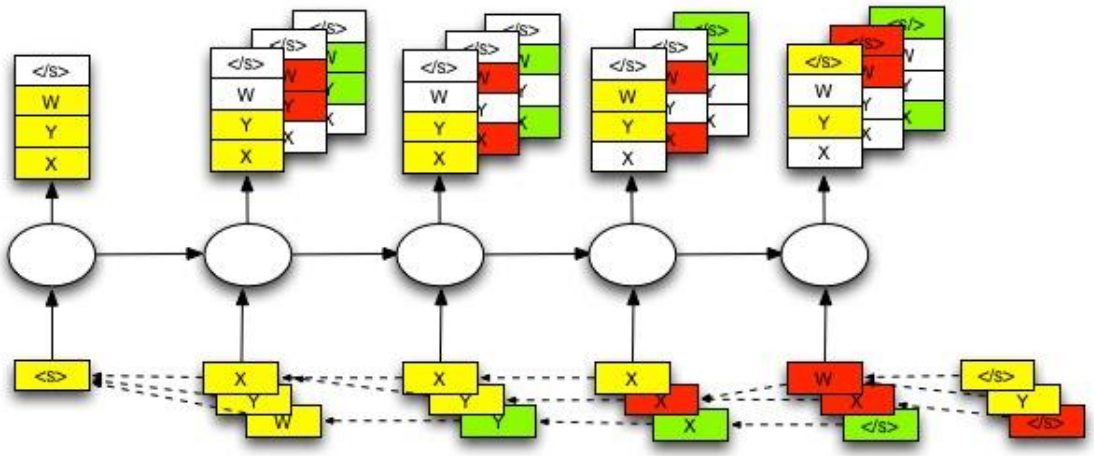
Then the next word is generated by: $w_{t+1}^g = \operatorname{argmax}_w p_\theta(w | w_t^g, \mathbf{h}_{t+1})$.

search error :

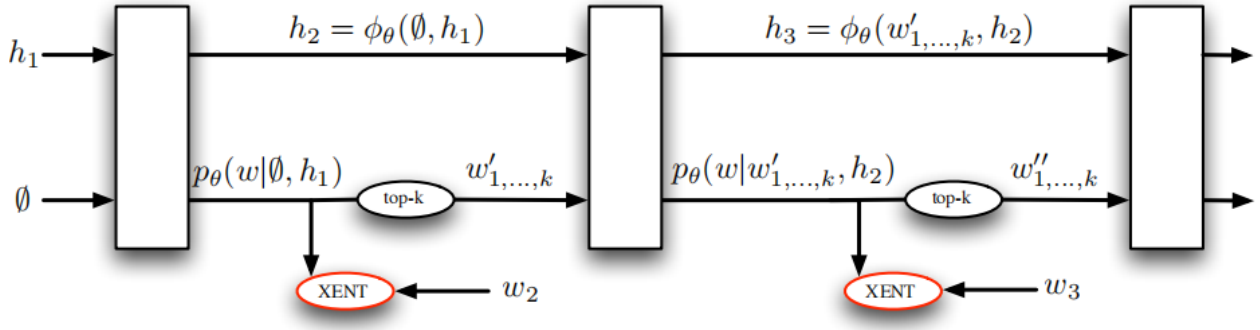
$$\prod_{t=1}^{T} \max_{w_{t+1}} p_\theta(w_{t+1} | w_t^g, \mathbf{h}_{t+1}) \leq \max_{w_1, \ldots, w_T} \prod_{t=1}^{T} p_\theta(w_{t+1} | w_t^g, \mathbf{h}_{t+1})$$



END-TO-END BACKPROP (E2E)



beam search



While this algorithm is a simple way to expose the model to its own predictions, the loss function optimized is still XENT at each time step. There is no explicit supervision at the sequence level while training the model.

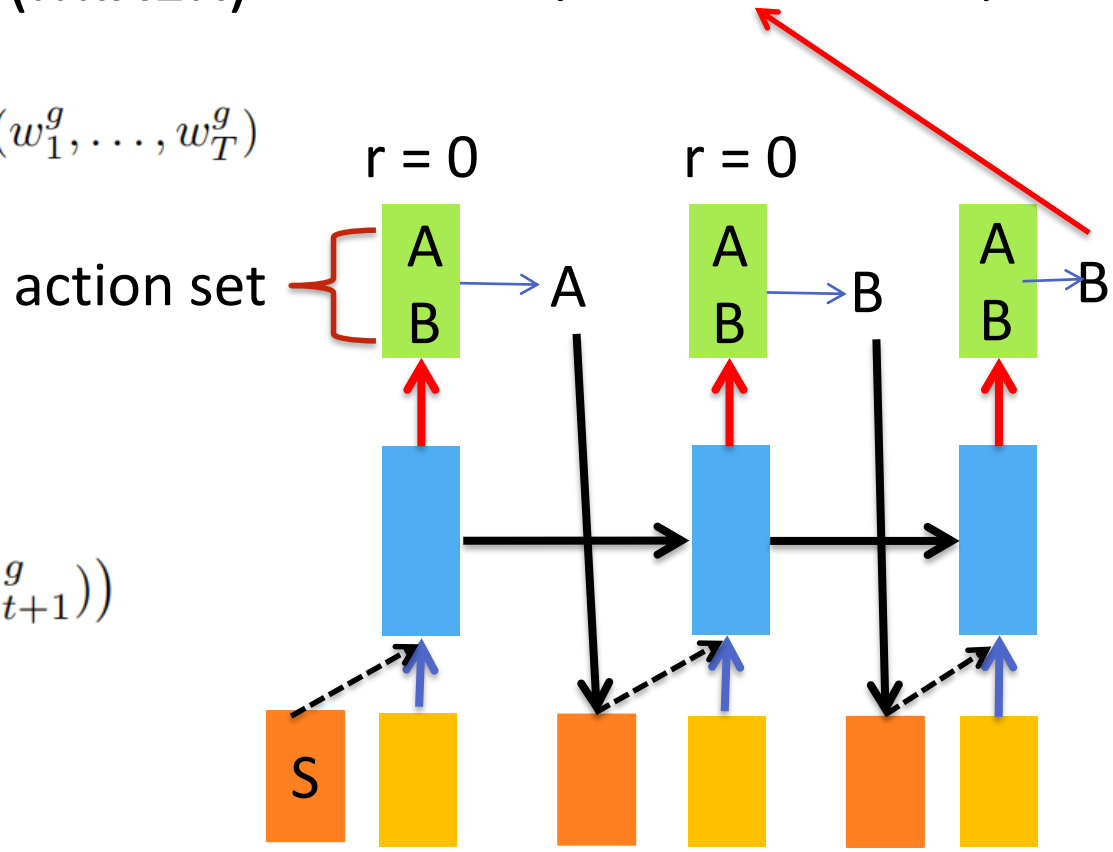
Sequence level training

Mixed Incremental Cross-Entropy Reinforce (MIXER)

