@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

Learning to Encode Text as Human-Readable Summaries

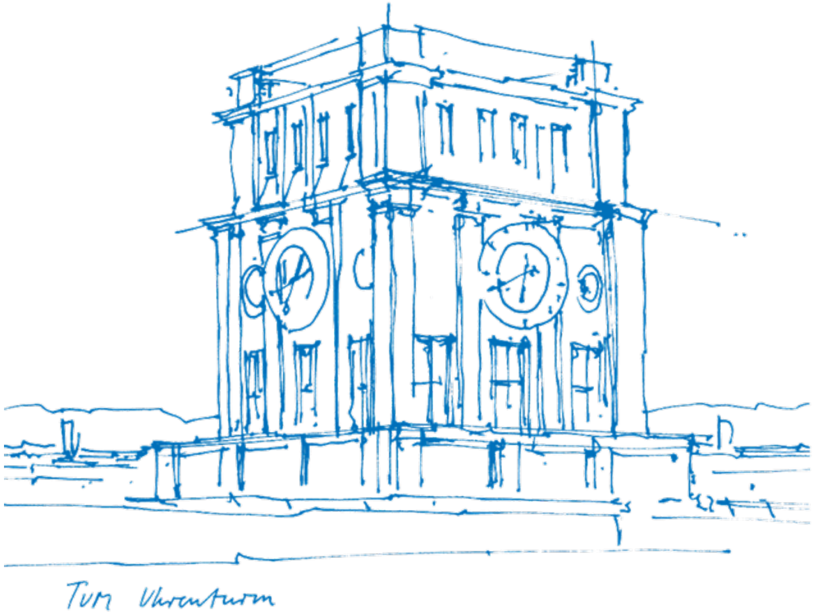
Generative Adversarial Networks

Denys Lazarenko[1], Author 2[2]

[1]Department of Mathematics , Technical University of Munich (TUM)

[2]Department of Informatics, Technical University of Munich (TUM)

Seminar Course - Adversarial and Secure Machine Learning



Motivation

[1]

[1] **jirauschek2014**.

Outline

Overview

Learning to Encode Text as Human-Readable Summaries using Generative Adversarial Networks

Yau-Shian Wang
National Taiwan University
king6101@gmail.com

Hung-Yi Lee
National Taiwan University
tlkagkb93901106@gmail.com

Abstract

Auto-encoders compress input data into a latent-space representation and reconstruct the original data from the representation. This latent representation is not easily interpreted by humans. In this paper, we propose training an auto-encoder that encodes input text into human-readable sentences, and unpaired abstractive summarization is thereby achieved. The auto-encoder is composed of a generator and a reconstructor. The generator encodes the input text into a shorter word sequence, and the reconstructor recovers the generator input

we use comprehensible natural language as a latent representation of the input source text in an auto-encoder architecture. This human-readable latent representation is shorter than the source text; in order to reconstruct the source text, it must reflect the core idea of the source text. Intuitively, the latent representation can be considered a summary of the text, so unpaired abstractive summarization is thereby achieved.

The idea that using human comprehensible language as a latent representation has been explored on text summarization, but only in a semi-

Assumptions

- CNN dataset
- Unpaired WGAN model
- 5000 words in vocabulary (Pointer model)
- without coverage mechanism (Pointer model)
- ...

Task	Labeled	Methods	R-1	R-2	R-L
(A)Supervised	3.8M	(A-1)Supervised training on generator	33.19	14.21	30.50
		(A-2) (Rush et al., 2015)†	29.76	11.88	26.96
		(A-3) (Chopra et al., 2016)†	33.78	15.97	31.15
		(A-4) (Zhou et al., 2017)†	**36.15**	**17.54**	**33.63**
(B) Trivial baseline	0	(B-1) Lead-8	21.86	7.66	20.45
(C) Unpaired	0	(C-1) Pre-trained generator	21.26	5.60	18.89
		(C-2) WGAN	28.09	9.88	25.06
		(C-3) **Adversarial REINFORCE**	**28.11**	**9.97**	**25.41**
(D) Semi-supervised	10K	(D-1) WGAN	29.17	10.54	26.72
		(D-2) **Adversarial REINFORCE**	**30.01**	**11.57**	**27.61**
	500K	(D-3)(Miao and Blunsom, 2016)†	30.14	12.05	27.99
		(D-4) WGAN	32.50	13.65	29.67
		(D-5) **Adversarial REINFORCE**	**33.33**	**14.18**	**30.48**
	1M	(D-6)(Miao and Blunsom, 2016)†	31.09	12.79	28.97
		(D-7) WGAN	33.18	14.19	30.69
		(D-8) **Adversarial REINFORCE**	**34.21**	**15.16**	**31.64**
(E) Transfer learning	0	(E-1) Pre-trained generator	21.49	6.28	19.34
		(E-2) **WGAN**	25.11	7.94	23.05
		(E-3) **Adversarial REINFORCE**	**27.15**	**9.09**	**24.11**

Table 1: Average F1 ROUGE scores on English Gigaword. R-1, R-2 and R-L refers to ROUGE 1, ROUGE 2 and ROUGE L respectively. Results marked with † are obtained from corresponding papers. In part (A), the model was trained supervisedly. In row (B-1), we select the article's first eight words as its summary. Part (C) are the results obtained without paired data. In part (D), we trained our model with few labeled data. In part (E), we pre-trained generator on CNN/Diary and used the summaries from CNN/Diary as real data for the discriminator.

CNN / Daily Mail dataset

DeepMind Q&A Dataset

Hermann et al. (2015) created two awesome datasets using news articles for Q&A research. Each dataset contains many documents (90k and 197k each), and each document companies on average 4 questions approximately. Each question is a sentence with one missing word/phrase which can be found from the accompanying document/context.

The original authors kindly released the scripts and accompanying documentation to generate the datasets (see here). Unfortunately due to instability of WaybackMachine, it is often cumbersome to generate the datasets from scratch using the provided scripts. Furthermore, in certain parts of the world, it turned out to be far from being straight-forward to access the WaybackMachine.

I am making the generated datasets available here. This will hopefully make the datasets used by a wider audience and lead to faster progress in Q&A research.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., & Blunsom, P. (2015).
Teaching machines to read and comprehend.
In Advances in Neural Information Processing Systems (pp. 1684-1692).

CNN

- Questions: here
- Stories: here
- Raw HTML: here

This dataset contains the documents and accompanying questions from the news articles of CNN. There are approximately 90k documents and 380k questions. I am making available 'questions/', which should be sufficient to reproduce the setting from the original paper, and 'stories/', which can be useful for other uses of this dataset. I am also making the raw html files available, but I cannot guarantee that these are complete.

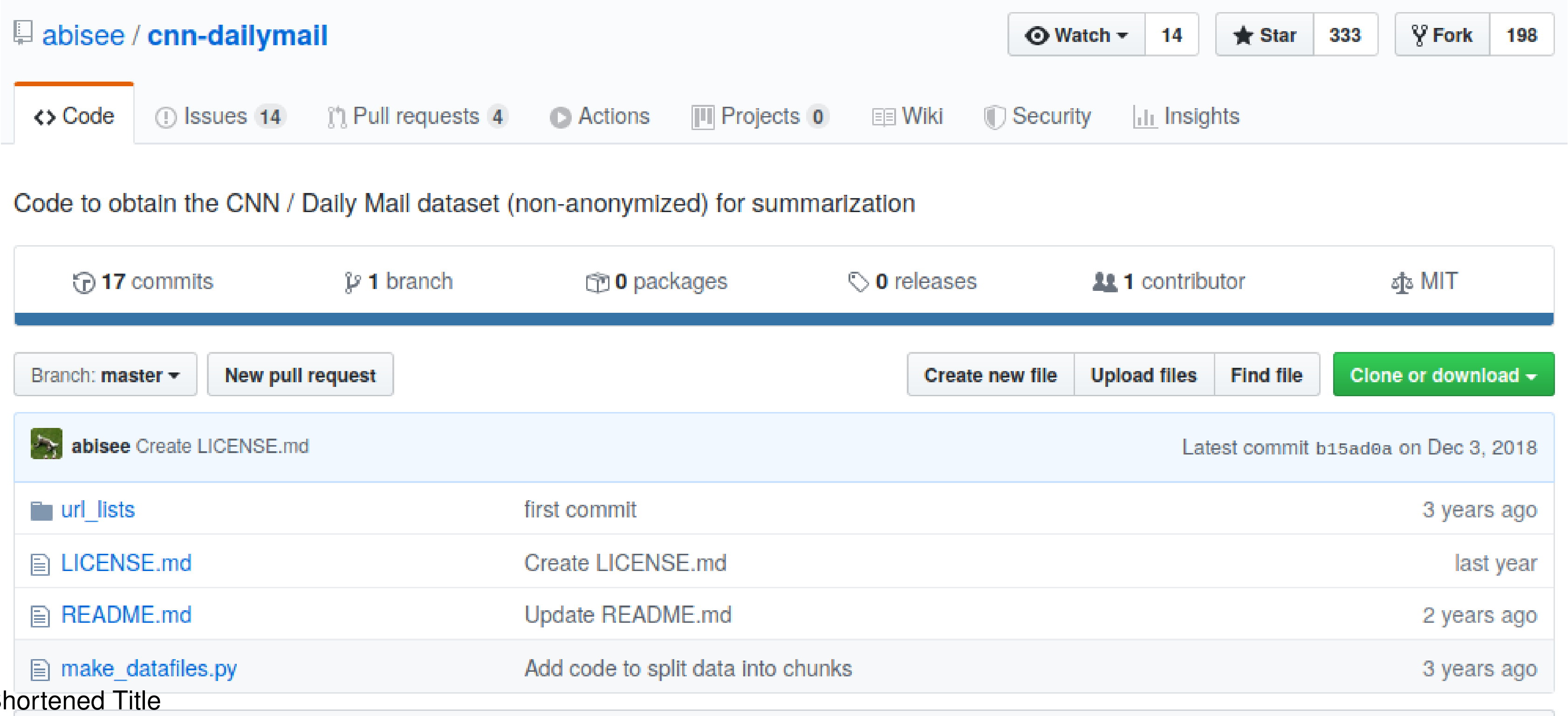
Daily Mail

- Questions: here
- Stories: here
- Raw HTML: here

This dataset contains the documents and accompanying questions from the news articles of Daily Mail. There are approximately 197k documents and 879k questions. I am making available 'questions/', which should be sufficient to reproduce the setting from the original paper, and 'stories/', which can be useful for other uses of this dataset. I am also making the raw html files available, but I cannot guarantee that these are complete.

