Statistical Learning for Text Data Analytics
Neural Text Generation with Reinforcement Learning

Yangqiu Song

Hong Kong University of Science and Technology

yqsong@cse.ust.hk

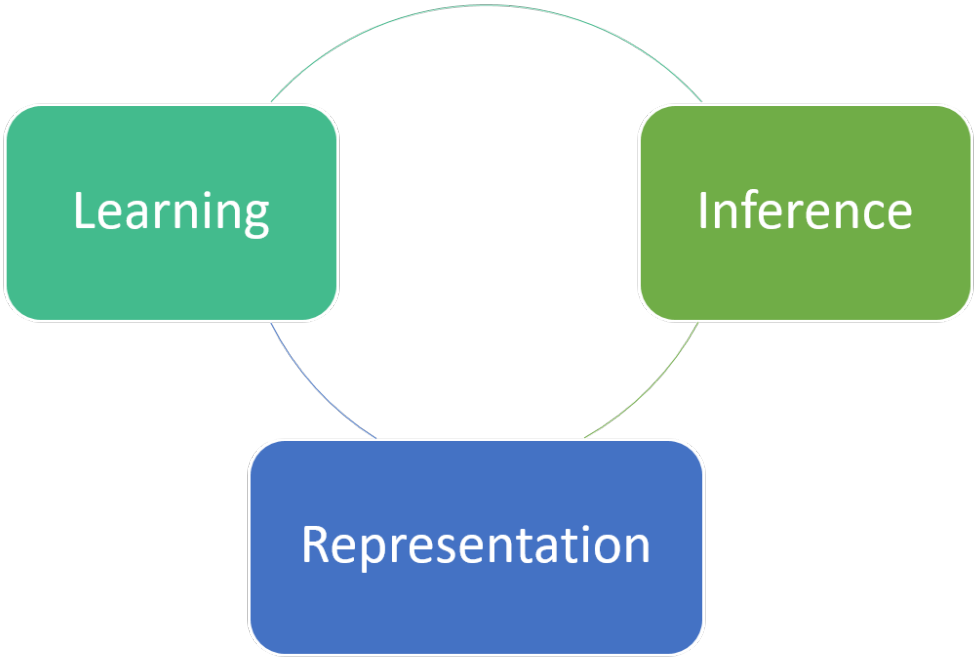
Spring 2018

∗Contents are based on materials created by Noah Smith, Andrej Karpathy, Fei-Fei Li, Thang Luong, Kyunghyun Cho, Christopher Manning, Liang Huang, John Schulman, Sergey Levine

Reference Content

- Noah Smith. CSE 517: Natural Language Processing https://courses.cs.washington.edu/courses/cse517/16wi/
- Andrej Karpathy and Fei-Fei Li. SF ML meetup: Automated Image Captioning with ConvNets and Recurrent Nets. https://cs.stanford.edu/people/karpathy/sfmltalk.pdf
- Thang Luong, Kyunghyun Cho, and Christopher Manning. ACL 2016 tutorial on Neural Machine Translation. https://sites.google.com/site/acl16nmt/
- Liang Huang. EMNLP 2017 Structured Prediction Workshop, Marrying Dynamic Programming with Recurrent Neural Networks. http://web.engr.oregonstate.edu/~huanlian/slides/DP-RNN-EMNLP.pdf
- John Schulman and Sergey Levine. CS 294: Deep Reinforcement Learning. http://rll.berkeley.edu/deeprlcoursesp17/

Course Topics



- Representation: language models, word embeddings, topic models
- Learning: supervised learning, semi-supervised learning, sequence models, deep learning, optimization techniques
- Inference: constrained modeling, joint inference, search algorithms

NLP applications: tasks introduced in Lecture 1

Overview

1. Introduction
2. Maximum Likelihood Based Approach
3. Reinforcement Learning
4. Parameter Estimation: Policy Gradient
5. Evaluation

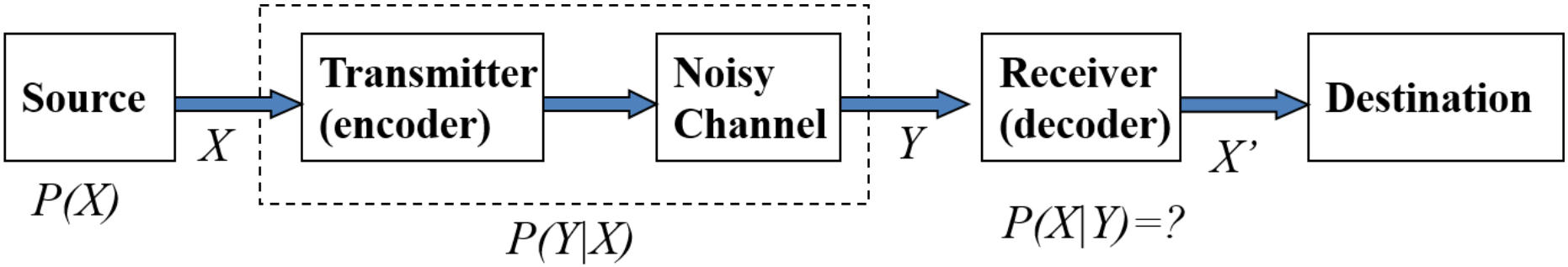
Text Generation

Example (Generated from language models of New York Times)

- Unigram
  - Months the my and issue of year foreign new exchanges september were recession exchange new endorsed a q acquire to six executives.
- Bigram
  - Last December through the way to preserve the Hudson corporation N.B.E.C. Taylor would seem to complete the major central planners one point five percent of U.S.E. has already told M.X. corporation of living on information such as more frequently fishing to keep her.
- Trigram
  - They also point to ninety nine point six billon dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions.

Without contexts, text generation cannot make any sense!

Source-Channel Framework [Shannon '48]



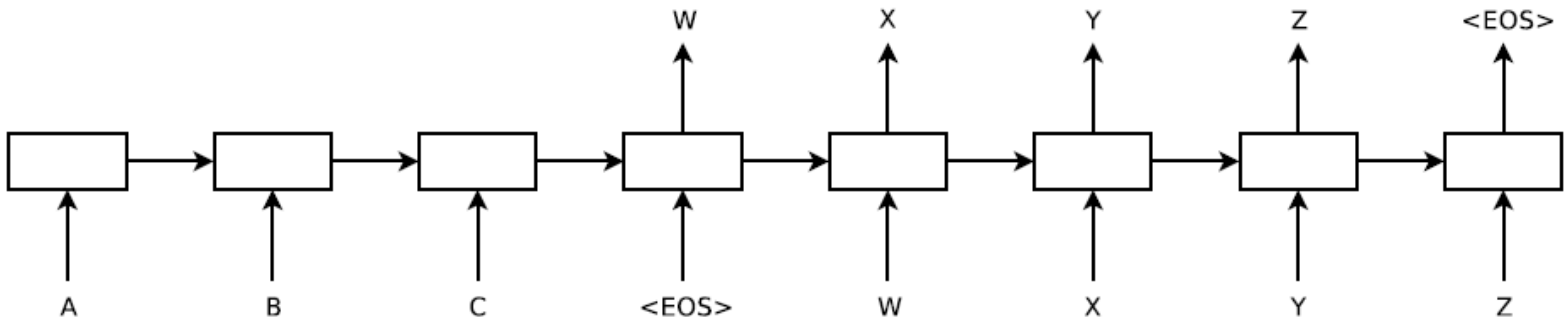
$$\hat{X} = \arg\max_X P(X|Y) = \arg\max_X P(Y|X)P(X) \text{ (Bayes Rule)}$$

When X is text, $P(X)$ is a language model

	X	Y
Speech recognition	Word sequence	Speech signal
Machine translation	English sentence	Chinese sentence
Question answering	Answer	Question
Document summarization	Summary	Document
Image Captioning	Caption	Image

Neural Language Generation

- High level idea (by skipping a lot of important classical papers in literature!)
  - Encoding and decoding
  - Take care: here, the terminology "encoder" and "decoder" are used differently than in the noisy-channel pattern
- Neural machine translation
  - Original idea of neural machine translation was proposed by Forcada and Ñeco (1997); resurgence in interest starting around 2013
  - Sequence to sequence learning (Sutskever et al. (2014))



  - Strong starting point using attention (Bahdanau et al. (2014))

Neural Language Generation (Cont'd)

- Neural image captioning (Donahue et al. (2014); Vinyals et al. (2015))
  - Encode image
  - Decode text

a cat is sitting on a toilet seat
logprob: -7.79

a display case filled with lots of different types of donuts
logprob: -7.78

a group of people sitting at a table with wine glasses
logprob: -6.71

High Level Model

$$
\begin{aligned}
P(W = \text{output} | I = input) &= P(W = \text{output} | \textbf{encode}(input)) \\
&= \prod_{i=1}^{N} P(w_i | w_1, \ldots, w_{i-1}, \textbf{encode}(input))
\end{aligned}
$$

The encoding of the source sentence is a deterministic function of input.