$$L_\theta = -\sum_{w_1^g,\ldots,w_T^g} p_\theta(w_1^g,\ldots,w_T^g) r(w_1^g,\ldots,w_T^g) = -\mathbb{E}_{[w_1^g,\ldots w_T^g]\sim p_\theta} r(w_1^g,\ldots,w_T^g)$$

$$\frac{\partial L_\theta}{\partial \theta} = \sum_t \frac{\partial L_\theta}{\partial \mathbf{o}_t}\frac{\partial \mathbf{o}_t}{\partial \theta}$$

$$\frac{\partial L_\theta}{\partial \mathbf{o}_t} = \left(r(w_1^g,\ldots,w_T^g) - \bar{r}_{t+1}\right)\left(p_\theta(w_{t+1}|w_t^g,\mathbf{h}_{t+1},\mathbf{c}_t) - \mathbf{1}(w_{t+1}^g)\right)$$



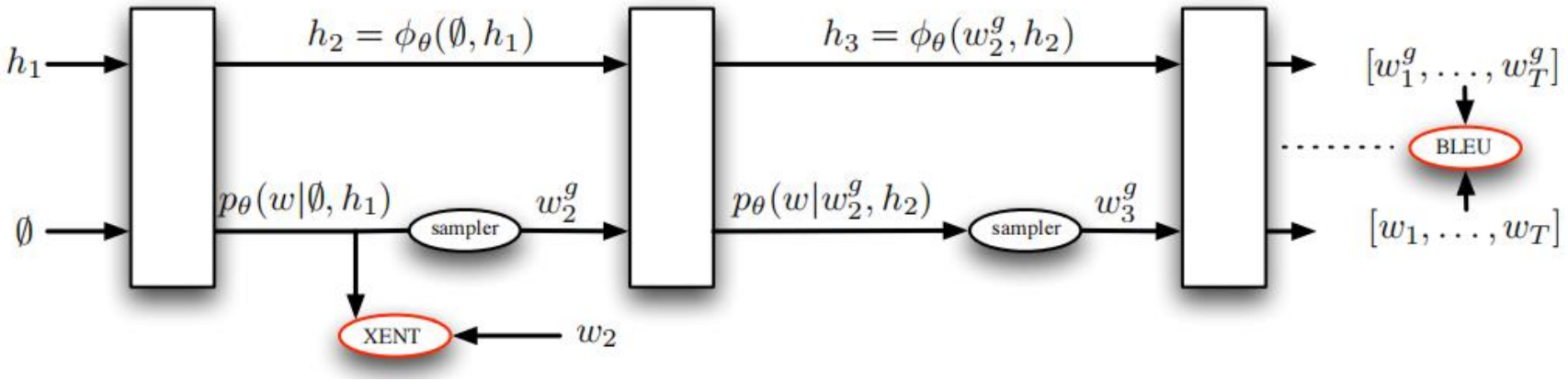
Mixed incremental cross-entropy reinforce (MIXER)

One major drawback associated with the algorithm is that it assumes a random policy to start with. This assumption can make the learning for large action spaces very challenging.

1、We first train the RNN with the cross-entropy loss for N^{XENT} epochs using the ground truth sequences.

2、introduce model predictions during training with an annealing schedule in order to gradually teach the model to produce stable sequences.

algorithm



Data: a set of sequences with their corresponding context.
Result: RNN optimized for generation.
Initialize RNN at random and set N^{XENT}, $N^{\text{XE+R}}$ and Δ;
for $s = T, 1, -\Delta$ **do**
  if $s == T$ **then**
    train RNN for N^{XENT} epochs using XENT only;
  else
    train RNN for $N^{\text{XE+R}}$ epochs. Use XENT loss in the first s steps, and REINFORCE (sampling from the model) in the remaining $T - s$ steps;
  end
end

Algorithm 1: MIXER pseudo-code.

experiment

TASK	XENT	DAD	E2E	MIXER
summarization	13.01	12.18	12.78	**16.22**
translation	17.74	20.12	17.77	**20.73**
image captioning	27.8	28.16	26.42	**29.16**

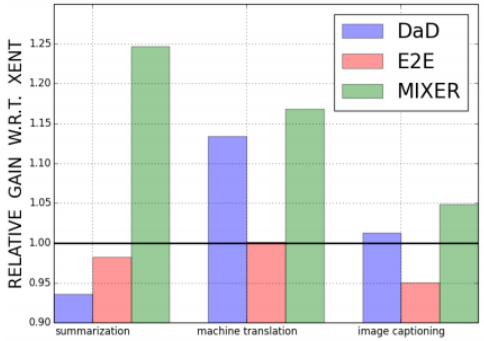


Figure 5: Left: BLEU-4 (translation and image captioning) and ROUGE-2 (summarization) scores using greedy generation. Right: Relative gains produced by DAD, E2E and MIXER on the three tasks. The relative gain is computed as the ratio between the score of a model over the score of the reference XENT model on the same task. The horizontal line indicates the performance of XENT.

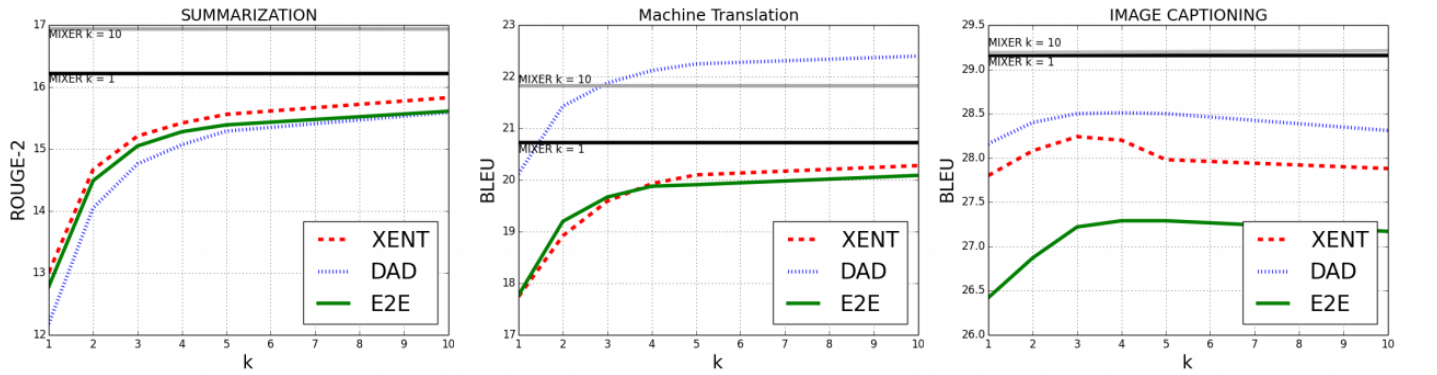


Figure 6: Test score (ROUGE for summarization and BLEU for machine translation and image captioning) as a function of the number of hypotheses k in the beam search. Beam search always improves performance, although the amount depends on the task. The dark line shows the performance of MIXER using greedy generation, while the gray line shows MIXER using beam search with $k = 10$.