Kyunghyun Cho

CNN / Daily Mail dataset



Load Dataset

abisee / **pointer-generator**

Watch 57 | Star 1.5k | Fork 654

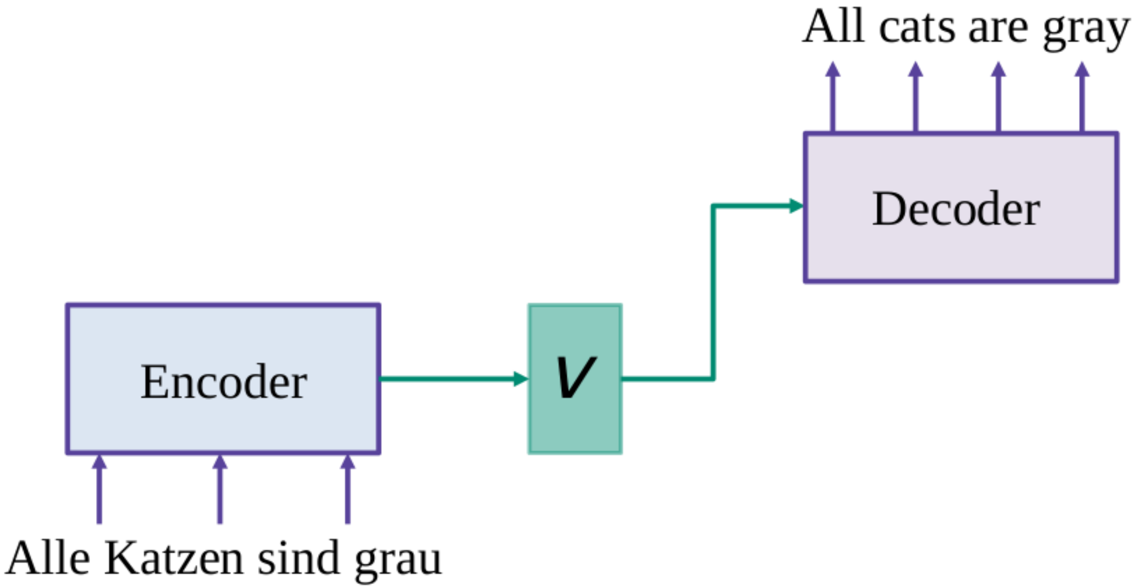
Code | Issues 84 | Pull requests 7 | Actions | Projects 0 | Wiki | Security | Insights

```python
class Batcher(object):
  """A class to generate minibatches of data. Buckets examples together based on length of the encoder sequence."""

  BATCH_QUEUE_MAX = 100 # max number of batches the batch_queue can hold

  def __init__(self, data_path, vocab, hps, single_pass):
    """Initialize the batcher. Start threads that process the data into batches.

    Args:
      data_path: tf.Example filepattern.
      vocab: Vocabulary object
      hps: hyperparameters
      single_pass: If True, run through the dataset exactly once (useful for when you want to run evaluation on the dev or test
    """
    self._data_path = data_path
    self._vocab = vocab
    self._hps = hps
    self._single_pass = single_pass
```

Encode-decoder architectur

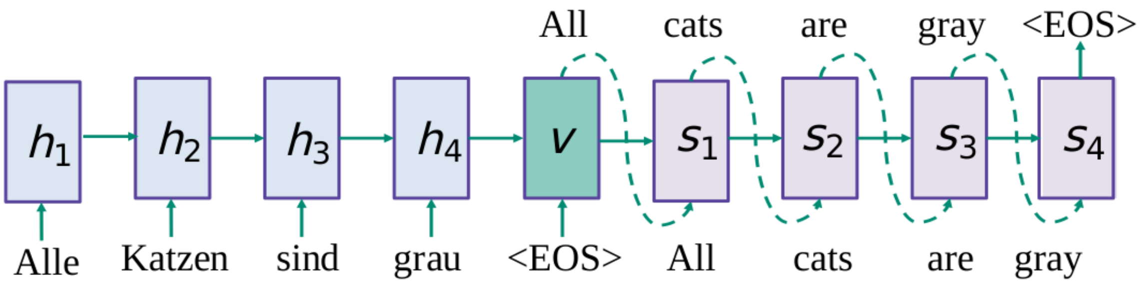
@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

Sequence to sequence



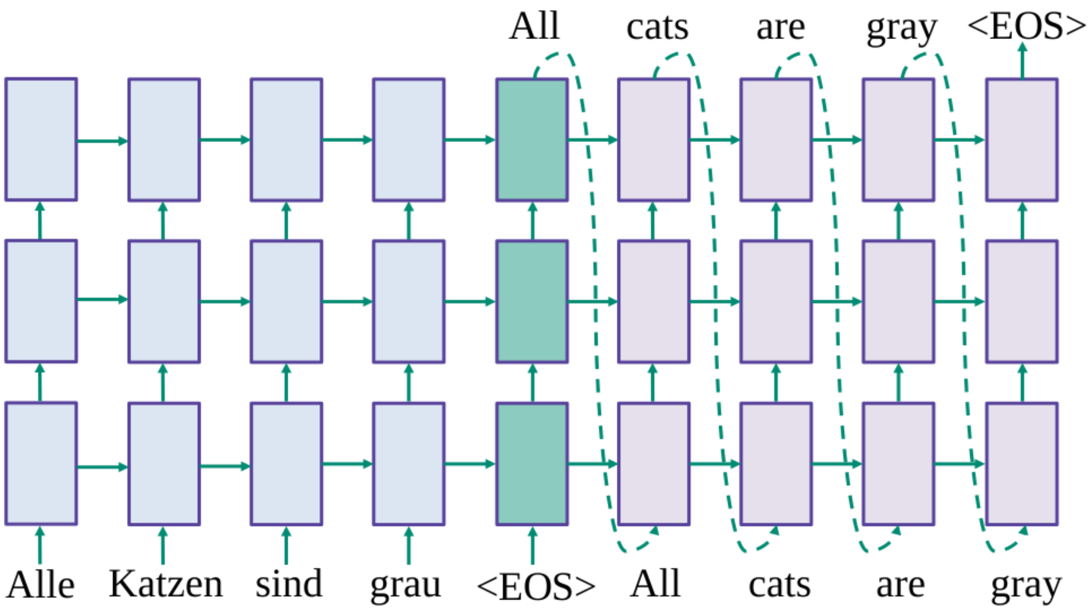
From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Sequence to sequence



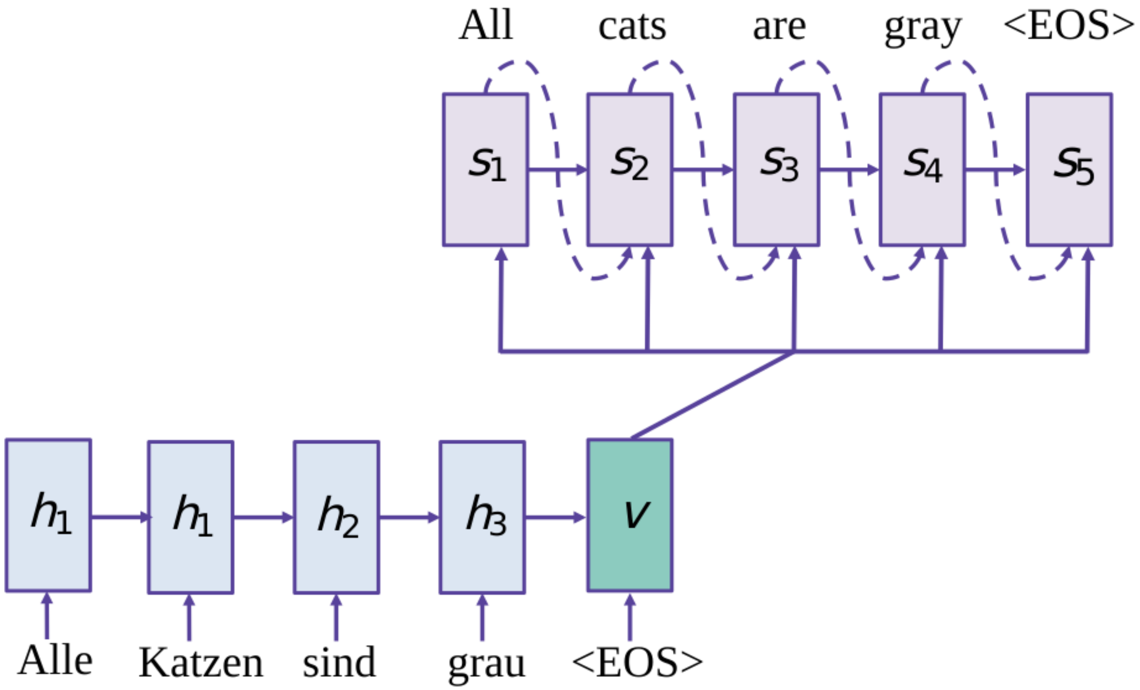
From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Sequence to sequence