Building Block: Recurrent Neural Network

- Each input element is understood to be an element of a sequence: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$
- At each timestep t:
  - The tth input element $\mathbf{x}_t$ is processed alongside the previous state $\mathbf{s}_{t-1}$ to calculate the new state $\mathbf{s}_t$
  - The tth output is a function of the state $\mathbf{s}_t$
  - The same functions are applied at each iteration:

$$\mathbf{s}_t = f_{\text{recurrent}}(\mathbf{x}_t, \mathbf{s}_{t-1})$$

$$\mathbf{y}_t = f_{\text{output}}(\mathbf{s}_t)$$

RNN Language Model

- The original version, by Mikolov et al. (2010) used a "simple" RNN architecture along these lines:

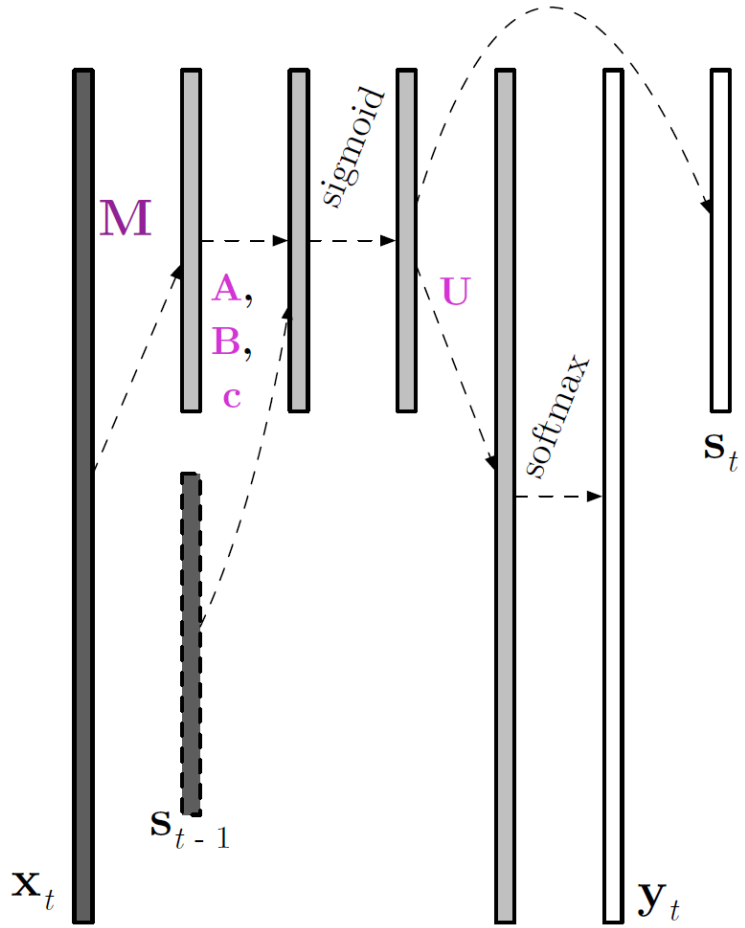
$$\mathbf{s}_t = f_{\text{recurrent}}(\mathbf{e}_{x_t}, \mathbf{s}_{t-1}) =$$

$$\text{sigmoid}\left(\left(\mathbf{e}_{x_t}^\top \mathbf{M}\right)^\top \mathbf{A} + \mathbf{s}_{t-1}^\top \mathbf{B} + \mathbf{c}\right)$$

$$\mathbf{y}_t = f_{\text{output}}(\mathbf{s}_t) = \text{softmax}(\mathbf{s}_t^\top \mathbf{U})$$

$$P(v|w_1, \ldots, w_{n-1}) = [\mathbf{y}_t]_v$$

- Note: this is not an n-gram (Markov) model!