Combinatorial Optimization Problems

Classic COP problems

Integer Programming
Maximum Weight Matching (MWM)
Boolean Satisfiability (SAT)
Maximum Clique (MC)
Maximum Independent Set (MIS)
Minimum Vertex Cover (MVC)
Maximum Cut(MaxCut)
Traveling Salesman Problem(TSP)
Knapsack Problem

Supervised learning, where a mapping from training inputs to outputs is learned, supervised learning is not applicable to most combinatorial optimization problems because it does not have access to optimal labels

Traveling salesman problem (TSP)

$state: (i, V)\ now\ at\ i\ passed\ cities\ V$

$system\ control: (i, V) \rightarrow (j, V \cup \{ j \})$

$state\ cost:\ d(i, j)\ distance\ from\ city\ i\ to\ city\ j$

$value\ function:\ J_k^*(i, V)\ the\ minimum\ cost\ of\ \ starting\ from\ the\ source\ point,$
$passing\ through\ the\ city\ set\ V, and\ finally\ staying\ at\ city\ x.$

$Bellman\ equation: J_0^*(s, \{ s \}) = 0$
$J_k^*(i, V \cup \{ i \}) = min_{j \in V} \{ d(i, j)\ +\ J_k^*(j, V) \}$

$time\ complexity\ O(n^2 2^n)$

Bellman equation is deceptive. the solution at each point is affected by the solutions at every other point in the space

南京航空航天大學

Nanjing University of Aeronautics and Astronautics

Pointer Networks

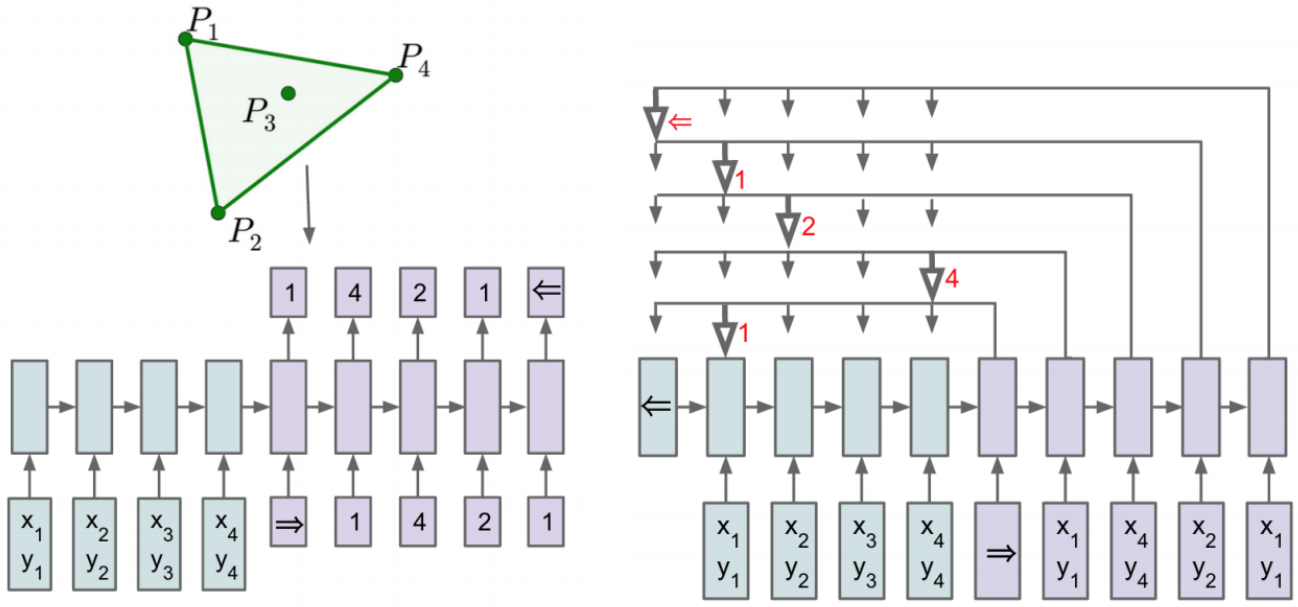
Oriol Vinyals*
Google Brain

Meire Fortunato*
Department of Mathematics, UC Berkeley

Navdeep Jaitly
Google Brain

NIPS 2015

pointer-network



(a) Sequence-to-Sequence

(b) Ptr-Net

Figure 1: **(a)** *Sequence-to-Sequence* - An RNN (blue) processes the input sequence to create a code vector that is used to generate the output sequence (purple) using the probability chain rule and another RNN. The output dimensionality is fixed by the dimensionality of the problem and it is the same during training and inference [1]. **(b)** *Ptr-Net* - An encoding RNN converts the input sequence to a code (blue) that is fed to the generating network (purple). At each step, the generating network produces a vector that modulates a content-based attention mechanism over inputs ([5, 2]). The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input.

Content Based Input Attention

encoder and decoder hidden states as $(e_1, \ldots, e_n)$ and $(d_1, \ldots, d_{m(\mathcal{P})})$

$$
\begin{aligned}
u_j^i &= v^T \tanh(W_1 e_j + W_2 d_i) \quad j \in (1, \ldots, n) \\
a_j^i &= \text{softmax}(u_j^i) \quad j \in (1, \ldots, n) \\
d_i' &= \sum_{j=1}^{n} a_j^i e_j
\end{aligned}
$$