From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Sequence to sequence



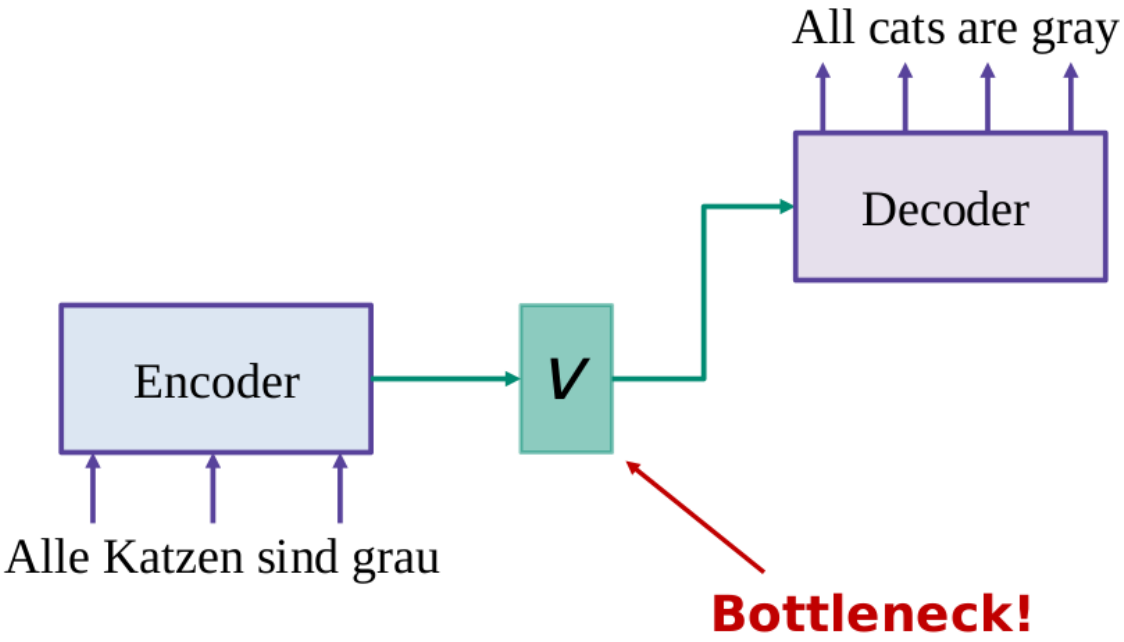
From "Natural Language Processing", A. Potapenko et.al. HSE, 2019

$$\mathbb{P}\left[y_1, \ldots y_J | x_1, \ldots, x_I\right] = \prod_{j=1}^{J} \mathbb{P}\left[y_j | v, y_1, \ldots, y_{j-1}\right]$$

- **Encoder:** maps the source sequence to the hidden vector
  $RNN : h_i = f(h_{i-1}, x_i)$, where $v = h_I$
- **Decoder:** performs language modeling given this vector
  $RNN : s_j = g(s_{j-1}, [y_{j-1}, v])$
- **Prediction:** $\mathbb{P}\left[y_j | v, y_1, \ldots, y_{j-1}\right] = softmax(Us_j + b)$

From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

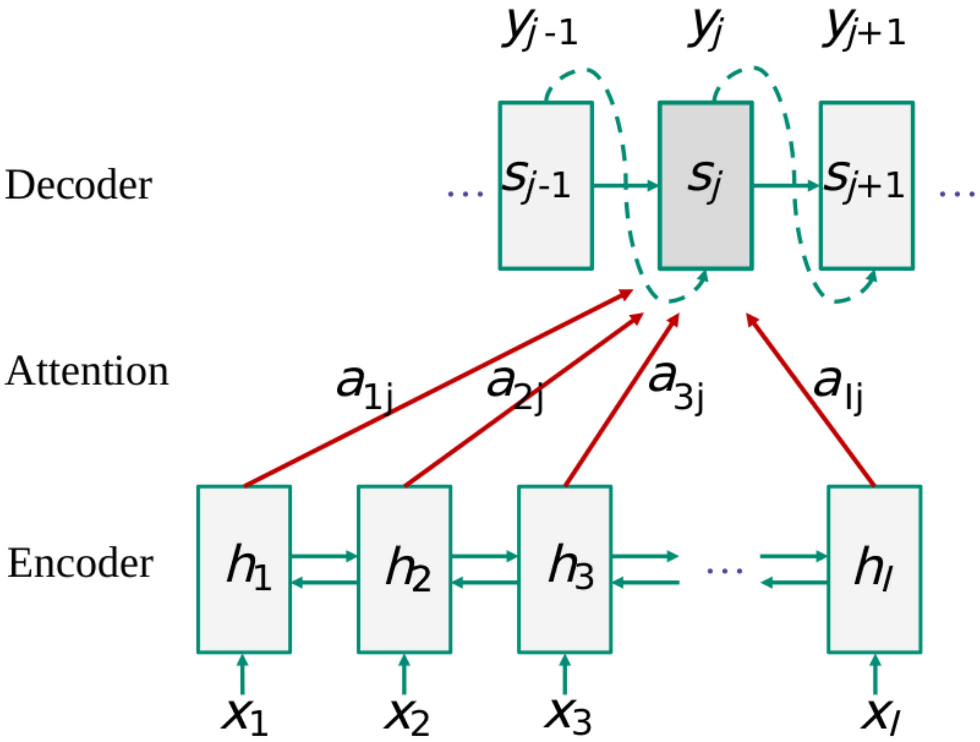
Sequence to sequence



From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Attention mechanism

Attention mechanism



- Encoder states are weighted to obtain the representation relevant to the decoder state:

$$v_j = \sum_{i=1}^{I} \alpha_{ij} h_i$$

- The weights are learnt and should find the most relevant encoder positions:

$$\alpha_{ij} = \frac{exp(sim(h_i, s_{j-1}))}{\sum_{i'}^{I} exp(sim(h_{i'}, s_{j-1}))}$$

From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

- **Additive attention:**

$$sim(h_i, s_j) = w^T \tanh(W_h h_i + W_s s_j)$$

- **Multiplicative attention:**

$$sim(h_i, s_j) = h_i^T W s_j$$

- **Dot product also works:**

$$sim(h_i, s_j) = h_i^T s_j$$

From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Put all together

$$\mathbb{P}\left[y_1, \ldots y_J | x_1, \ldots, x_I\right] = \prod_{j=1}^{J} \mathbb{P}\left[y_j | v_j, y_1, \ldots, y_{j-1}\right]$$

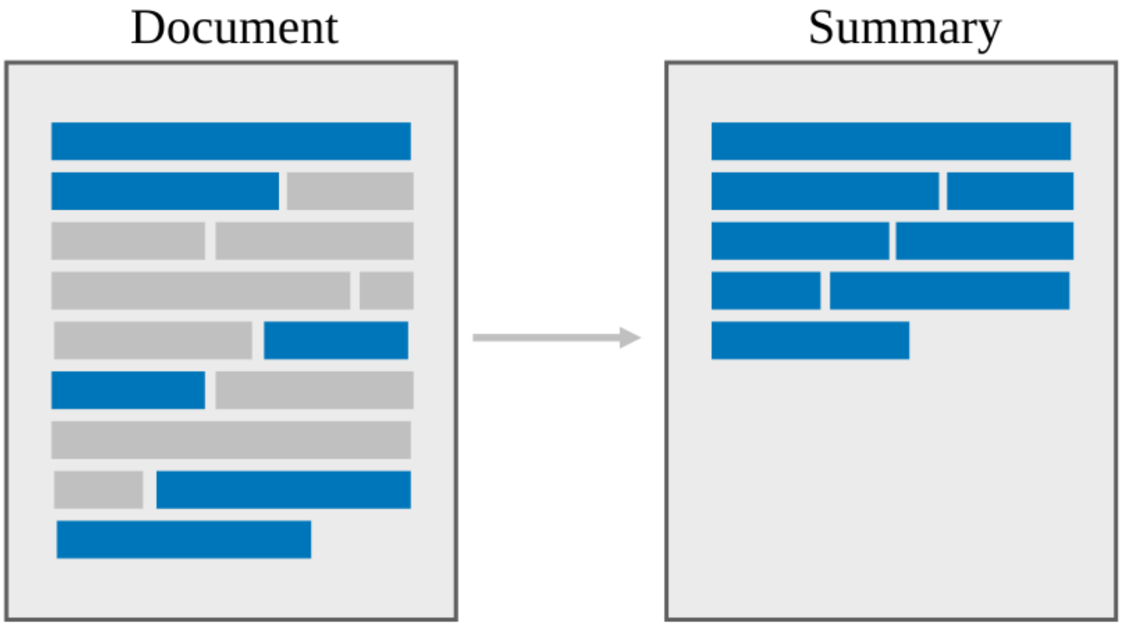
- **Encoder:** $h_i = f(h_{i-1}, x_i)$
- **Decoder:** $s_j = g(s_{j-1}, [y_{j-1}, v_j])$