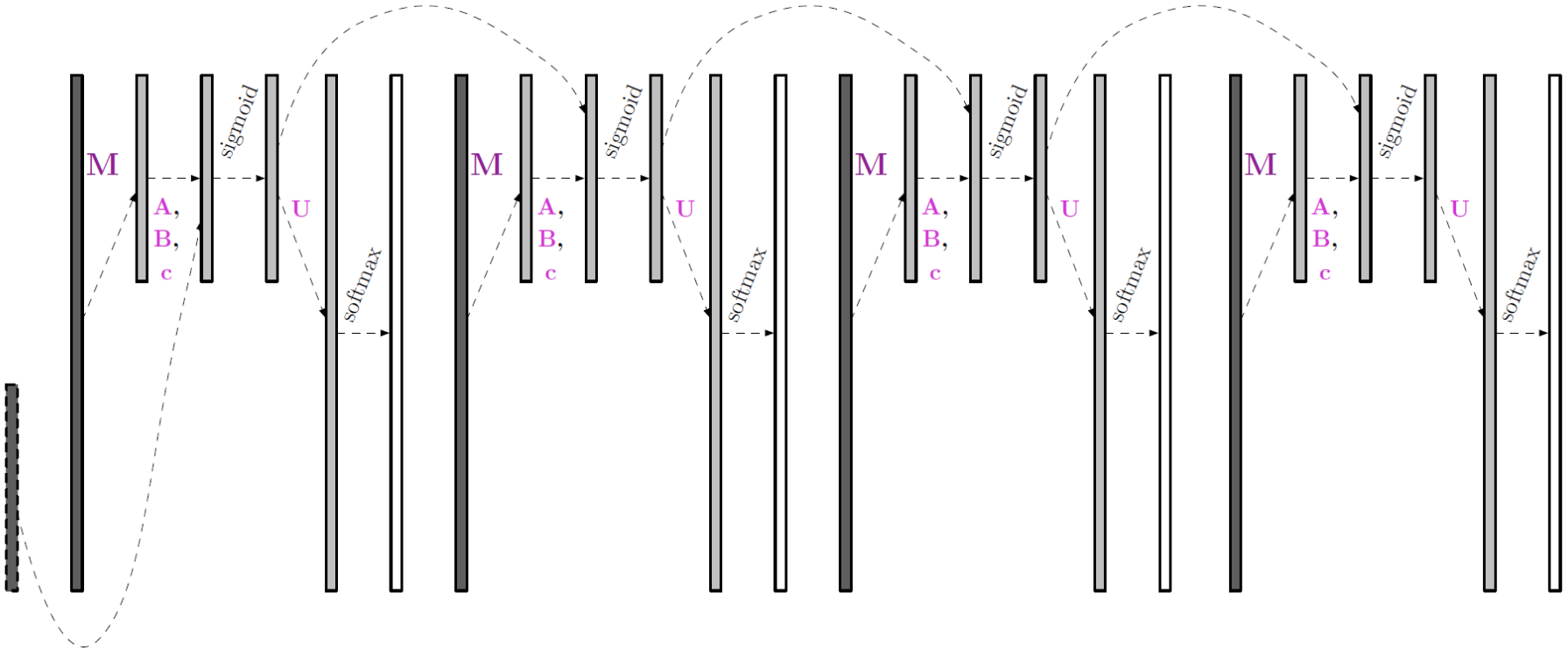


Overview

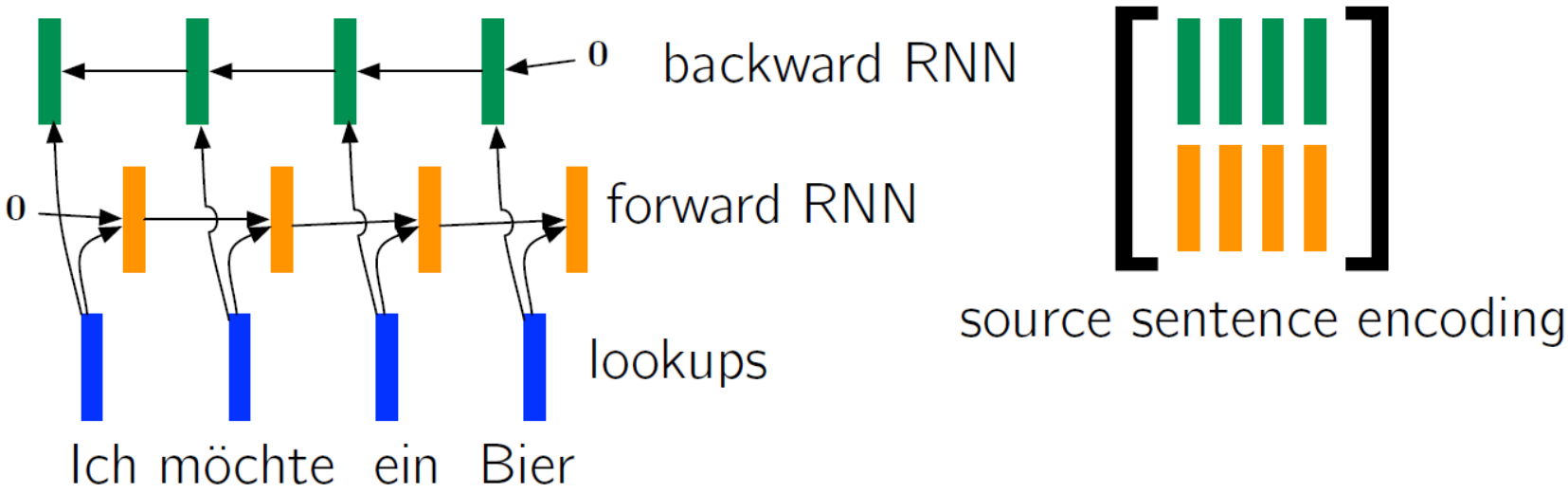
1. Introduction
2. Maximum Likelihood Based Approach
3. Reinforcement Learning
4. Parameter Estimation: Policy Gradient
5. Evaluation

Maximum Likelihood (Neural Language Model)

$$P(\mathcal{W}) = \prod_{t=1}^{N} P(w_t | w_1, w_2, \ldots, w_{t-1})$$



Machine Translation: Encoder



If we have m words in the source language, we encode them into $2d \times m$ matrix $\mathbf{E}_{in}$

Attention (Bahdanau et al. (2014))

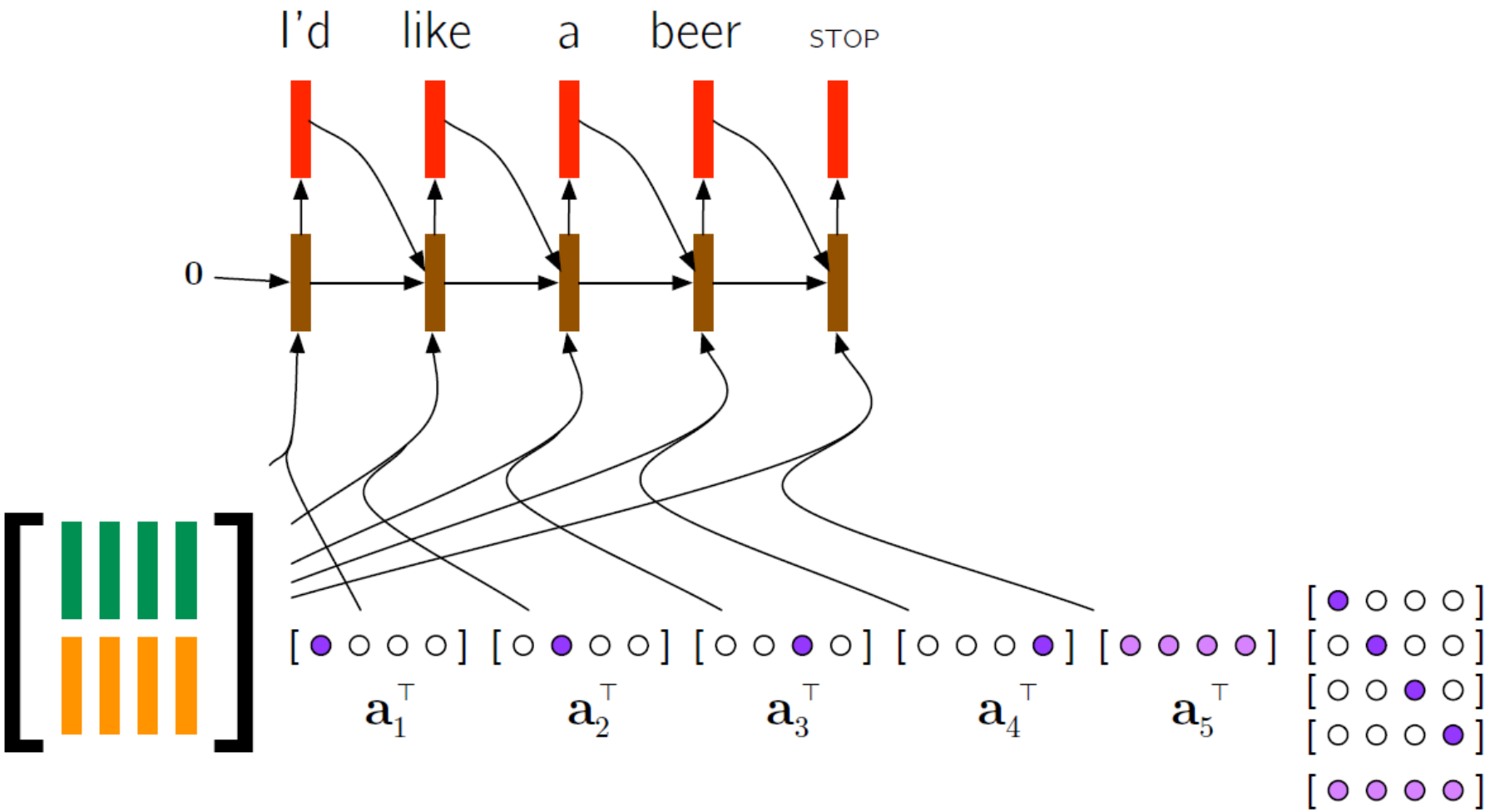
- Let $\mathbf{V}\mathbf{s}_{t-1}$ be the "expected" input embedding for timestep t ($\mathbf{V}$ are the parameters)
  - to make $\mathbf{s}_{t-1}$ and $\mathbf{E}_{in}$ compatible
- Attention to the historical words is computed as $\mathbf{a}_t = \text{softmax}(\mathbf{E}_{in}^\top \mathbf{V}\mathbf{s}_{t-1})$
- Context $\mathbf{c}_t = \mathbf{E}_{in}\mathbf{a}_t$ is a weighted sum of source words' in-context representations
- Then we have a new recurrent function

$$\mathbf{s}_t = f_{\text{recurrent}}(\mathbf{e}_{x_t}, \mathbf{c}_t, \mathbf{s}_{t-1}) = \text{sigmoid}\left(\left(\mathbf{e}_{x_t}^\top \mathbf{M}\right)^\top \mathbf{A} + \mathbf{s}_{t-1}^\top \mathbf{B} + \mathbf{c}_t^\top \mathbf{C}\right)$$