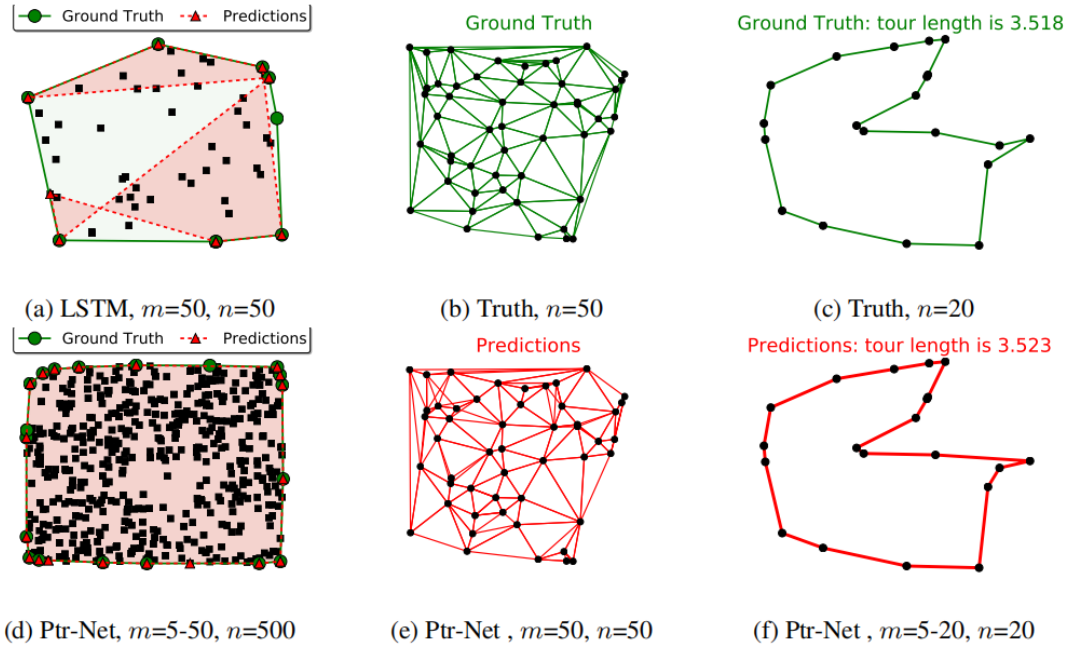
$$
u_j^i = v^T \tanh(W_1 e_j + W_2 d_i) \quad j \in (1, \ldots, n)
$$

$$
p(C_i | C_1, \ldots, C_{i-1}, \mathcal{P}) = \text{softmax}(u^i) \qquad \mathcal{P} = \{P_1, \ldots, P_n\}
$$

experiment of ptr-net

Table 1: Comparison between LSTM, LSTM with attention, and our Ptr-Net model on the convex hull problem. Note that the baselines must be trained on the same n that they are tested on.

METHOD	TRAINED n	n	ACCURACY	AREA
LSTM [1]	50	50	1.9%	FAIL
+ATTENTION [5]	50	50	38.9%	99.7%
PTR-NET	50	50	72.6%	99.9%
LSTM [1]	5	5	87.7%	99.6%
PTR-NET	5-50	5	92.0%	99.6%
LSTM [1]	10	10	29.9%	FAIL
PTR-NET	5-50	10	87.0%	99.8%
PTR-NET	5-50	50	69.6%	99.9%
PTR-NET	5-50	100	50.3%	99.9%
PTR-NET	5-50	200	22.1%	99.9%
PTR-NET	5-50	500	1.3%	99.2%

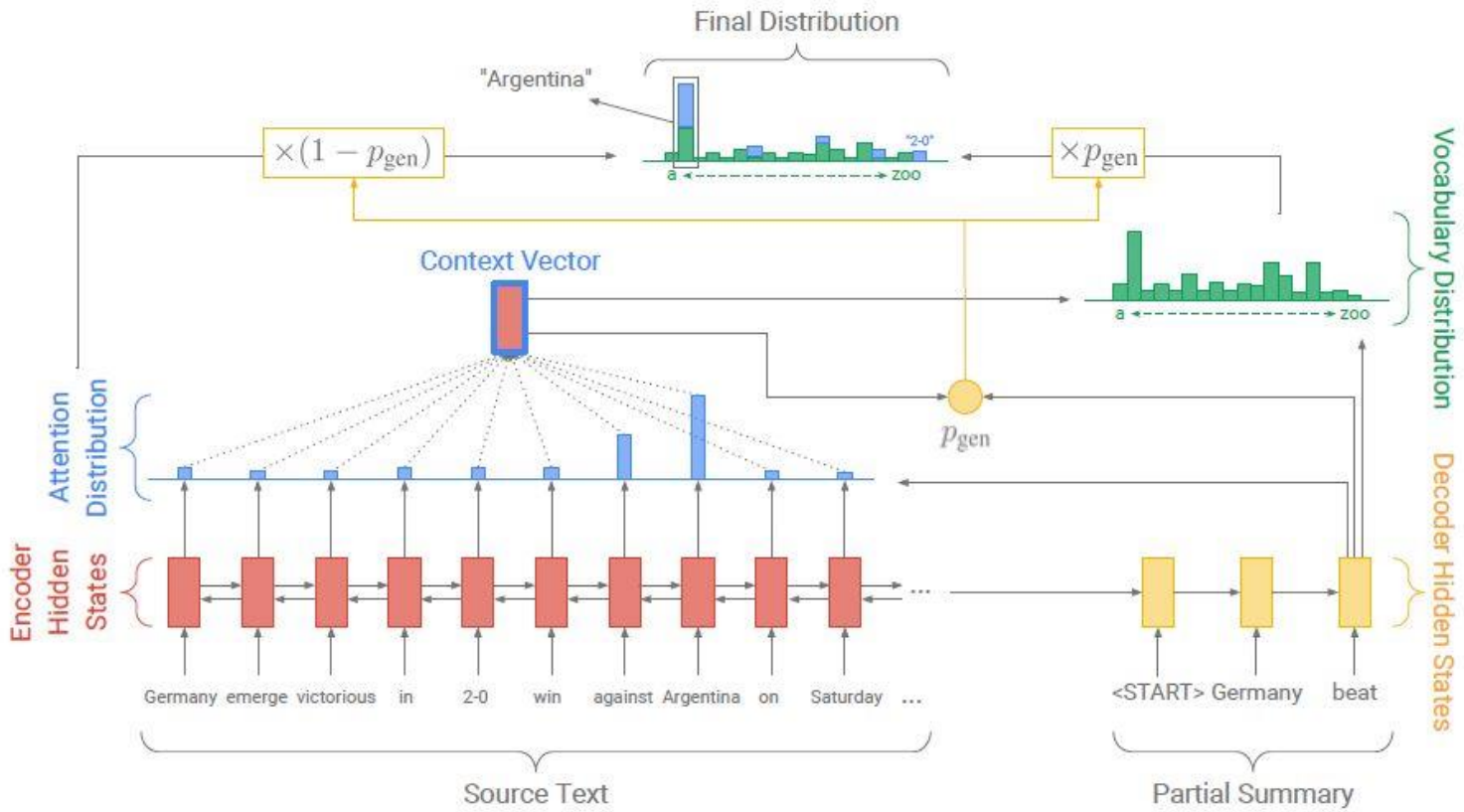


(a) LSTM, m=50, n=50 (b) Truth, n=50 (c) Truth, n=20

(d) Ptr-Net, m=5-50, n=500 (e) Ptr-Net , m=50, n=50 (f) Ptr-Net , m=5-20, n=20

Figure 3: Examples of our model on Convex hulls (left), Delaunay (center) and TSP (right), trained on m points, and tested on n points. A failure of the LSTM sequence-to-sequence model for Convex hulls is shown in (a). Note that the baselines cannot be applied to a different length from training. Using the Ptr-Net model for $n = 5$, we obtained an accuracy of 80.7% and triangle coverage of 93.0%. For $n = 10$, the accuracy was 22.6% and the triangle coverage 81.3%. For $n = 50$, we did not produce any precisely correct triangulation, but obtained 52.8% triangle coverage. See the middle column of Figure 3 for an example for $n = 50$.