From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

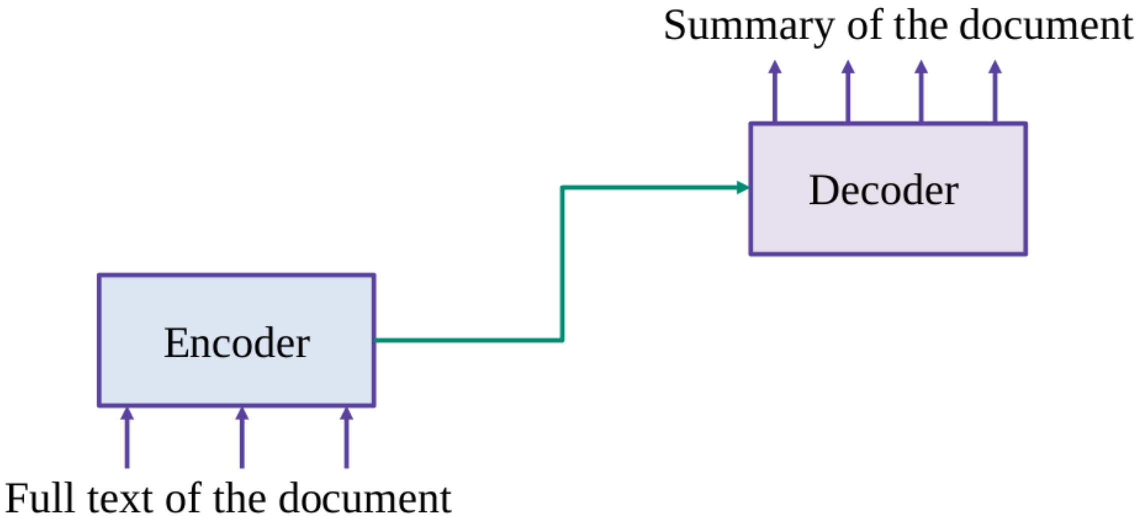
Summarization with pointer-generator networks