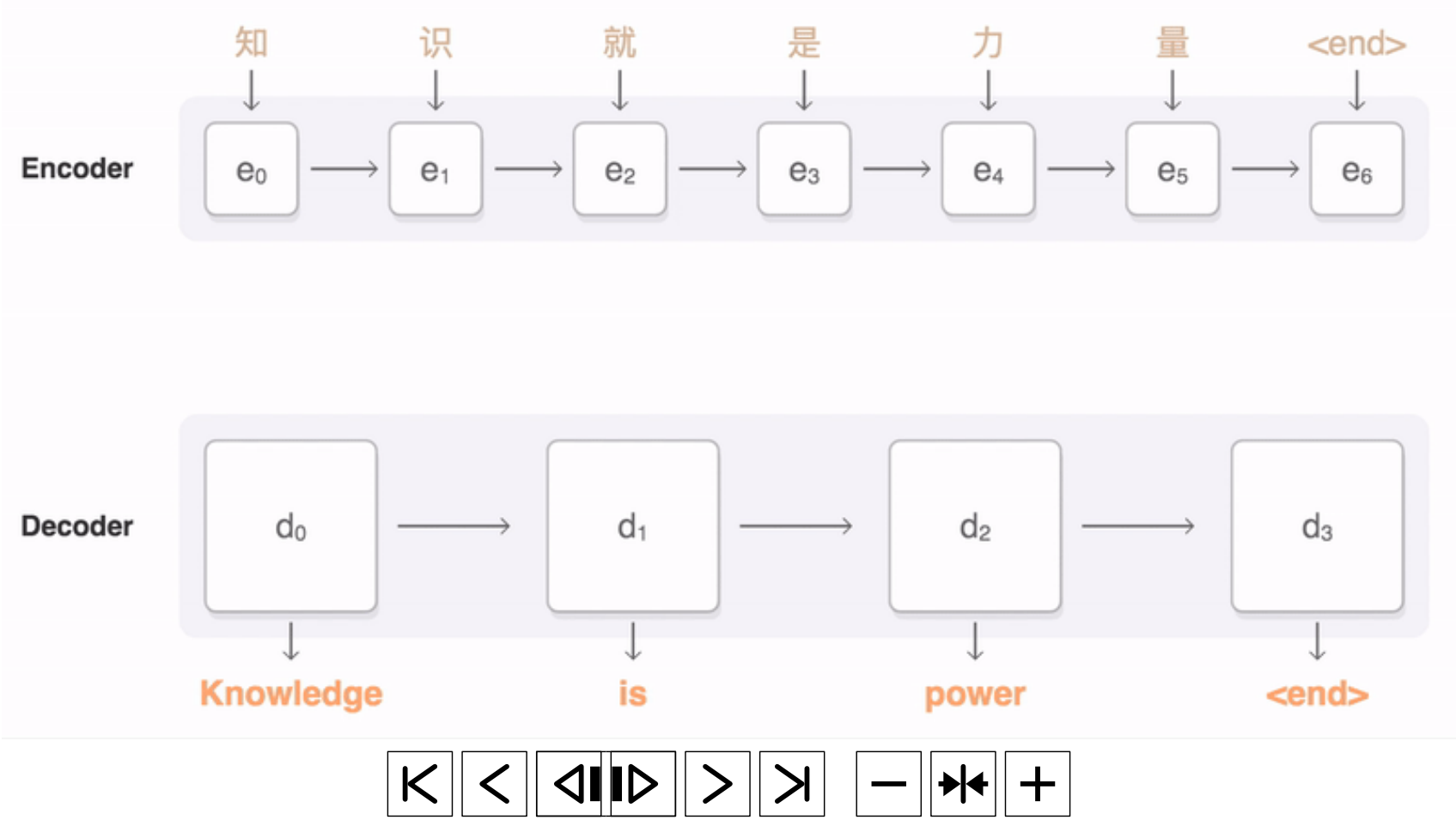
$$\mathbf{y}_t = f_{\text{output}}(\mathbf{s}_t) = \text{softmax}(\mathbf{s}_t^\top \mathbf{U})$$

in decoder

Example of Neural MT Decoder



Example from Google NMT (Wu et al. (2016))



Maximum Likelihood (Neural Machine Translation)

Put all together:

$$\mathbf{c}_t = \mathbf{E}_{in}\mathbf{a}_t, \quad \text{where} \quad \mathbf{a}_t = \text{softmax}(\mathbf{E}_{in}^\top \mathbf{V}\mathbf{s}_{t-1})$$

$$\mathbf{s}_t = f_{\text{recurrent}}(\mathbf{e}_{x_t}, \mathbf{c}_t, \mathbf{s}_{t-1}) = \text{sigmoid}\left(\left(\mathbf{e}_{x_t}^\top \mathbf{M}\right)^\top \mathbf{A} + \mathbf{s}_{t-1}^\top \mathbf{B} + \mathbf{c}_t^\top \mathbf{C}\right)$$

$$\mathbf{y}_t = f_{\text{output}}(\mathbf{s}_t) = \text{softmax}(\mathbf{s}_t^\top \mathbf{U})$$

$$P(v|w_1, \ldots, w_{n-1}, \mathbf{E}_{in}) = [\mathbf{y}_t]_v$$

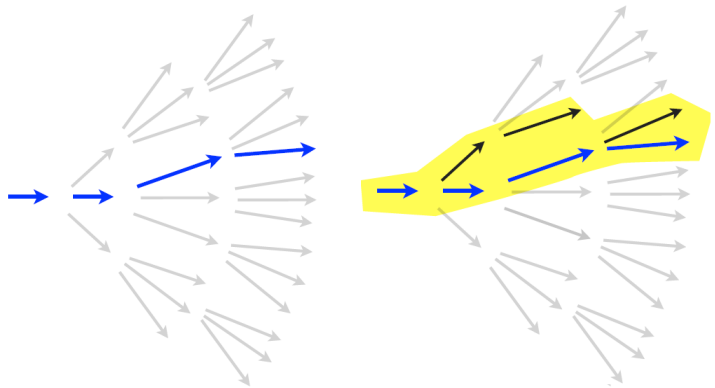
The likelihood for a sequence-to-sequence translation (sentence s) is

$$P(\mathcal{W}) = \prod_{s=1}^{M} \prod_{t=1}^{s_N} P(w_t^s | w_1^s, \ldots, w_{t-1}^s, \mathbf{E}_{in}^s)$$

- This is differentiable with respect to all parameters of the neural network, allowing "end-to-end" training
- Trick: train on shorter sentences first, then add in longer ones
- Decoding typically uses beam search

Beam Search

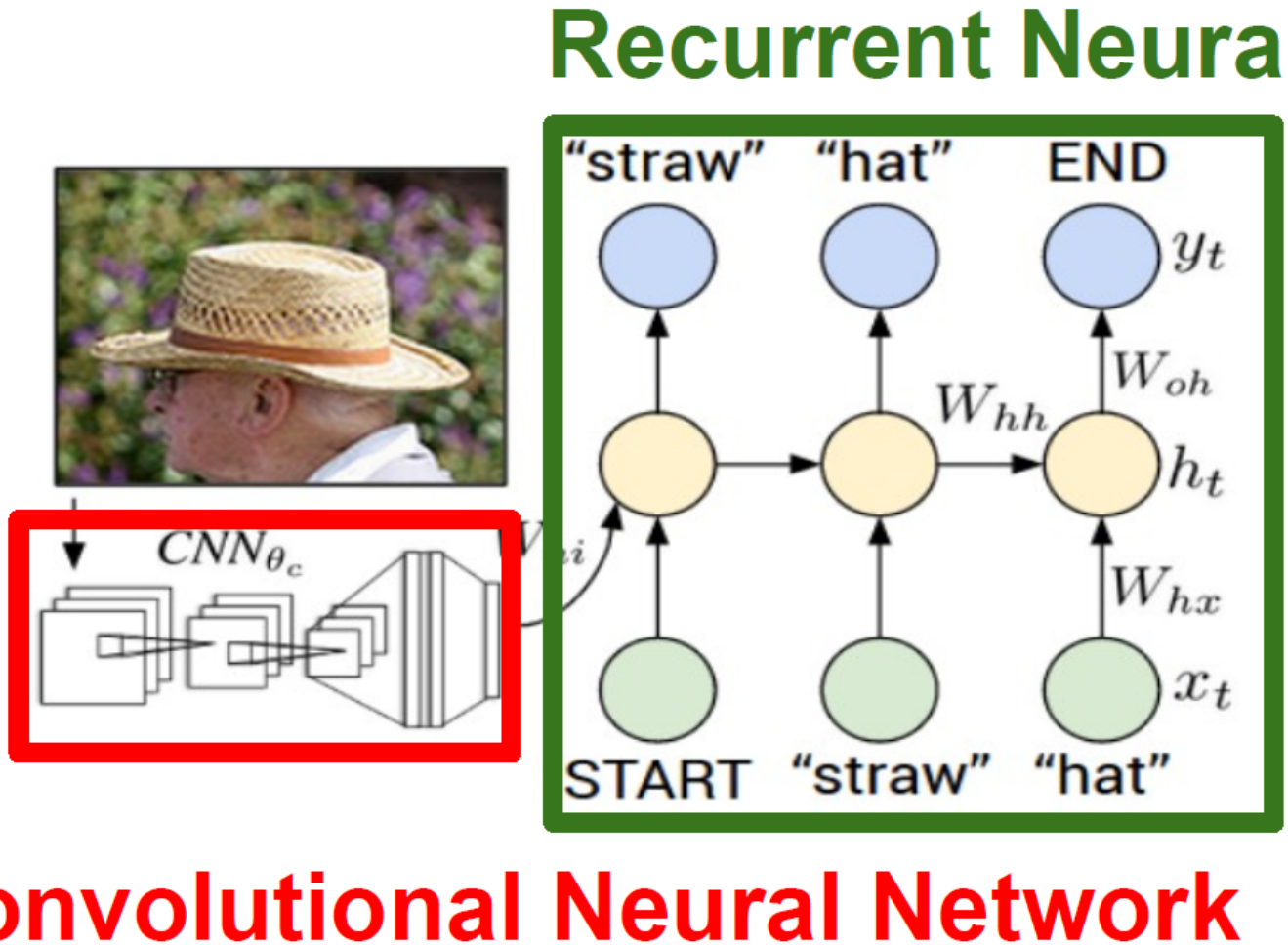
- Beam inference
  - At each position keep the top k complete sequences
  - Extend each sequence in each local way
  - The extensions compete for the k slots at the next position