Pointer-Generator（2017 ACL)



$$p_{\text{gen}} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{\text{ptr}}) \quad (8)$$

$$P(w) = p_{\text{gen}} P_{\text{vocab}}(w) + (1 - p_{\text{gen}}) \sum_{i:w_i=w} a_i^t \quad (9)$$

南京航空航天大學

Nanjing University of Aeronautics and Astronautics

NEURAL COMBINATORIAL OPTIMIZATION WITH REINFORCEMENT LEARNING

Irwan Bello*, **Hieu Pham***, **Quoc V. Le, Mohammad Norouzi, Samy Bengio**
Google Brain
{ibello,hyhieu,qvl,mnorouzi,bengio}@google.com

ICLR 2017

OPTIMIZATION WITH POLICY GRADIENTS

Our training objective is the expected tour length which, given an input graph s, is defined as

$$J(\boldsymbol{\theta} \mid s) = \mathbb{E}_{\pi \sim p_\theta(.|s)} L(\pi \mid s) \,. \quad L(\pi \mid s) = \left\|\mathbf{x}_{\pi(n)} - \mathbf{x}_{\pi(1)}\right\|_2 + \sum_{i=1}^{n-1} \left\|\mathbf{x}_{\pi(i)} - \mathbf{x}_{\pi(i+1)}\right\|_2$$

$$\nabla_\theta J(\theta \mid s) = \mathbb{E}_{\pi \sim p_\theta(.|s)} \Big[\big(L(\pi \mid s) - b(s)\big) \nabla_\theta \log p_\theta(\pi \mid s) \Big]$$

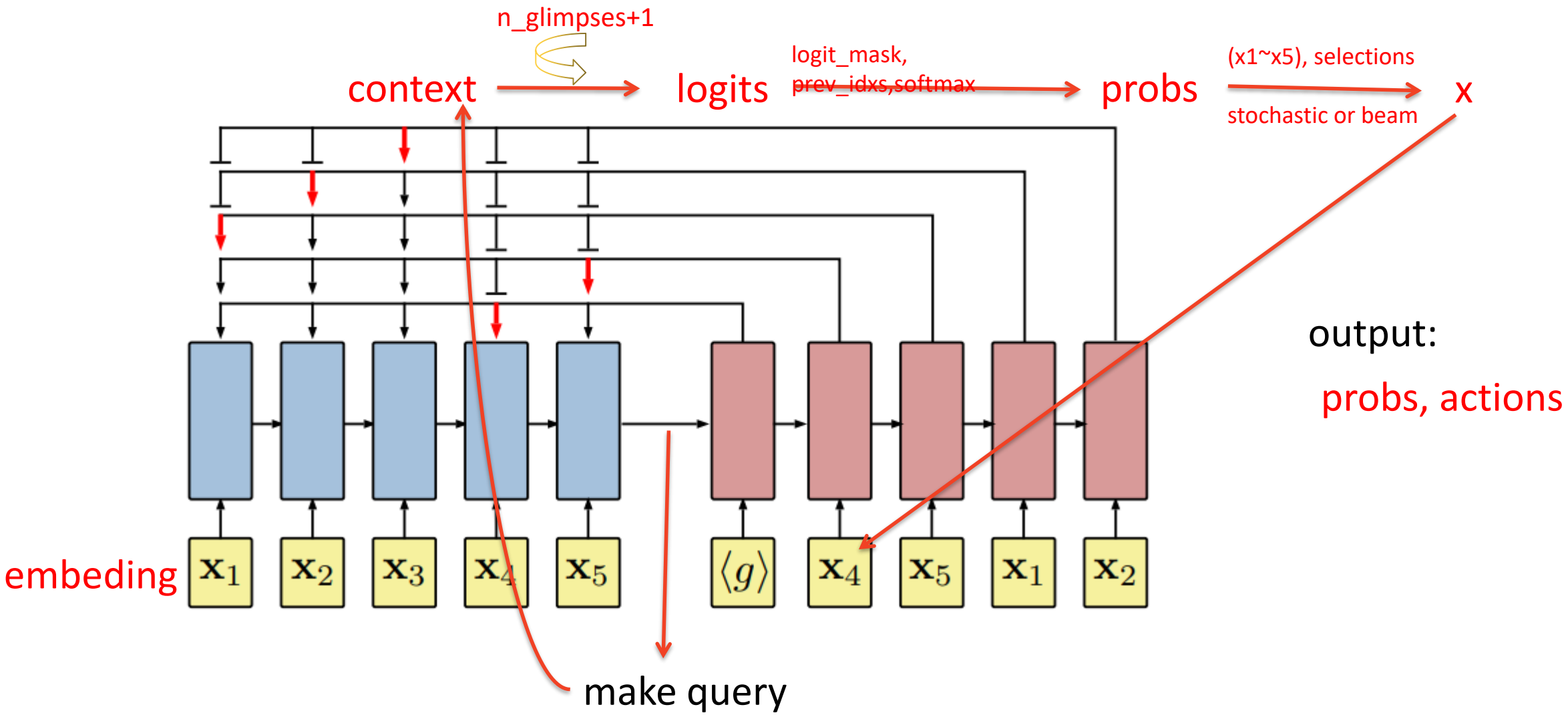
sample graphs $s_1, s_2, \ldots, s_B \sim \mathcal{S}$

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} \Big(L(\pi_i|s_i) - b(s_i) \Big) \nabla_\theta \log p_\theta(\pi_i \mid s_i) \,.$$

critic objective

$$\mathcal{L}(\theta_v) = \frac{1}{B} \sum_{i=1}^{B} \left\| b_{\theta_v}(s_i) - L(\pi_i \mid s_i) \right\|_2^2 .$$

network


n_glimpses+1
context
logits
logit_mask,
prev_idxs,softmax
probs
(x1~x5), selections
stochastic or beam
x
output:
probs, actions
embeding
x_1 x_2 x_3 x_4 x_5 $\langle g \rangle$ x_4 x_5 x_1 x_2
make query