@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

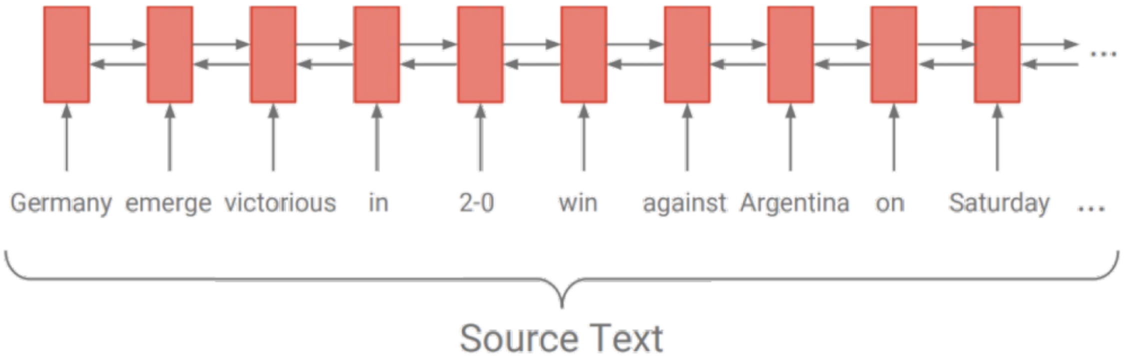
Summarization



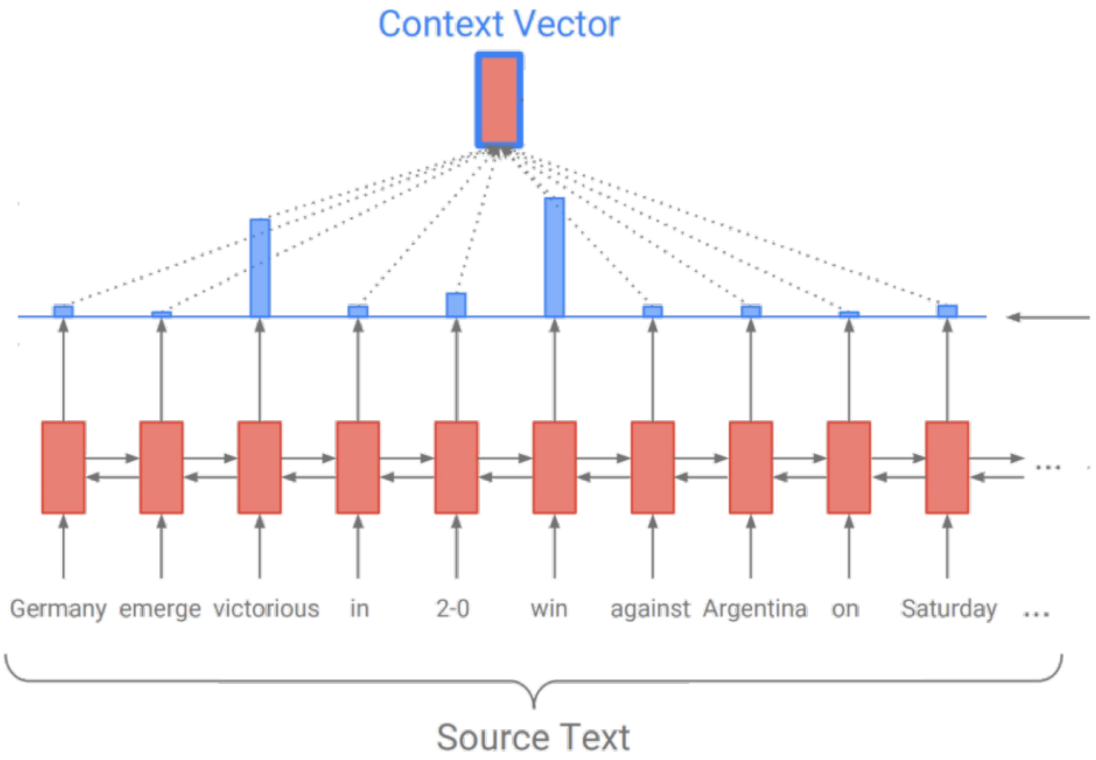
Summarization



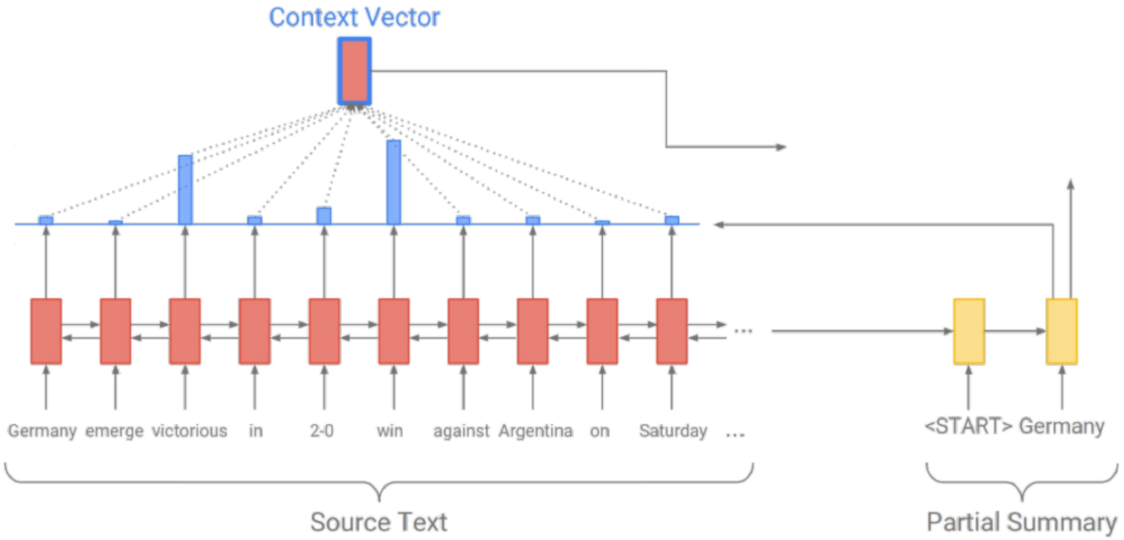
Seq2seq + attention



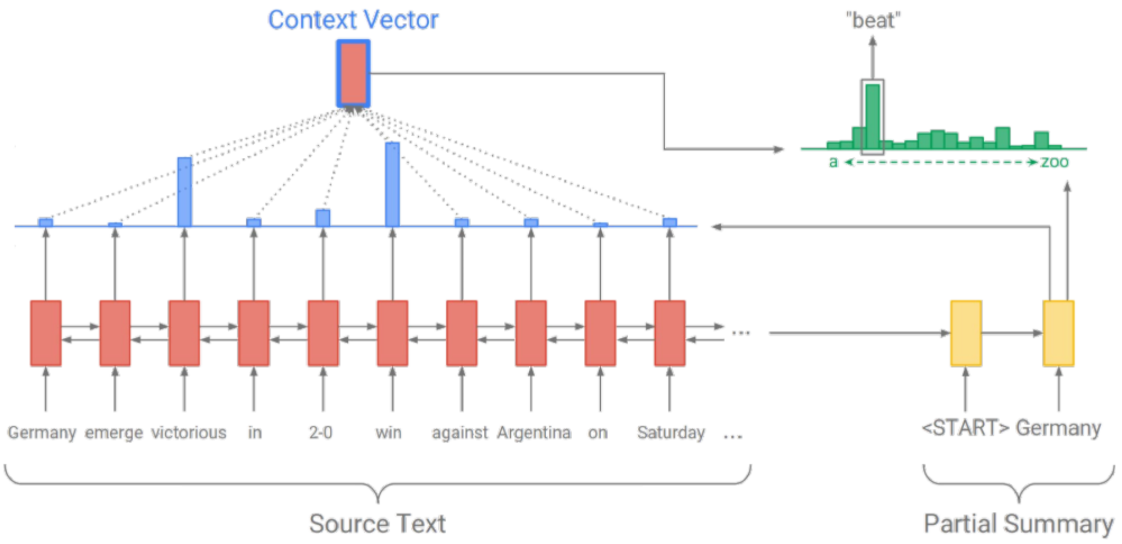
Seq2seq + attention



Seq2seq + attention



Seq2seq + attention



1. Attention distribution (over source positions):

$$e_i^j = w^T \tanh(W_h h_i + W_s s_j + b_{attn})$$

$$p^j = softmax(e^j)$$

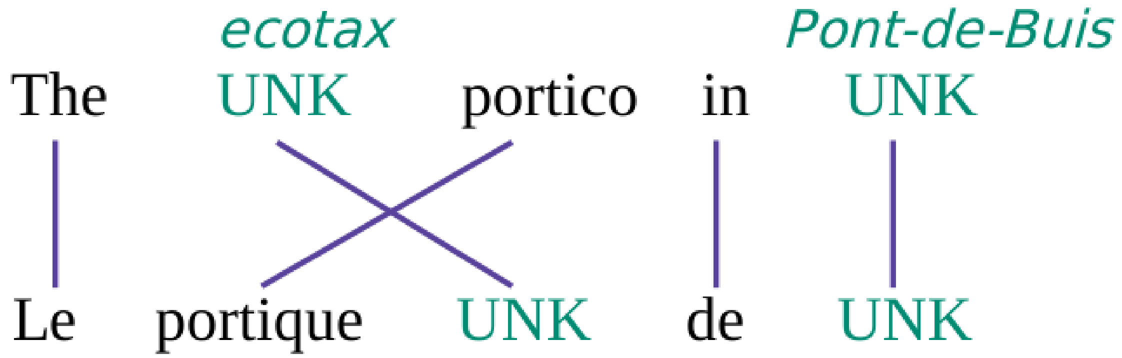
2. Vocabulary distribution (generative model):

$$v_j = \sum_i p_i^j h_i$$

$$p_{vocab} = softmax(V^{'}(V[s_j, v_j] + b) + b^{'})$$

Copy mechanism

What do we do with OOV words?

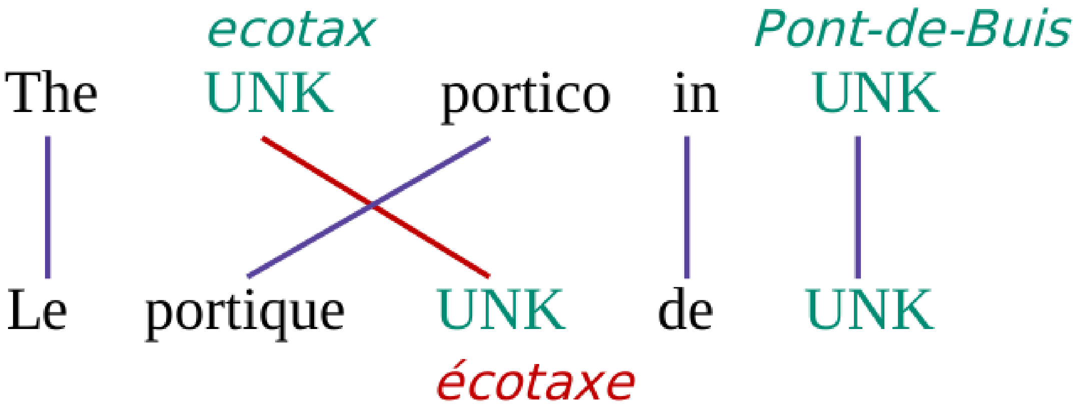


From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Copy mechanism

What do we do with OOV words?

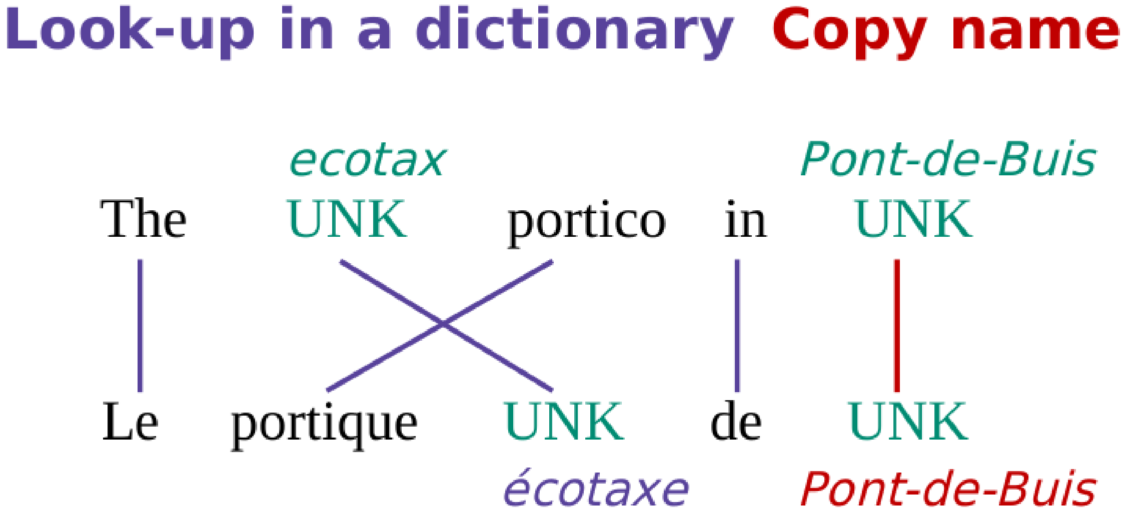
Look-up in a dictionary



From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Copy mechanism

What do we do with OOV words?

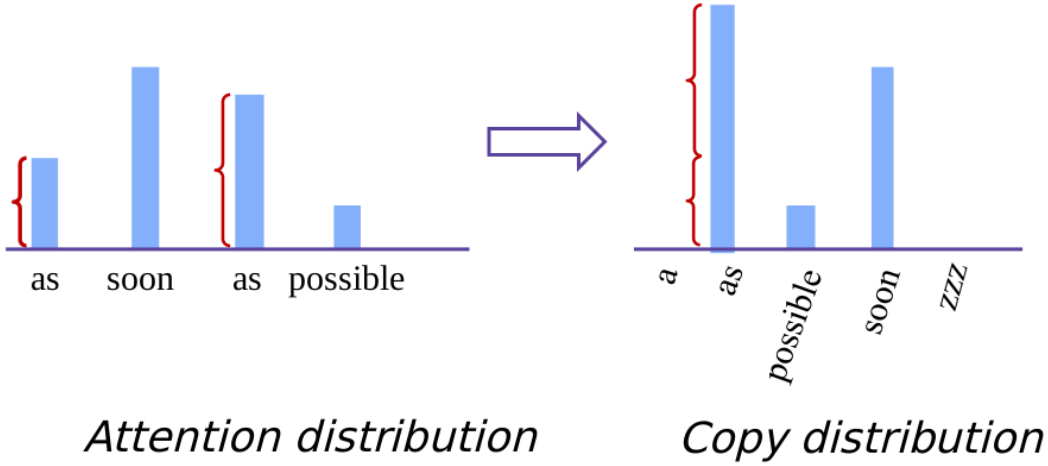


From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

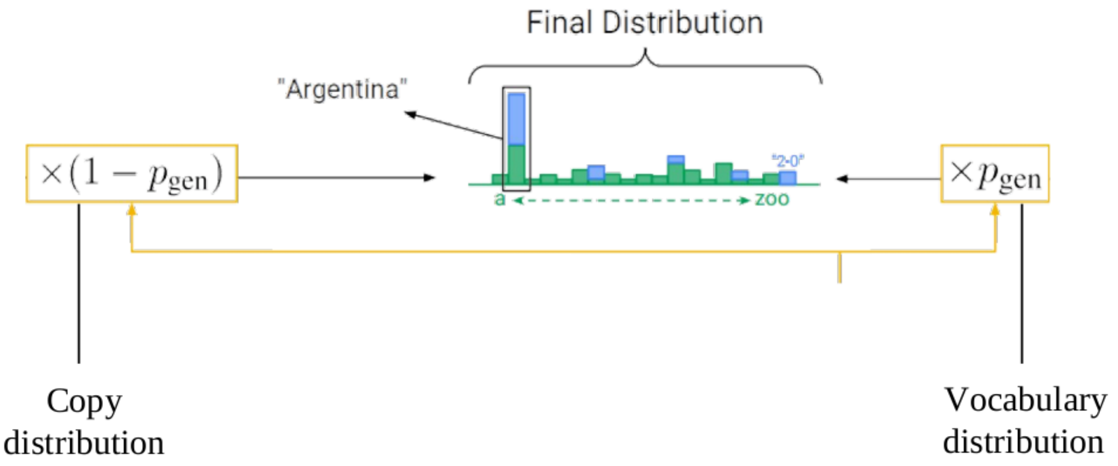
3. Copy distribution (over words from source):

$$p_{copy}(w) = \sum_{i:x_i=w} p_i^j$$



From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Pointer-generator network