(a) Greedy (b) Beam Search

- Advantages
  - Fast; beam sizes of 3-5 are almost as good as exact inference in many cases
  - Easy to implement (no dynamic programming required)
- Disadvantage
  - Inexact: the globally best sequence can fall off the beam

Example of Image Captioning Encoder and Decoder



Overview

1. Introduction
2. Maximum Likelihood Based Approach
3. Reinforcement Learning
4. Parameter Estimation: Policy Gradient
5. Evaluation

Problem with Maximum Likelihood Training

Once trained, one can use the model to generate an entire sequence as follows

- Let w_t^g denote the word generated by the model at the t-th time step. Then the next word is generated by:

$$w_{t+1}^g = \arg\max_v P_\theta(v|w_t^g, \mathbf{s}_{t+1})$$

  where we denote $\theta = \{\mathbf{M}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{U}\}$ as the parameters
- However, the model was trained to maximize $\arg\max_v P_\theta(v|w_t, \mathbf{s}_{t+1})$ shown in training data (Ranzato et al. (2016))
  - During training, the model is only exposed to the ground truth words (exposure bias)
  - At test time the model has only access to its own predictions, which may not be correct
  - During generation the model can potentially deviate quite far from the actual sequence to be generated
- This phenomena is commonly known as a search error (compounding error)

Text Generation as a MDP

- A Markov Decision Process (MDP) is defined as
  - A set of states $s \in \mathcal{S}$ of the environment
  - A set of actions $a \in \mathcal{A}$
  - A transition function $T(s, a, s') = P(s'|s, a)$
  - A reward function $R(s, a, s')$
  - A discount factor $\gamma \in [0, 1)$ downweights future rewards
  - A start state
- For MDPs, we want an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$
  - A policy π gives an action for each state

Text Generation as a MDP (Cont'd)

- State $s_t = \{\mathbf{E}_{in}, w_1, \cdots, w_t\}$.
  - The current state is a configuration based on the input encoder and all historical words.
- Action $a_t \leftarrow s_t$.
  - Taking an action means given the current state, we choose to generate a new word, and the action space corresponds to the whole vocabulary.
- Reward $r(s, a, s') = \begin{cases} 0 & \text{if } a \neq \text{END} \\ \beta(s) & \text{otherwise} \end{cases}$.
  - A non-zero reward $\beta(s)$ is only evaluated at the end of a sentence since at intermediate positions we do not know how good the generation is compared to human labeled sentences.
- Transition probability $P_{sa}(s') = P(s'|s, a)$ is the probability of generating a new state based on an action.
  - We set $P_{sa}(s') = I(s' = s \cup \{a\})$ where $I(\cdot)$ is the indicator function. This is because given an selected action, there is no randomness of failure of using the selected words to generate a next state.
- We set discounting factor $\gamma = 1$ if we do not penalize longer text generation.

Problems Involving MDPs

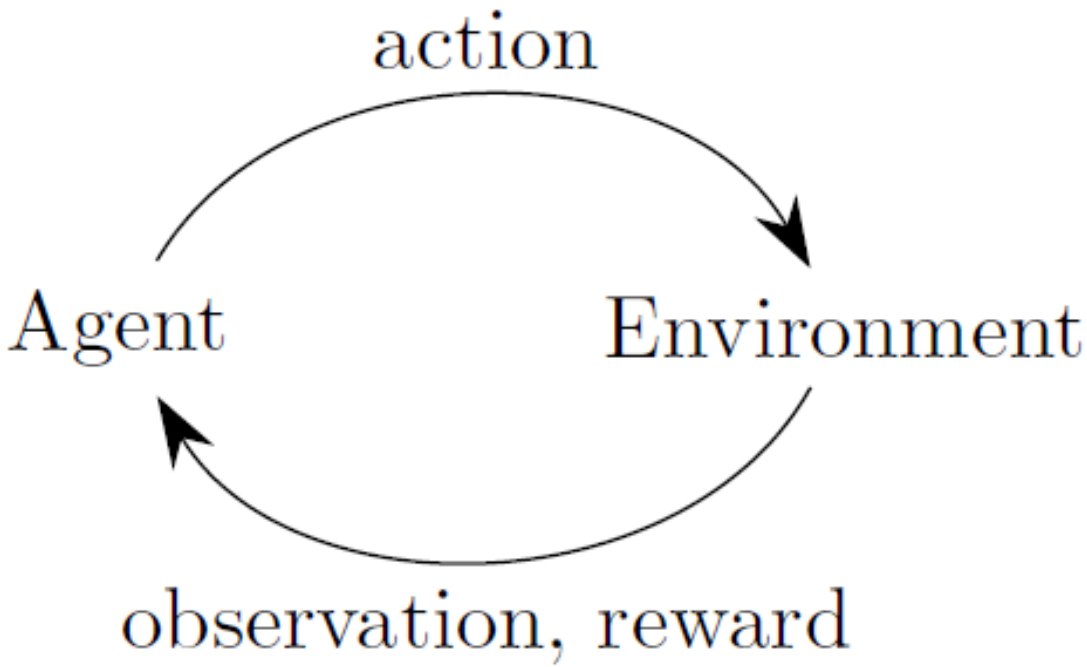
- Polycies
  - Deterministic policies: $a = \pi(s)$
  - Stochastic policies: $a \sim \pi(s)$ (our case)
- Policy optimization: maximize expected reward with respect to policy π

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$$

- Policy evaluation: compute expected return for fixed policy π
  - return := sum of future rewards in an episode (i.e., a trajectory)
    - Discounted return: $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$
    - Undiscounted return: $r_t + r_{t+1} + r_{t+2} + \ldots$
  - Performance of policy: $\eta(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$
  - State value function: $V^{\pi}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$
  - State-action value function: $Q^{\pi}(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a]$

Reinforcement Learning

- Basic idea:
  - Receive feedback in the form of rewards
  - Agents utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!



Episodic Setting

$$s_0 \sim P(s_0)$$
$$a_0 \sim \pi(a_0|s_0),\ s_1 \sim P(s_1|s_0, a_0),\ \text{receive}\ r_1 = r(s_0, a_0, s_1)$$
$$a_1 \sim \pi(a_1|s_1),\ s_2 \sim P(s_2|s_1, a_1),\ \text{receive}\ r_2 = r(s_1, a_1, s_2)$$
$$\dots$$
$$a_{T-1} \sim \pi(a_{T-1}|s_{T-1}),\ s_T \sim P(s_T|s_{T-1}, a_{T-1}),\ \text{receive}$$
$$r_T = r(s_{T-1}, a_{T-1}, s_T)$$

Objective:

$$\max \mathbb{E}_{P(\tau)} \left[r_0 + r_{t+1} + r_{t+2} + \ldots + R_T \right]$$

where

$$P(\tau) = P(s_0, a_0, s_1, a_1, \ldots, s_T, a_T) = P(s_0) \prod_{t=1}^{T-1} P(s_t|s_{t-1}, a_{t-1}) \pi(a_{t-1}|s_{t-1})$$

Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
  - Deterministic $a = \pi(s, \theta)$
  - Stochastic $a \sim \pi(a|s, \theta)$
- Then we have

$$P_\theta(\tau) = P(s_0, a_0, s_1, a_1, \ldots, s_T, a_T|\theta)$$

  where $\tau = s_0, a_0, s_1, a_1, \ldots, s_T, a_T$
- Objective:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\sum_t r_t \right] = \arg\max_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\sum_t r(\tau) \right]$$

- Given observed training data sampled from $P_\theta(\tau)$, we have

$$\theta^* = \arg\max_\theta \sum_{i=1}^{N} \sum_t r_t$$

Parameterization of π for Text Generation

- Copy the parameterized probability here:

$$P_\theta(\tau) = P(s_0, a_0, s_1, a_1, \ldots, s_T, a_T|\theta)$$

- Action $a_t \leftarrow s_t$.
  - Taking an action means given the current state, we choose to generate a new word, and the action space corresponds to the whole vocabulary.
- Transition probability $P_{sa}(s') = P(s'|s, a)$ is the probability of generating a new state based on an action.
  - We set $P_{sa}(s') = I(s' = s \cup \{a\})$ where $I(\cdot)$ is the indicator function. This is because given an selected action, there is no randomness of failure of using the selected words to generate a next state.
- We use the RNN model to parameterize the text generation:

$$\pi_\theta(a_t|s_t) = P_\theta(v|w_t, \mathbf{s}_{t-1})$$

where we denote $\theta = \{\mathbf{M}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{U}\}$ as the parameters

Overview

1. Introduction
2. Maximum Likelihood Based Approach
3. Reinforcement Learning
4. Parameter Estimation: Policy Gradient
5. Evaluation

Score Function Gradient Estimator

- Consider an expectation $\mathbb{E}_{x\sim P(x)}[f(x)]$, we compute gradient w.r.t. θ

$$\begin{aligned}\nabla_\theta \mathbb{E}_{x\sim P(x)}[f(x)] &= \nabla_\theta \int_x dx P(x|\theta) f(x) \\ &= \int_x dx \nabla_\theta P(x|\theta) f(x) \\ &= \int_x dx P(x|\theta) \frac{\nabla_\theta P(x|\theta)}{P(x|\theta)} f(x) \\ &= \int_x dx P(x|\theta) \nabla_\theta \log P(x|\theta) f(x) \\ &= \mathbb{E}_{x\sim P(x)}[f(x) \nabla_\theta \log P(x|\theta)]\end{aligned}$$

- Last expression gives us an unbiased gradient estimator. So we can sample $x_i \sim P(x|\theta)$ and compute

$$\hat{g}_i = f(x_i) \nabla_\theta \log P(x_i|\theta)$$

  for stochastic gradient decent
- Valid even if $f(x)$ is discontinuous, and unknown, or sample space (containing x) is a discrete set

Score Function Gradient Estimator for Policies

- Now the random variable is from the whole trajectory $\tau = s_0, a_0, s_1, a_1, \ldots, s_T, a_T$

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau[R(\tau)\nabla_\theta \log P(x_i|\theta)]$$

  where $R(\tau) = \sum_t r_t$
- Since $P_\theta(\tau) = P(s_0)\prod_{t=1}^{T} P(s_t|s_{t-1}, a_{t-1})\pi(a_{t-1}|s_{t-1}, \theta)$

$$\log P_\theta(\tau) = \log P(s_0) + \sum_{t=1}^{T}[\log P(s_t|s_{t-1}, a_{t-1}) + \log \pi(a_{t-1}|s_{t-1}, \theta)]$$

- So $\nabla_\theta \log P_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta)$ and

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau[R(\tau) \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta)]$$

Comparison to Maximum Likelihood

- Policy gradient

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) \right) \left(\sum_{t=1}^{T} r_t \right)$$

- Maximum likelihood

$$\nabla_\theta \log P(\mathcal{W}) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) \right)$$

Comparison of ML and RL

We can sample τ from $\pi_\theta(a_t|s_t)$ (self play)

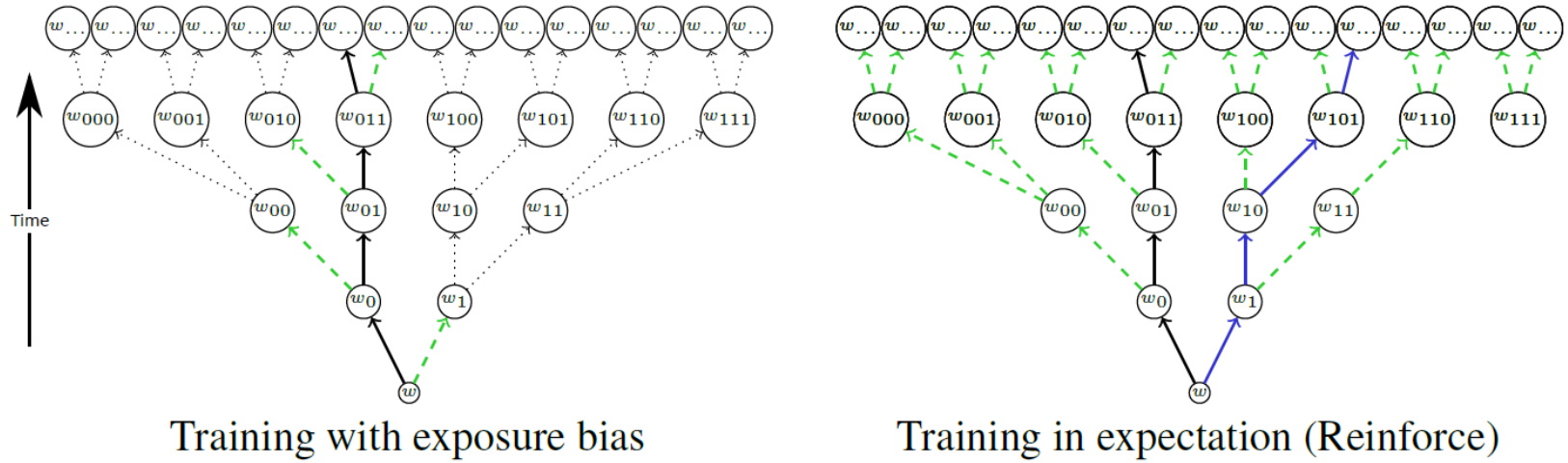


Figure 9: Search space for the toy case of a binary vocabulary and sequences of length 4. The trees represent all the 2^4 possible sequences. The solid black line is the ground truth sequence. **(Left)** Greedy training such as XENT optimizes only the probability of the next word. The model may consider choices indicated by the green arrows, but it starts off from words taken from the ground truth sequence. The model experiences exposure bias, since it sees only words branching off the ground truth path; **(Right)** REINFORCE and MIXER optimize over all possible sequences, using the predictions made by the model itself. In practice, the model samples only a single path indicated by the blue solid line. The model does not suffer from exposure bias; the model is trained as it is tested.

(Ranzato et al. (2016))

Policy Gradient: Introduce Baseline

- Policy at time t' cannot affect reward at time t where $t < t'$, by denoting $\sum_{t'=t}^{T} r_{t'}$ as "reward to go" we have

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) \left(\sum_{t'=t}^{T} r_{t'} \right) \right]$$

  Then at every time step t we can compute stochastic gradient
- Further reduce variance by introducing a baseline $b(s)$

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) \left(\sum_{t'=t}^{T} r_{t'} - b(s_t) \right) \right]$$

- For any choice of b, gradient estimator is unbiased
- Near optimal choice is expected return $b(s_t) = \mathbb{E}[r_t + r_{t+1} + \ldots + r_T]$
- Interpretation: increase log prob of action a_t proportionally to how much returns $\sum_{t'=t}^{T} r_{t'}$ are better than expected

Baseline – Derivation of Unbiased

$$
\begin{aligned}
& \mathbb{E}_{\tau}\left[\nabla_{\theta} \log \pi(a_t|s_t,\theta) b(s_t)\right] \\
= \; & \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_{\theta} \log \pi(a_t|s_t,\theta) b(s_t)\right]\right] \text{ (break up expectation)} \\
= \; & \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_{\theta} \log \pi(a_t|s_t,\theta)\right]\right] \text{ (pull baseline term out)} \\
= \; & \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\mathbb{E}_{a_t}\left[\nabla_{\theta} \log \pi(a_t|s_t,\theta)\right]\right] \text{ (remove irrelevant variables)} \\
= \; & \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\cdot 0\right]
\end{aligned}
$$