Pointing mechanism: Its computations are parameterized by two attention matrices $W_{ref}, W_q \in \mathbb{R}^{d \times d}$ and an attention vector $v \in \mathbb{R}^d$ as follows:

$$u_i = \begin{cases} v^\top \cdot \tanh\left(W_{ref} \cdot r_i + W_q \cdot q\right) & \text{if } i \neq \pi(j) \text{ for all } j < i \\ -\infty & \text{otherwise} \end{cases} \quad \text{for } i = 1, 2, ..., k \tag{8}$$

$$A(ref, q; W_{ref}, W_q, v) \stackrel{\text{def}}{=} softmax(u). \tag{9}$$

Our pointer network, at decoder step j, then assigns the probability of visiting the next point $\pi(j)$ of the tour as follows:

$$p(\pi(j)|\pi(< j), s) \stackrel{\text{def}}{=} A(enc_{1:n}, dec_j). \tag{10}$$

Setting the logits of cities that already appeared in the tour to $-\infty$, as shown in Equation 8, ensures that our model only points at cities that have yet to be visited and hence outputs valid TSP tours.

Attending mechanism: Specifically, our glimpse function $G(ref, q)$ takes the same inputs as the attention function A and is parameterized by $W^g_{ref}, W^g_q \in \mathbb{R}^{d \times d}$ and $v^g \in \mathbb{R}^d$. It performs the following computations:

$$p = A(ref, q; W^g_{ref}, W^g_q, v^g) \tag{11}$$

$$G(ref, q; W^g_{ref}, W^g_q, v^g) \stackrel{\text{def}}{=} \sum_{i=1}^{k} r_i p_i. \tag{12}$$

The glimpse function G essentially computes a linear combination of the reference vectors weighted by the attention probabilities. It can also be applied multiple times on the same reference set ref:

$$g_0 \stackrel{\text{def}}{=} q \tag{13}$$

$$g_l \stackrel{\text{def}}{=} G(ref, g_{l-1}; W^g_{ref}, W^g_q, v^g) \tag{14}$$

A.2 Improving Exploration

Softmax temperature: We modify Equation 9 as follows:

$$A(ref, q, T; W_{ref}, W_q, v) \stackrel{\text{def}}{=} softmax(u/T), \tag{15}$$

where T is a *temperature* hyperparameter set to $T = 1$ during training. When $T > 1$, the distribution represented by $A(ref, q)$ becomes less steep, hence preventing the model from being overconfident.

Logit clipping: We modify Equation 9 as follows:

$$A(ref, q; W_{ref}, W_q, v) \stackrel{\text{def}}{=} softmax(C \tanh(u)), \tag{16}$$

where C is a hyperparameter that controls the range of the logits and hence the entropy of $A(ref, q)$.

algorithm

Algorithm 1 Actor-critic training

procedure TRAIN(training set S, number of training steps T, batch size B)
    Initialize pointer network params θ
    Initialize critic network params θ_v
    for $t = 1$ to T **do**
        $s_i \sim \text{SAMPLEINPUT}(S)$ for $i \in \{1, \ldots, B\}$
        $\pi_i \sim \text{SAMPLESOLUTION}(p_\theta(.|s_i))$ for $i \in \{1, \ldots, B\}$
        $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, \ldots, B\}$
        $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i|s_i) - b_i) \nabla_\theta \log p_\theta(\pi_i|s_i)$
        $\mathcal{L}_v \leftarrow \frac{1}{B} \sum_{i=1}^{B} \|b_i - L(\pi_i)\|_2^2$
        $\theta \leftarrow \text{ADAM}(\theta, g_\theta)$
        $\theta_v \leftarrow \text{ADAM}(\theta_v, \nabla_{\theta_v} \mathcal{L}_v)$
    end for
    return θ
end procedure

active search

Algorithm 2 Active Search

procedure ACTIVESEARCH(input s, θ, number of candidates K, B, α)
  $\pi \leftarrow$ RANDOMSOLUTION()
  $L_\pi \leftarrow L(\pi \mid s)$
  $n \leftarrow \lceil \frac{K}{B} \rceil$
  for $t = 1 \ldots n$ **do**
    $\pi_i \sim$ SAMPLESOLUTION$(p_\theta(. \mid s))$ for $i \in \{1, \ldots, B\}$
    $j \leftarrow$ ARGMIN$(L(\pi_1 \mid s) \ldots L(\pi_B \mid s))$
    $L_j \leftarrow L(\pi_j \mid s)$
    if $L_j < L_\pi$ **then**
      $\pi \leftarrow \pi_j$
      $L_\pi \leftarrow L_j$
    end if
    $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i \mid s) - b) \nabla_\theta \log p_\theta(\pi_i \mid s)$
    $\theta \leftarrow$ ADAM(θ, g_θ)
    $b \leftarrow \alpha \times b + (1 - \alpha) \times (\frac{1}{B} \sum_{i=1}^{B} b_i)$
  end for
  return π
end procedure

experiment

Table 1: Different learning configurations.

Configuration	Learn on training data	Sampling on test set	Refining on test set
RL pretraining-Greedy	Yes	No	No
Active Search (AS)	No	Yes	Yes
RL pretraining-Sampling	Yes	Yes	No
RL pretraining-Active Search	Yes	Yes	Yes

Table 2: Average tour lengths (lower is better). Results marked $^{(\dagger)}$ are from (Vinyals et al., 2015b).

Task	Supervised Learning	RL pretraining				AS	Christo -fides	OR Tools' local search	Optimal
		greedy	greedy@16	sampling	AS				
TSP20	$3.88^{(\dagger)}$	3.89	–	3.82	3.82	3.96	4.30	3.85	3.82
TSP50	$6.09^{(\dagger)}$	5.95	5.80	5.70	5.70	5.87	6.62	5.80	5.68
TSP100	10.81	8.30	7.97	7.88	7.83	8.19	9.18	7.99	7.77

Table 3: Running times in seconds (s) of greedy methods compared to OR Tool's local search and solvers that find the optimal solutions. Time is measured over the entire test set and averaged.