4. Final distribution:

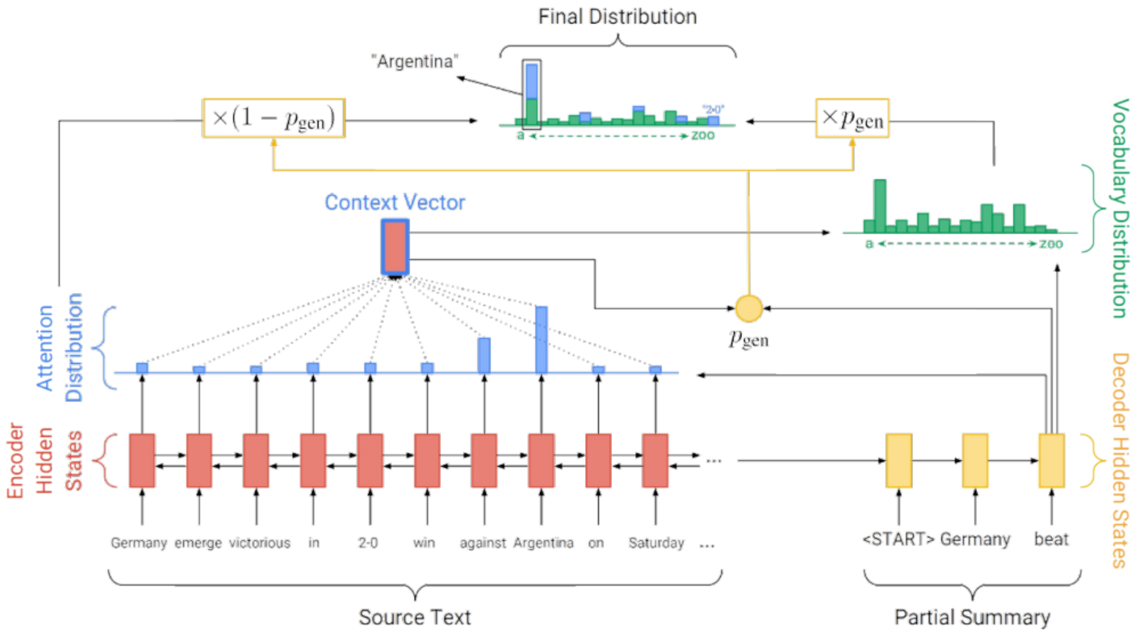
$$p_{final} = p_{gen} p_{vocab} + (1 - p_{gen}) p_{copy}$$

$$p_{gen} = sigmoid(w_v^T v_j + w_s^T s_j + w_x^T y_{j-1} + b_{gen})$$

5. Training:

$$Loss = -\frac{1}{J} \sum_{j=1}^{J} \log p_{final}(y_j)$$

Pointer-generator network



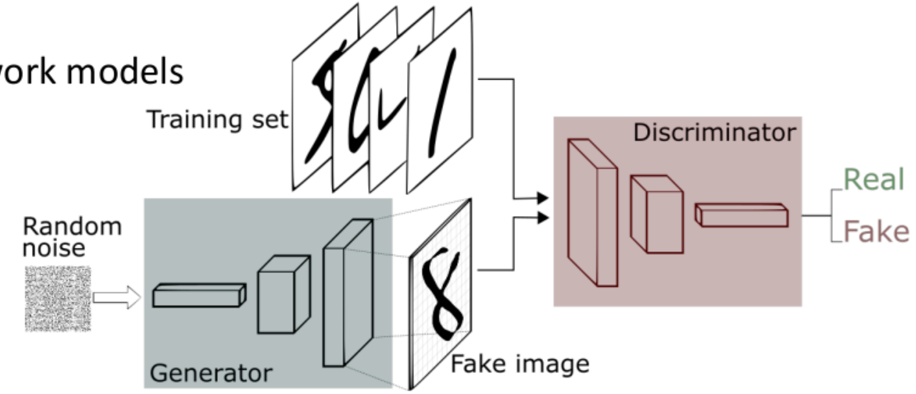
Comparison of the models

	ROUGE score		
	1	2	L
abstractive model (Nallapati et al., 2016)	35.46	13.30	32.65
extractive model (Nallapati et al., 2017)	39.6	16.2	35.3
lead-3 baseline	40.34	17.70	36.57
seq2seq + attention	31.33	11.81	28.83
pointer-generator	36.44	15.66	33.42
pointer-generator + coverage	**39.53**	**17.28**	**36.38**

@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

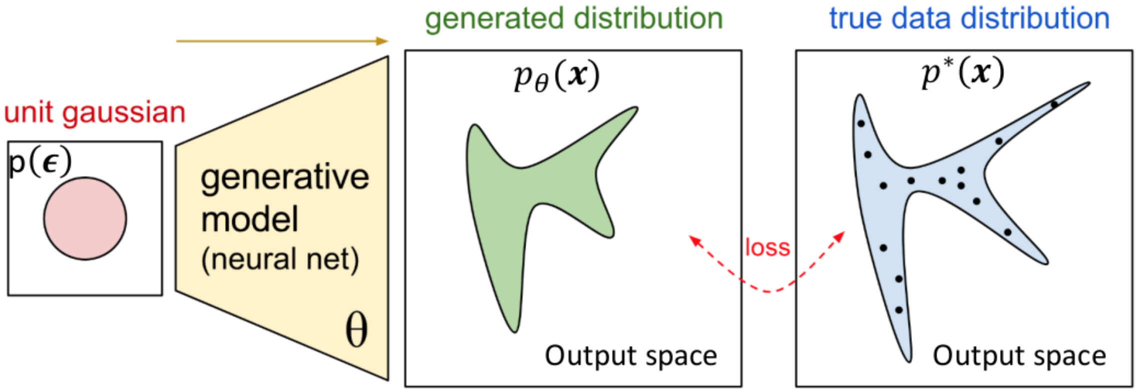
Generative Adversarial Networks

- GANs have become very popular for learning deep generative models
- Informally, the main idea is:
  - Two competing neural network models
  - Generator: takes noise as input and generates ("fake") samples
  - Discriminator: receives samples from both generator and training data and has to distinguish between the two → classify input as "real" or "fake"
  - Goal: Train the generator in such a way that the discriminator can **not** distinguish between real and "fake" samples
    - ➢ In this case, the generator generates realistic examples

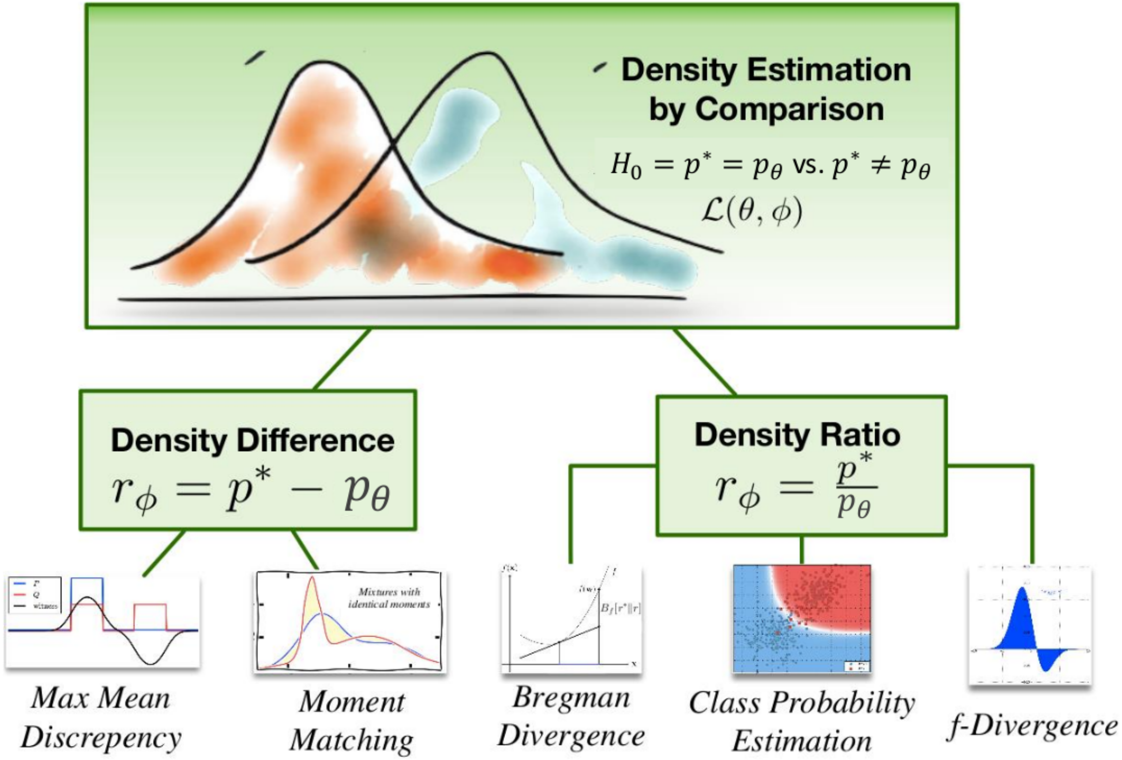


From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- What can we do if we can **easily draw samples** from the model but we cannot evaluate the density?
- Idea: We can use **any** method that compares two **sets of samples**.
  - This process is called ***density estimation by comparison.***
- We **test** the **hypothesis** that the true data distribution $p^*(\boldsymbol{x})$ and the model distribution $p_\theta(\boldsymbol{x})$ are **equal**



From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.