This equals to 0 because

$$
\begin{aligned}
& \mathbb{E}_{a_t}\left[\nabla_{\theta} \log \pi(a_t|s_t,\theta)\right] \\
= \; & \int_{a_t} \pi(a_t|s_t,\theta) \nabla_{\theta} \log \pi(a_t|s_t,\theta) da_t \\
= \; & \int_{a_t} \nabla_{\theta} \pi(a_t|s_t,\theta) da_t \\
= \; & \nabla_{\theta} \int_{a_t} \pi(a_t|s_t,\theta) da_t = \nabla_{\theta} 1
\end{aligned}
$$

Baseline – Derivation of Variance Reduction

- We denote a simplified form of gradient as $\mathbb{E}_\tau[\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b)]$
- $\text{Var}[x] = \mathbb{E}[x^2] - E[x]^2$

$$\text{Var} = \mathbb{E}_\tau\left[(\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b))^2\right] - \mathbb{E}_\tau[\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b)]^2$$

- $\mathbb{E}_\tau[\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b)] = \mathbb{E}_\tau[\nabla_\theta \log \pi_\theta(\tau)R(\tau)]$ according to unbiased, which is irrelevant to b
- Denote $\nabla_\theta \log \pi_\theta(\tau) = g_\theta(\tau)$. We have

$$\frac{d\text{Var}}{db} = \frac{d}{db}\mathbb{E}_\tau\left[(g_\theta(\tau)(R(\tau) - b))^2\right]$$

$$= -2\mathbb{E}_\tau[g_\theta(\tau)^2 R(\tau)] + 2b\mathbb{E}_\tau[g_\theta(\tau)^2] = 0$$

- So we have

$$b = \frac{\mathbb{E}_\tau[g_\theta(\tau)^2 R(\tau)]}{\mathbb{E}_\tau[g_\theta(\tau)^2]} \text{(expected reward weighted by gradient magnitudes)}$$

REINFORCE Algorithm

- Initialize policy parameters θ, baseline b
- for iterations= 1,2,...
  - Sample a set of trajectories$\{\tau_i\}$ from $\pi_\theta(a_t|s_t)$
  - At each time in each trajectory, compute return $R_t = \sum_{t'=t}^{T-1} r_{t'}$ and $R_t - b(s_t)$
  - Update the policy, using policy gradient estimate $\sum \nabla_\theta \log \pi(a_t|s_t, \theta)(R_t - b(s_t))$ (plug into SGD or ADAM)
  - Re-fit baseline by $\min ||b(s_t) - R_t||^2$

Practical Algorithm by Ranzato et al. (2016)

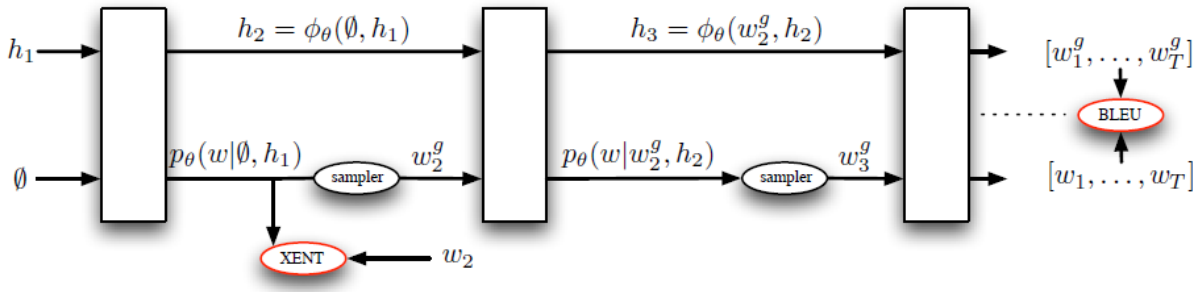


Figure 4: Illustration of MIXER. In the first s unrolling steps (here $s = 1$), the network resembles a standard RNN trained by XENT. In the remaining steps, the input to each module is a sample from the distribution over words produced at the previous time step. Once the end of sentence is reached (or the maximum sequence length), a reward is computed, e.g., BLEU. REINFORCE is then used to back-propagate the gradients through the sequence of samplers. We employ an annealing schedule on s, starting with s equal to the maximum sequence length T and finishing with $s = 1$.

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.

Overview

1. Introduction
2. Maximum Likelihood Based Approach
3. Reinforcement Learning
4. Parameter Estimation: Policy Gradient
5. Evaluation

Evaluation (Machine Translation)

- BLEU (bilingual evaluation understudy) (Papineni et al. (2002))
  - BLEU was one of the first metrics to claim a high correlation with human judgements of quality, and remains one of the most popular automated and inexpensive metrics
  - BLEUs output is always a number between 0 and 1
  - 1 means identical to the reference translations

Example (Poor machine translation output with high precision)

Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

Precision is $\frac{7}{7}$, since all the seven words in the candidate translation appear in the reference translations.

BLEU Score

- First define brevity penalty (BP)

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

  where c is the length of the candidate translation and r is the effective reference corpus length
- p_n: n-gram precision
- BLEU score is defined as weighed geometric mean multiplied by brevity penalty

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

- The ranking behavior is more immediately apparent in the log domain

$$\log \text{BLEU} = \min(1 - \frac{r}{c}, 0) + \sum_{n=1}^{N} w_n \log p_n$$

Google Neural Machine Translation (Wu et al. (2016))

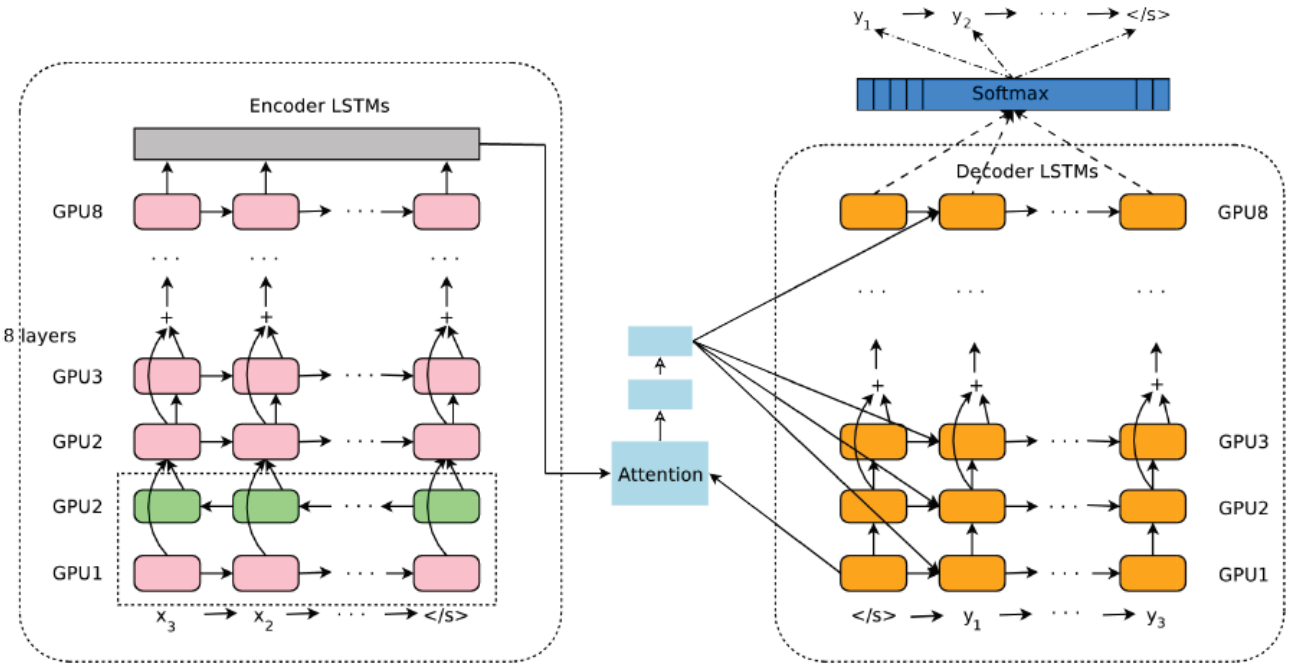


Figure 1: The model architecture of GNMT, Google's Neural Machine Translation system. On the left is the encoder network, on the right is the decoder network, in the middle is the attention module. The bottom encoder layer is bi-directional: the pink nodes gather information from left to right while the green nodes gather information from right to left. The other layers of the encoder are uni-directional. Residual connections start from the layer third from the bottom in the encoder and decoder. The model is partitioned into multiple GPUs to speed up training. In our setup, we have 8 encoder LSTM layers (1 bi-directional layer and 7 uni-directional layers), and 8 decoder layers. With this setting, one model replica is partitioned 8-ways and is placed on 8 different GPUs typically belonging to one host machine. During training, the bottom bi-directional encoder layers compute in parallel first. Once both finish, the uni-directional encoder layers can start computing, each on a separate GPU. To retain as much parallelism as possible during running the decoder layers, we use the bottom decoder layer output only for obtaining recurrent attention context, which is sent directly to all the remaining decoder layers. The softmax layer is also partitioned and placed on multiple GPUs. Depending on the output vocabulary size we either have them run on the same GPUs as the encoder and decoder networks, or have them run on a separate set of dedicated GPUs.