Task	RL pretraining		OR-Tools' local search	Optimal	
	greedy	greedy@16		Concorde	LK-H
TSP50	0.003s	0.04s	0.02s	0.05s	0.14s
TSP100	0.01s	0.15s	0.10s	0.22s	0.88s

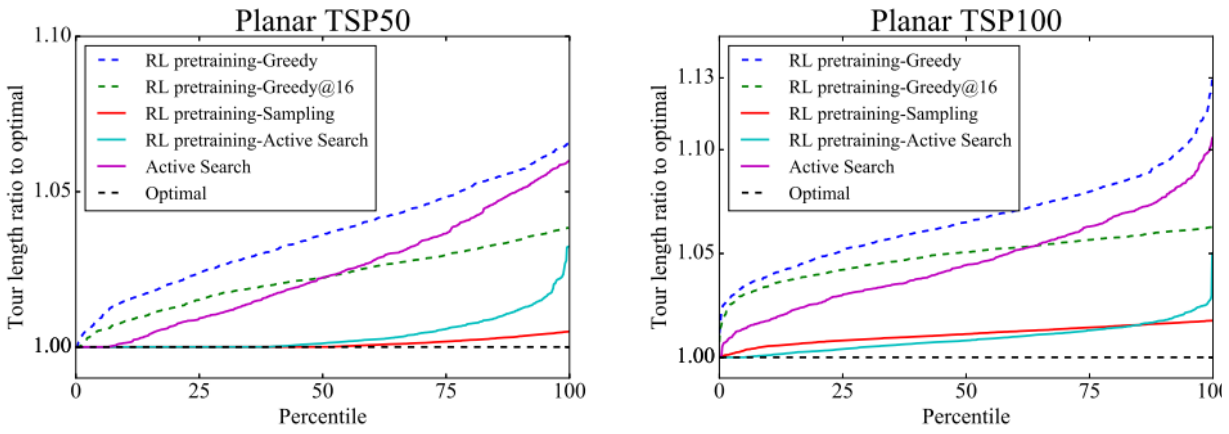
experiment

Table 4: Average tour lengths of RL pretraining-Sampling and RL pretraining-Active Search as they sampl more solutions. Corresponding running times on a single Tesla K80 GPU are in parantheses.

Task	# Solutions	RL pretraining		
		Sampling $T = 1$	Sampling $T = T^*$	Active Search
TSP50	128	5.80 (3.4s)	5.80 (3.4s)	5.80 (0.5s)
	1,280	5.77 (3.4s)	5.75 (3.4s)	5.76 (5s)
	12,800	5.75 (13.8s)	5.73 (13.8s)	5.74 (50s)
	128,000	5.73 (110s)	5.71 (110s)	5.72 (500s)
	1,280,000	5.72 (1080s)	5.70 (1080s)	5.70 (5000s)
TSP100	128	8.05 (10.3s)	8.09 (10.3s)	8.04 (1.2s)
	1,280	8.00 (10.3s)	8.00 (10.3s)	7.98 (12s)
	12,800	7.95 (31s)	7.95 (31s)	7.92 (120s)
	128,000	7.92 (265s)	7.91 (265s)	7.87 (1200s)
	1,280,000	7.89 (2640s)	7.88 (2640s)	7.83 (12000s)

Table 5: Results of RL pretraining-Greedy and Active Search on KnapSack (higher is better).

Task	RL pretraining greedy	Active Search	Random Search	Greedy	Optimal
KNAP50	19.86	**20.07**	17.91	19.24	**20.07**
KNAP100	40.27	**40.50**	33.23	38.53	**40.50**
KNAP200	57.10	**57.45**	35.95	55.42	**57.45**



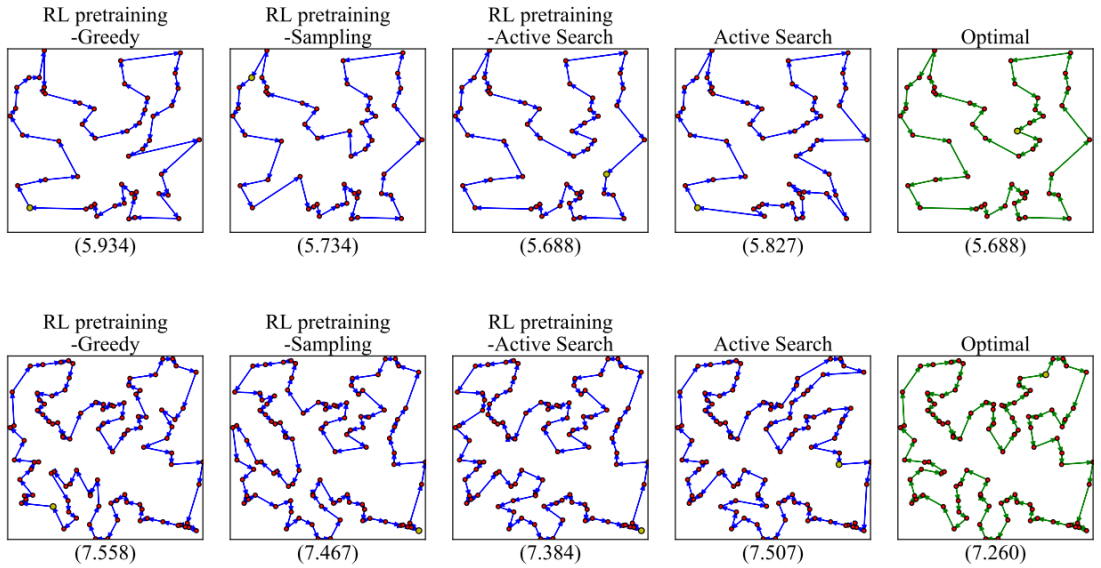
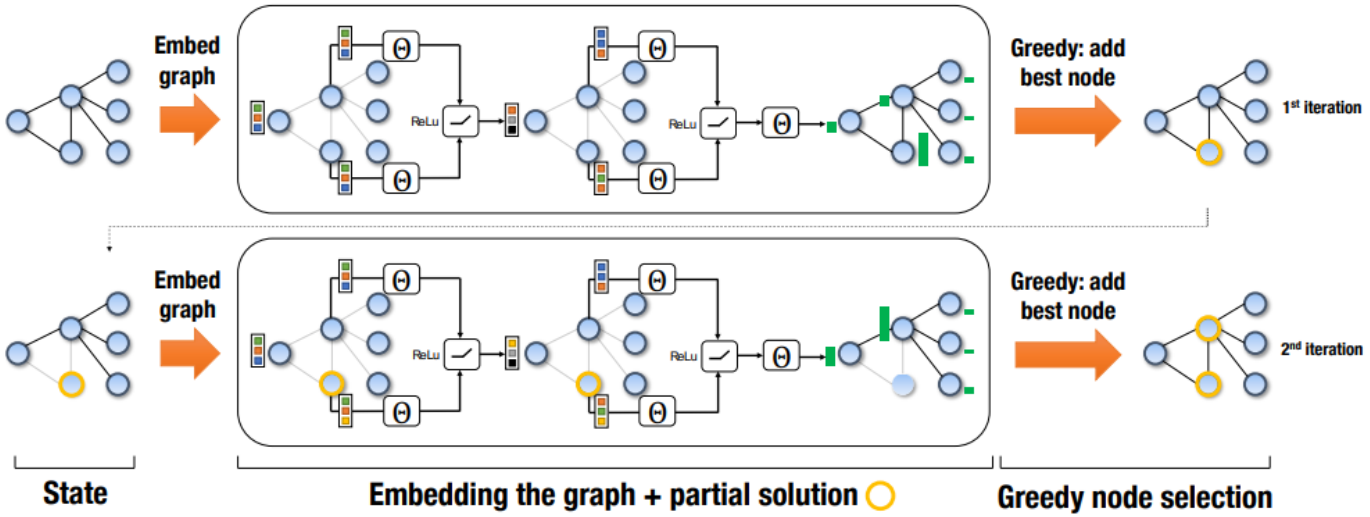