From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Density ratio $r^*(\boldsymbol{x}) = p^*(\boldsymbol{x}) / p_\theta(\boldsymbol{x})$
  - In the best case always 1, i.e. the two distributions are indistinguishable
  - However, we cannot compute ratio in closed form/easily
- Idea: Approximate the true density ratio $r^*(\boldsymbol{x})$ by $r_\phi(\boldsymbol{x})$
  - Finding the approximation $r_\phi(\boldsymbol{x})$ often means solving again a learning problem

- Thus, we get the following general principle for learning
  - **Optimize Ratio loss**:
    approximate the true density ratio $r^*(\boldsymbol{x})$ (i.e. learning ϕ)
  - **Optimize Generative loss**:
    drive the density ratio towards 1 (i.e. learning θ)

  - Essentially a bi-level optimization problem, which is usually just solved alternatingly

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Density ratio $r^*(\boldsymbol{x}) = p^*(\boldsymbol{x}) / p_\theta(\boldsymbol{x})$
  - In the best case always 1, i.e. the two distributions are indistinguishable
  - However, we cannot compute ratio in closed form/easily
- Idea: Approximate the true density ratio $r^*(\boldsymbol{x})$ by $r_\phi(\boldsymbol{x})$
  - Finding the approximation $r_\phi(\boldsymbol{x})$ often means solving again a learning problem

- Thus, we get the following general principle for learning
  - **Optimize Ratio loss**:
    approximate the true density ratio $r^*(\boldsymbol{x})$ (i.e. learning ϕ)
  - **Optimize Generative loss**:
    drive the density ratio towards 1 (i.e. learning θ)

  - Essentially a bi-level optimization problem, which is usually just solved alternatingly

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Let Y denote a random variable which assigns label $Y = 1$ to samples from the true data distribution; $Y = 0$ to those from the generator distribution
- Then $p^*(\boldsymbol{x}) = p(\boldsymbol{x} \mid Y = 1)$ and $p_\theta(\boldsymbol{x}) = p(\boldsymbol{x} \mid Y = 0)$
- Denote $P(Y = 1) = \pi$. From Bayes we have:

$$r^*(\boldsymbol{x}) = \frac{p^*(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} = \frac{p(Y = 1 \mid \boldsymbol{x})}{p(Y = 0 \mid \boldsymbol{x})} \frac{1 - \pi}{\pi}$$

  - Apparently density ratio estimation is equal to **class-probability estimation**
  - Simply speaking: we can consider a classifier for $\boldsymbol{x}$ (predicting labels Y=0 or 1)

➢ Specify a scoring function or a **discriminator** $D_\phi(\boldsymbol{x}) = p(Y = 1 \mid \boldsymbol{x})$

- e.g. logistic regression or a neural network

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Specify a scoring function or a **discriminator** $D_\phi(\boldsymbol{x}) = p(Y = 1|\boldsymbol{x})$
  - e.g. logistic regression or a neural network

- For learning $D_\phi(\boldsymbol{x})$, we need a loss function, e.g., the cross-entropy loss:

$$\mathcal{L}_{\theta,\phi} = \mathbb{E}_{p(\boldsymbol{x}|y)p(y)}\left[-y\log[D_\phi(\boldsymbol{x})] - (1-y)\log[1-D_\phi(\boldsymbol{x})]\right]$$

$$= \pi\mathbb{E}_{p^*(\boldsymbol{x})}\left[-\log D_\phi(\boldsymbol{x})\right] + (1-\pi)\mathbb{E}_{p(\boldsymbol{z})}\left[-\log[1-D_\phi(f_\theta(\boldsymbol{z}))]\right]$$

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

1. Solving $$\phi^*(\theta) = \underset{\phi}{\operatorname{argmin}}\, \mathcal{L}_{\theta,\phi}$$
leads to the "best" discriminator for a given generative model (θ)

- That is, we well approximate $r^*(\boldsymbol{x}) = \frac{p^*(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} = \frac{p(Y=1 \mid \boldsymbol{x})}{p(Y=0 \mid \boldsymbol{x})} \approx \frac{D_{\phi^*(\theta)}(\boldsymbol{x})}{1-D_{\phi^*(\theta)}(\boldsymbol{x})}$
  - here w.l.o.g. we set $\pi = 0.5$

2. We aim to drive the density ratio $r^*(\boldsymbol{x})$ towards 1
   - aim: $p(Y = 1 \mid \boldsymbol{x}) = p(Y = 0 \mid \boldsymbol{x})$

- That is, find generative model (θ) such that (even) the "best" discriminator cannot distinguish the classes

➢ Solving $$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}_{\theta,\phi^*(\theta)}$$

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Generator and discriminator play a **minimax game**:

$$\min_{\theta} \max_{\phi} \pi \mathbb{E}_{p^*(\boldsymbol{x})}\left[\log D_\phi\left(\boldsymbol{x}\right)\right] + (1-\pi)\mathbb{E}_{p(\boldsymbol{z})}\left[\log\left[1 - D_\phi(f_\theta(\boldsymbol{z}))\right]\right]$$

  - Discriminator: aims to distinguish between (samples from) $p^*(\boldsymbol{x})$ and $p_\theta(\boldsymbol{x})$
    - ➢ Maximization

      // minimization of cross-entropy
  - Generator: aims to generate samples that are indistinguishable
    - ➢ Minimization

      // maximization of (lowest) cross-entropy

- This bilevel problem is typically tackled via alternating optimization.
- **Ratio loss** (discriminator loss) optimization:

$$\min_{\phi} \pi \mathbb{E}_{p^*(\boldsymbol{x})}\left[-\log D_\phi\left(\boldsymbol{x}\right)\right] + (1-\pi)\mathbb{E}_{p(\boldsymbol{z})}\left[-\log\left[1 - D_\phi(f_\theta(\boldsymbol{z}))\right]\right]$$

- **Generative loss** optimization:

$$\min_{\theta} \mathbb{E}_{p(\boldsymbol{z})}\left[\log\left[1 - D_\phi(f_\theta(\boldsymbol{z}))\right]\right]$$

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

Wasserstein Generative Adversarial Networks

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

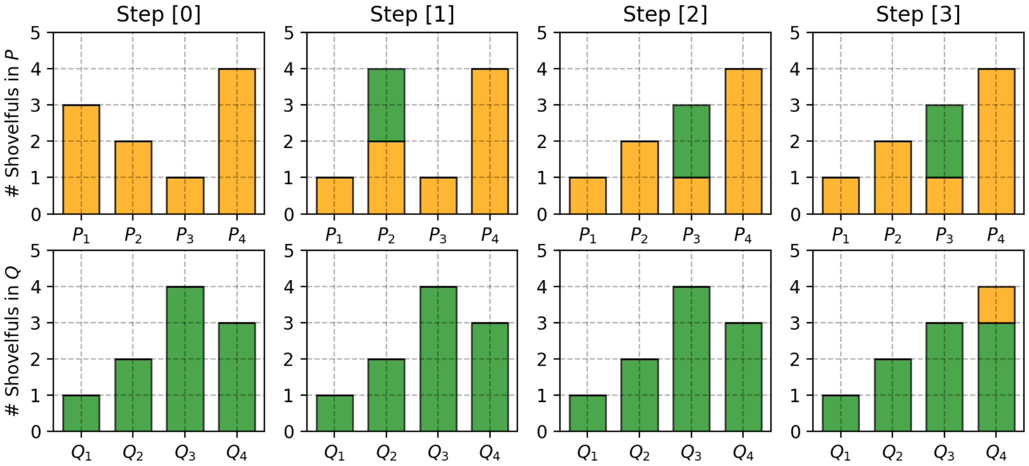
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
    for $t = 0, ..., n_{\text{critic}}$ **do**
        Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
        Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
        $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
        $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
        $w \leftarrow \text{clip}(w, -c, c)$
    end for
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
    $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Image source: Arjovsky, Chintala, Bottou, 2017.

Wasserstein distance

Wasserstein Distance is a measure of the distance between two probability distributions. It is also called Earth Mover's distance. It can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution.



From "https://arxiv.org/pdf/1904.08994.pdf".

$$P_1 = 3, P_2 = 2, P_3 = 1, P_4 = 4$$
$$Q_1 = 1, Q_2 = 2, Q_3 = 4, Q_4 = 3$$

If we label the cost to pay to make P_i and Q_i match δ_i, we would have $\delta_{i+1} = \delta_i + P_i - Q_i$ and in example:

$$\begin{aligned} \delta_0 &= 0 \\ \delta_1 &= 0 + 3 - 1 = 2 \\ \delta_2 &= 2 + 2 - 2 = 2 \\ \delta_3 &= 2 + 1 - 4 = -1 \\ \delta_4 &= -1 + 4 - 3 = 0 \end{aligned} \tag{1}$$

When dealing with the continuous probability domain, the distance formula becomes:

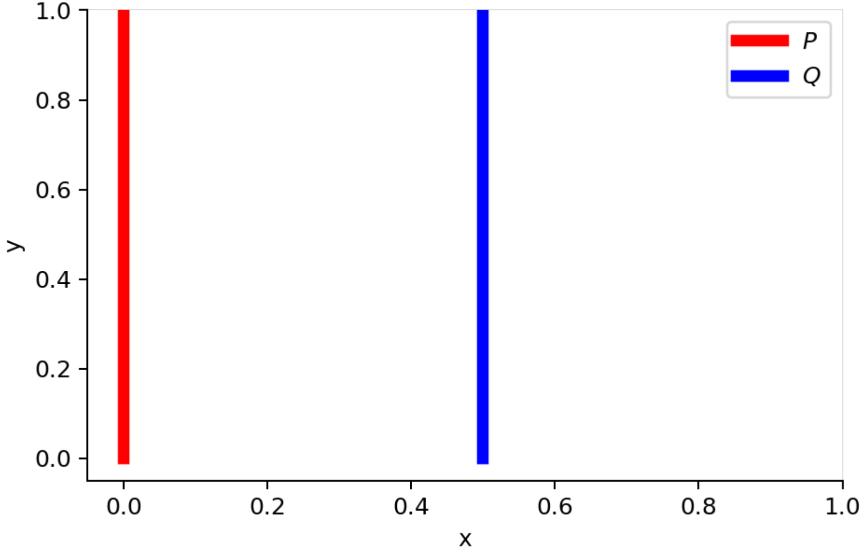
$$W(p^*(x), p(z)) = \inf_{\gamma \sim \Pi(p^*(x), p(z))} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2}$$

Image source: https://arxiv.org/pdf/1904.08994.pdf".