Results (Single Model) (Wu et al. (2016))

- English to French

Table 4: Single model results on WMT En→Fr (newstest2014)

Model	BLEU	CPU decoding time per sentence (s)
Word	37.90	0.2226
Character	38.01	1.0530
WPM-8K	38.27	0.1919
WPM-16K	37.60	0.1874
WPM-32K	38.95	0.2118
Mixed Word/Character	38.39	0.2774
PBMT [15]	37.0	
LSTM (6 layers) [31]	31.5	
LSTM (6 layers + PosUnk) [31]	33.1	
Deep-Att [45]	37.7	
Deep-Att + PosUnk [45]	39.2	

Results (Single Model) (Wu et al. (2016))

- English to German

Table 5: Single model results on WMT En→De (newstest2014)

Model	BLEU	CPU decoding time per sentence (s)
Word	23.12	0.2972
Character (512 nodes)	22.62	0.8011
WPM-8K	23.50	0.2079
WPM-16K	24.36	0.1931
WPM-32K	24.61	0.1882
Mixed Word/Character	24.17	0.3268
PBMT [6]	20.7	
RNNSearch [37]	16.5	
RNNSearch-LV [37]	16.9	
RNNSearch-LV [37]	16.9	
Deep-Att [45]	20.6	

Results (Refine with RL)

- Single models and Ensemble Models

Table 6: Single model test BLEU scores, averaged over 8 runs, on WMT En→Fr and En→De

Dataset	Trained with log-likelihood	Refined with RL
En→Fr	38.95	39.92
En→De	24.67	24.60

Table 7: Model ensemble results on WMT En→Fr (newstest2014)

Model	BLEU
WPM-32K (8 models)	40.35
RL-refined WPM-32K (8 models)	41.16
LSTM (6 layers) [31]	35.6
LSTM (6 layers + PosUnk) [31]	37.5
Deep-Att + PosUnk (8 models) [45]	40.4

Table 8: Model ensemble results on WMT En→De (newstest2014). See Table 5 for a comparison against non-ensemble models.

Model	BLEU
WPM-32K (8 models)	26.20
RL-refined WPM-32K (8 models)	26.30

Results (Human Evaluation)

- Compared to the best phrase based translations as downloaded from `http://matrix.statmt.org/systems/show/2065`

Table 9: Human side-by-side evaluation scores of WMT En→Fr models.

Model	BLEU	Side-by-side averaged score
PBMT [15]	37.0	3.87
NMT before RL	40.35	4.46
NMT after RL	41.16	4.44
Human		4.82

Even though RL refinement can achieve better BLEU scores, it barely improves the human impression of the translation quality.

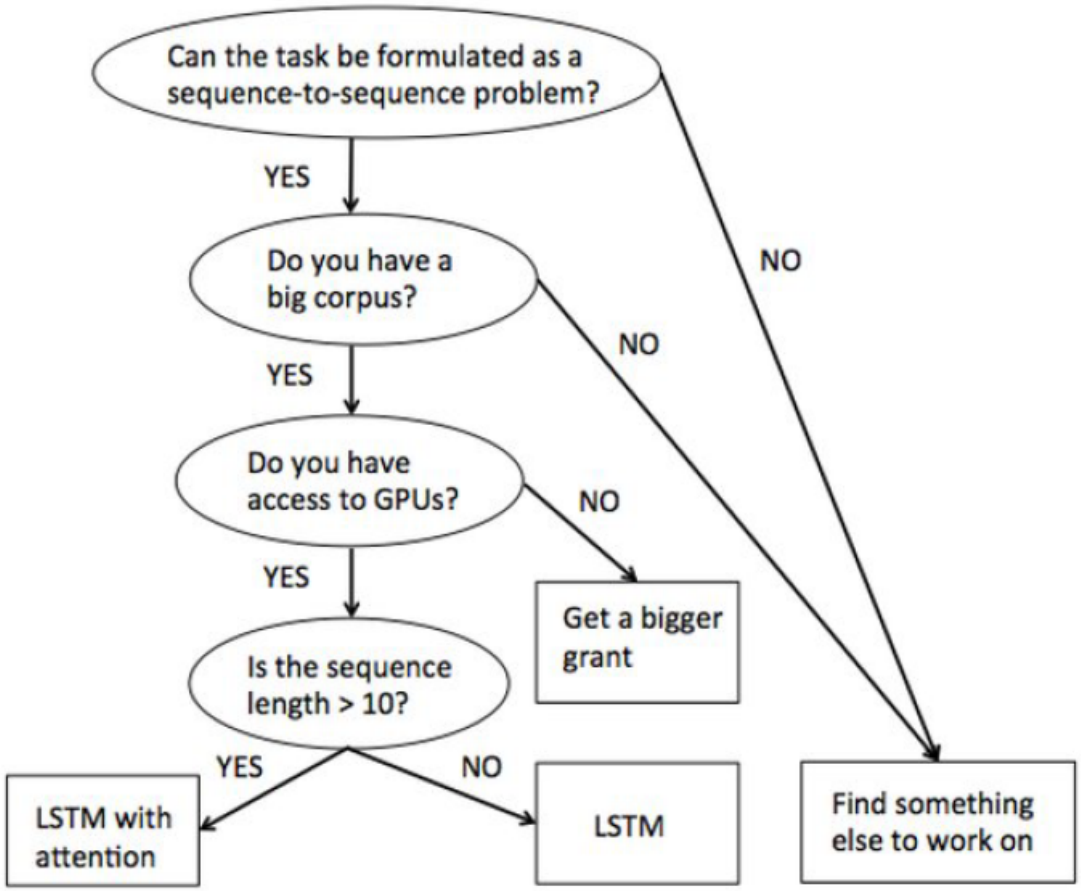
- The relatively small sample size for the experiment (only 500 examples for side-by-side)
- The improvement in BLEU score by RL is relatively small after model ensembling (0.81)
- The possible mismatch between BLEU as a metric and real translation quality as perceived by human raters

Possible Improvements

- Imitation Learning
  - Instead of directly learning from evaluation scores, e.g., BLEU, imitation learning assumes the reward is unknown, and directly learns a policy from human references
- Firstly proposed as "behavioral cloning" (Pomerleau (1991)) and "inverse reinforcement learning" (Russell (1998); Ng et al. (2000))
- Recently, people have built relationships between adversarial training of imitation learning and generative adversarial networks (Goodfellow et al. (2014))
  - (Ho and Ermon (2016); Finn et al. (2016))
- Many algorithms are (being) developed

Isabelle Augenstein@University of Copenhagen

https://twitter.com/IAugenstein/status/710837374473920512



References I

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. In *ICLR*.

Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2014). Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, pages 677–691.

Finn, C., Christiano, P., Abbeel, P., and Levine, S. (2016). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. In *NIPS-workshop*.

Forcada, M. L. and Ñeco, R. P. (1997). Recursive hetero-associative memories for translation. In *IWANN*, pages 453–462.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial nets. In *NIPS*, pages 2672–2680.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *NIPS*, pages 4565–4573.

References II

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670.

Papineni, K., Roukos, S., Ward, T., and Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318.

Pomerleau, D. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2016). Sequence level training with recurrent neural networks. In *ICLR*.

Russell, S. J. (1998). Learning agents for uncertain environments (extended abstract). In *COLT*, pages 101–103.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *CVPR*.

References III

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.