Figure 2: Sorted tour length ratios to optimality

Learning Combinatorial Optimization Algorithms over Graphs(NeurIPS, 2017)



Structure2Vec $$\mu_v^{(t+1)} \leftarrow F\left(x_v, \{\mu_u^{(t)}\}_{u\in\mathcal{N}(v)}, \{w(v,u)\}_{u\in\mathcal{N}(v)}; \Theta\right),$$

$$\mu_v^{(t+1)} \leftarrow \mathrm{relu}\left(\theta_1 x_v + \theta_2 \sum\nolimits_{u\in\mathcal{N}(v)} \mu_u^{(t)} + \theta_3 \sum\nolimits_{u\in\mathcal{N}(v)} \mathrm{relu}(\theta_4\, w(v,u))\right)$$

$$\widehat{Q}(h(S), v; \Theta) = \theta_5^{\top}\, \mathrm{relu}([\theta_6 \sum\nolimits_{u\in V} \mu_u^{(T)}, \theta_7\, \mu_v^{(T)}])$$

ATTENTION, LEARN TO SOLVE ROUTING PROBLEMS! （ICLR 2019）

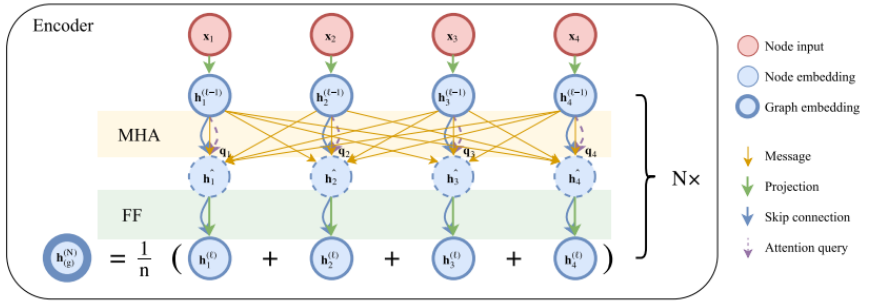


Figure 1: Attention based encoder. Input nodes are embedded and processed by N sequential layers, each consisting of a multi-head attention (MHA) and node-wise feed-forward (FF) sub-layer. The graph embedding is computed as the mean of node embeddings. Best viewed in color.

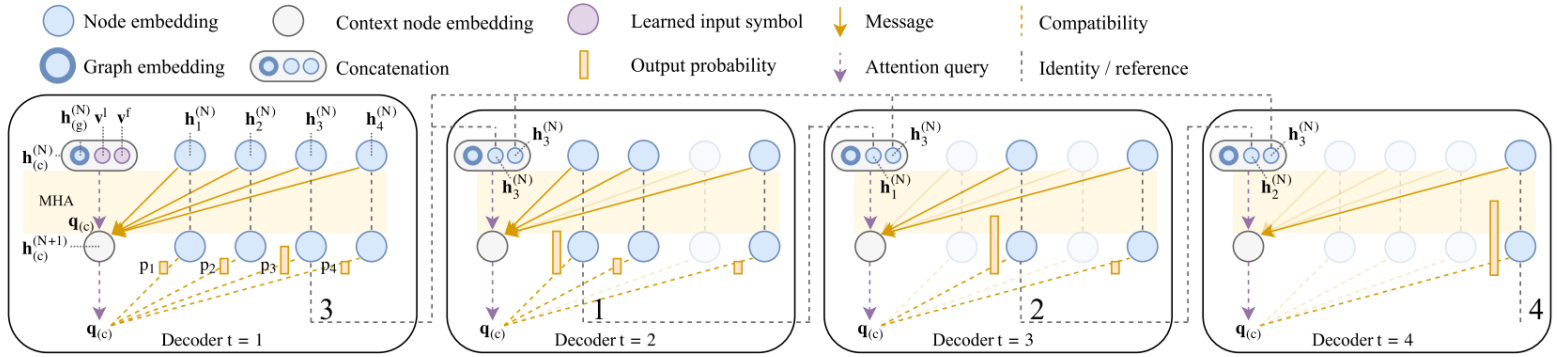


Figure 2: Attention based decoder for the TSP problem. The decoder takes as input the graph embedding and node embeddings. At each time step t, the context consist of the graph embedding and the embeddings of the first and last (previously output) node of the partial tour, where learned placeholders are used if $t = 1$. Nodes that cannot be visited (since they are already visited) are masked. The example shows how a tour $\boldsymbol{\pi} = (3, 1, 2, 4)$ is constructed. Best viewed in color.

Context embedding

$$\mathbf{h}_{(c)}^{(N)} = \begin{cases} \left[\bar{\mathbf{h}}^{(N)}, \mathbf{h}_{\pi_{t-1}}^{(N)}, \mathbf{h}_{\pi_1}^{(N)}\right] & t > 1 \\ \left[\bar{\mathbf{h}}^{(N)}, \mathbf{v}^{\mathrm{l}}, \mathbf{v}^{\mathrm{f}}\right] & t = 1. \end{cases}$$

$$\mathbf{q}_{(c)} = W^Q \mathbf{h}_{(c)} \quad \mathbf{k}_i = W^K \mathbf{h}_i, \quad \mathbf{v}_i = W^V \mathbf{h}_i.$$

$$u_{(c)j} = \begin{cases} C \cdot \tanh\left(\frac{\mathbf{q}_{(c)}^T \mathbf{k}_j}{\sqrt{d_k}}\right) & \text{if } j \neq \pi_{t'} \quad \forall t' < t \\ -\infty & \text{otherwise.} \end{cases}$$

$$p_i = p_{\boldsymbol{\theta}}(\pi_t = i | s, \boldsymbol{\pi}_{1:t-1}) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}}.$$