Why Wasserstein is better than JS or KL divergence?

Suppose we have two probability distributions, P and Q:

$$\begin{aligned} \forall(x, y) \in P, x = 0 \text{ and } y \sim U(0,1) \\ \forall(x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0,1) \end{aligned} \tag{3}$$



$$
\begin{aligned}
D_{KL}(P\|Q) &= \sum_{x=0, y\sim U(0,1)} 1\cdot\log\frac{1}{0} = +\infty \\
D_{KL}(Q\|P) &= \sum_{x=\theta, y\sim U(0,1)} 1\cdot\log\frac{1}{0} = +\infty \\
D_{JS}(P, Q) &= \frac{1}{2}\Big(\sum_{x=0, y\sim U(0,1)} 1\cdot\log\frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1\cdot\log\frac{1}{1/2}\Big) = \log 2 \\
W(P, Q) &= |\theta|
\end{aligned}
\tag{4}
$$

But when $\theta = 0$, two distributions are fully overlapped:

$$
\begin{aligned}
D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P, Q) = 0 \\
W(P, Q) = 0 = |\theta|
\end{aligned}
\tag{5}
$$

From "https://arxiv.org/pdf/1904.08994.pdf".

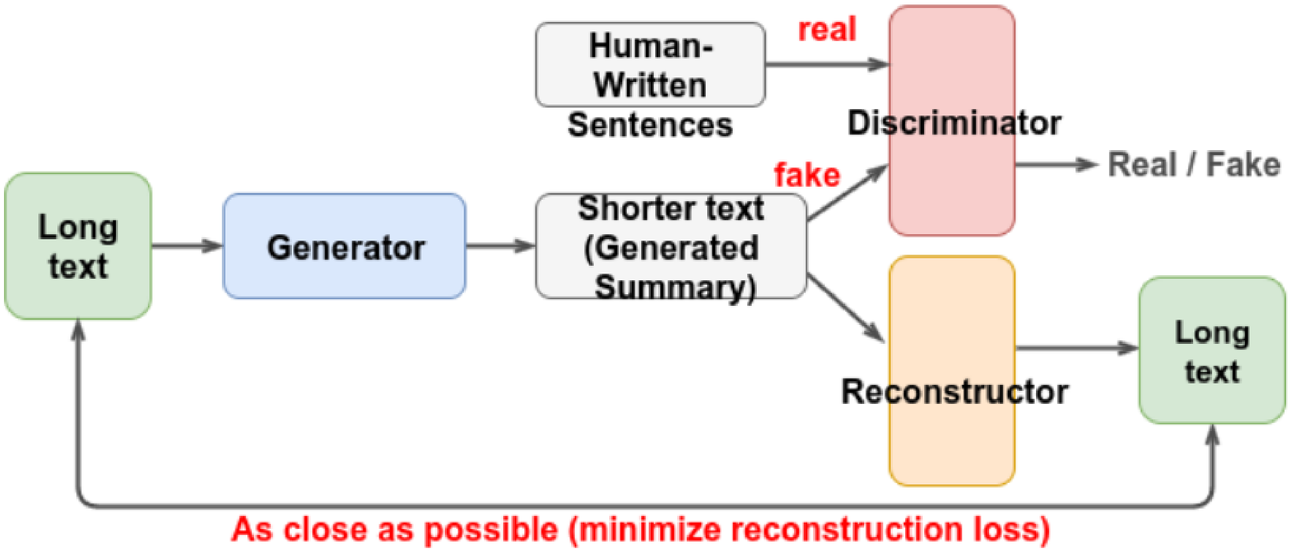
The differences in implementation for the WGAN

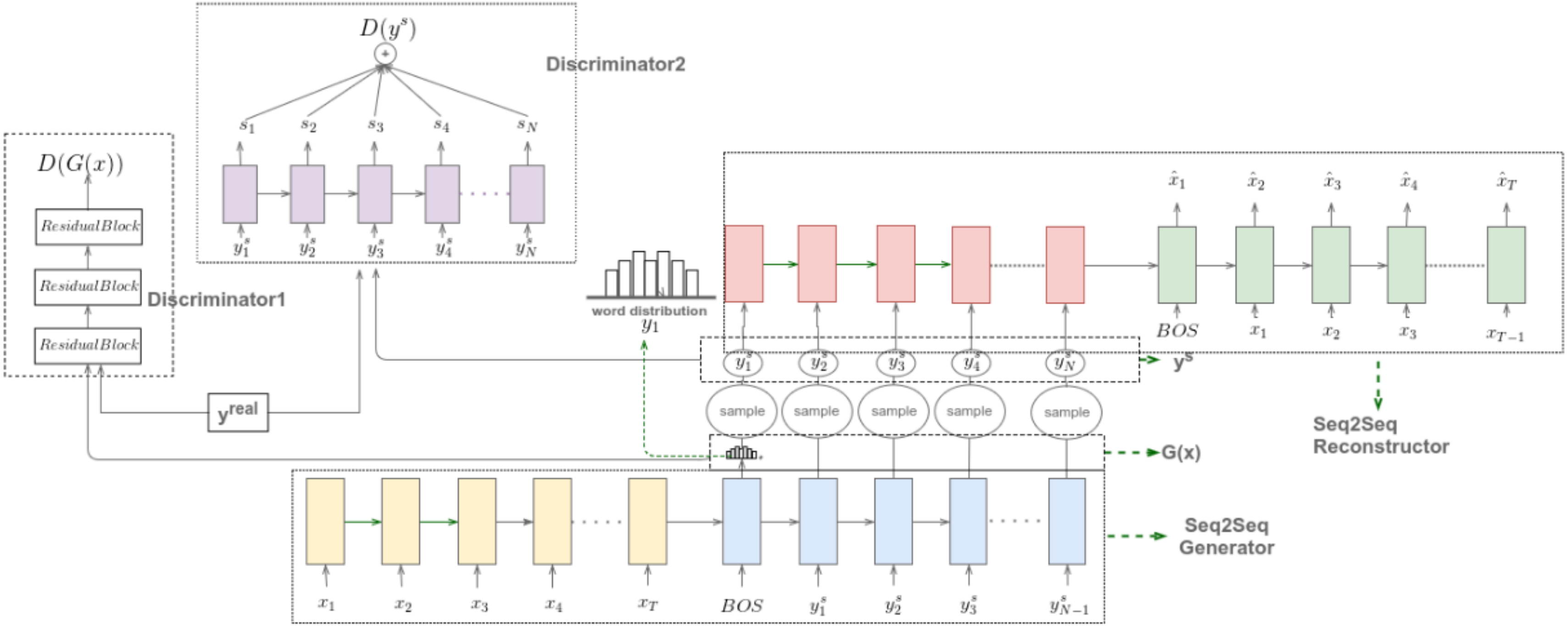
1. Use a linear activation function in the output layer of the critic model (instead of sigmoid).
2. Use -1 labels for real images and 1 labels for fake images (instead of 1 and 0).
3. Use Wasserstein loss to train the critic and generator models.
4. Constrain critic model weights to a limited range after each mini batch update (e.g. [-0.01,0.01]).
5. Update the critic model more times than the generator each iteration (e.g. 5).
6. Use the RMSProp version of gradient descent with a small learning rate and no momentum (e.g. 0.00005).

From "https://arxiv.org/pdf/1904.08994.pdf".

Putting all together

@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich




$D(y^s)$
Discriminator2
s_1 s_2 s_3 s_4 s_N
y_1^s y_2^s y_3^s y_4^s y_N^s
$D(G(x))$
ResidualBlock
ResidualBlock
ResidualBlock
Discriminator1
word distribution
y_1
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$ $\hat{x}_T$
BOS x_1 x_2 x_3 x_{T-1}
y^s
sample
y^{real}
G(x)
Seq2Seq Reconstructor
Seq2Seq Generator
x_1 x_2 x_3 x_4 x_T BOS y_1^s y_2^s y_3^s y_{N-1}^s

@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

Key learnings from the paper

- Keras very flexible framework which is underestimated in community
- Keras and tensorflow is very unstable from version to version
- migration from python2 to python3 is nightmare

Critics of authors

- Paper is very bad detiled, there are missed part
- Code provided by authors is hardly reprodusable and not documented
- ...

@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

Thank you for your attention

Any questions?

References