@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

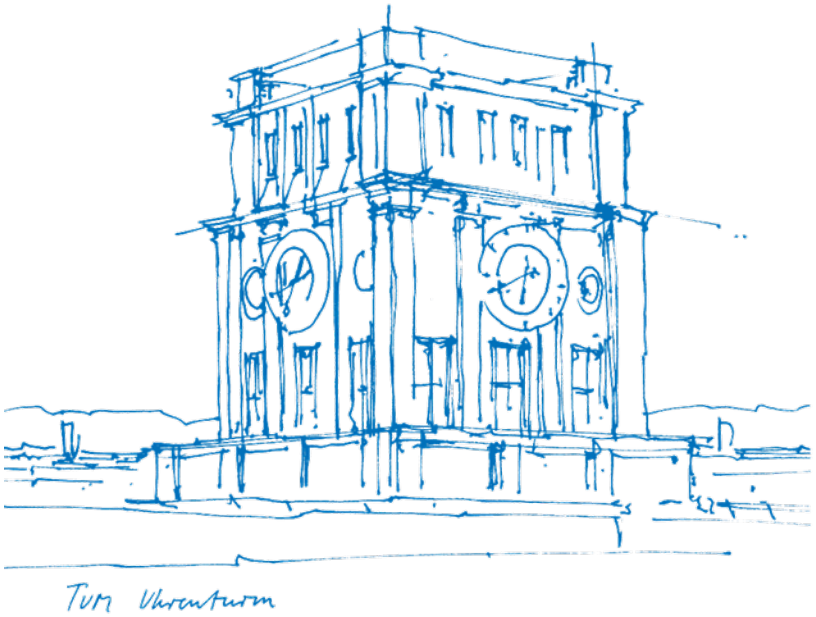
Generative Adversarial Networks

Denys Lazarenko[1], **Kamilia Mullakaeva**[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 (why it is important)

- Volume of text data
  - The amount of online-text grows really fast and it is hard to monitor changes;
  - Automatic document classification, sentiment classification is important to companies to monitor changes connecting to their products and for analysis of customer feedback;
- Volume of documents, which are unpaired with summaries
  - movie reviews
  - lecture transcripts

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-

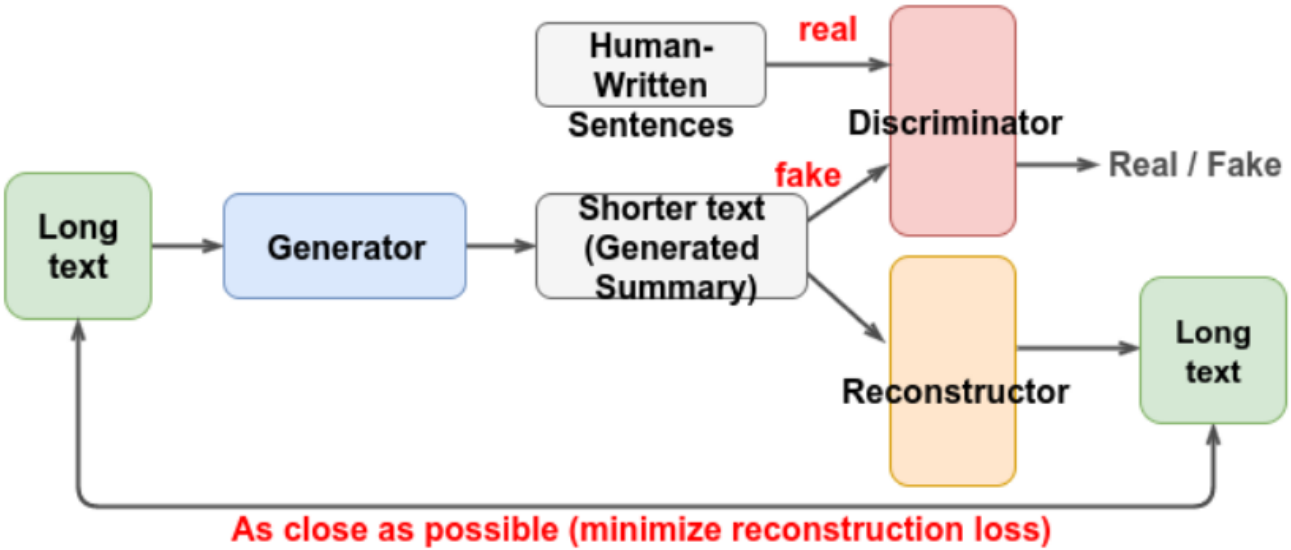

Human-Written Sentences
real
Discriminator
Real / Fake
fake
Long text
Generator
Shorter text (Generated Summary)
Reconstructor
Long text
As close as possible (minimize reconstruction loss)

Figure: From

English Gigaword

Source Text: global one communications will launch its global intranet <unk> -lrb- virtual private networks -rrb- service in new zealand by the end of the year , the new zealand infotech weekly reported monday .	
Ground Truth: international intranet service destined for new zealand	**(A-1)Supervised Result:** global one communications to launch global networks in new zealand
(C-2)WGAN: global communications launch global service in new zealand	**(C-3)Adversarial REINFORCE:** global communications launch global virtual private networks in new zealand
(E-2)WGAN: global one communications will launch its global networks -rrb- service	**(E-3)Adversarial REINFORCE:** global one communications will launch its global virtual networks

Figure: From

Dataset have to be preprocess, it contains a special token <unk> inside the train and validation sets. <unk> is inconsistent with what is used in the test set — UNK.

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.

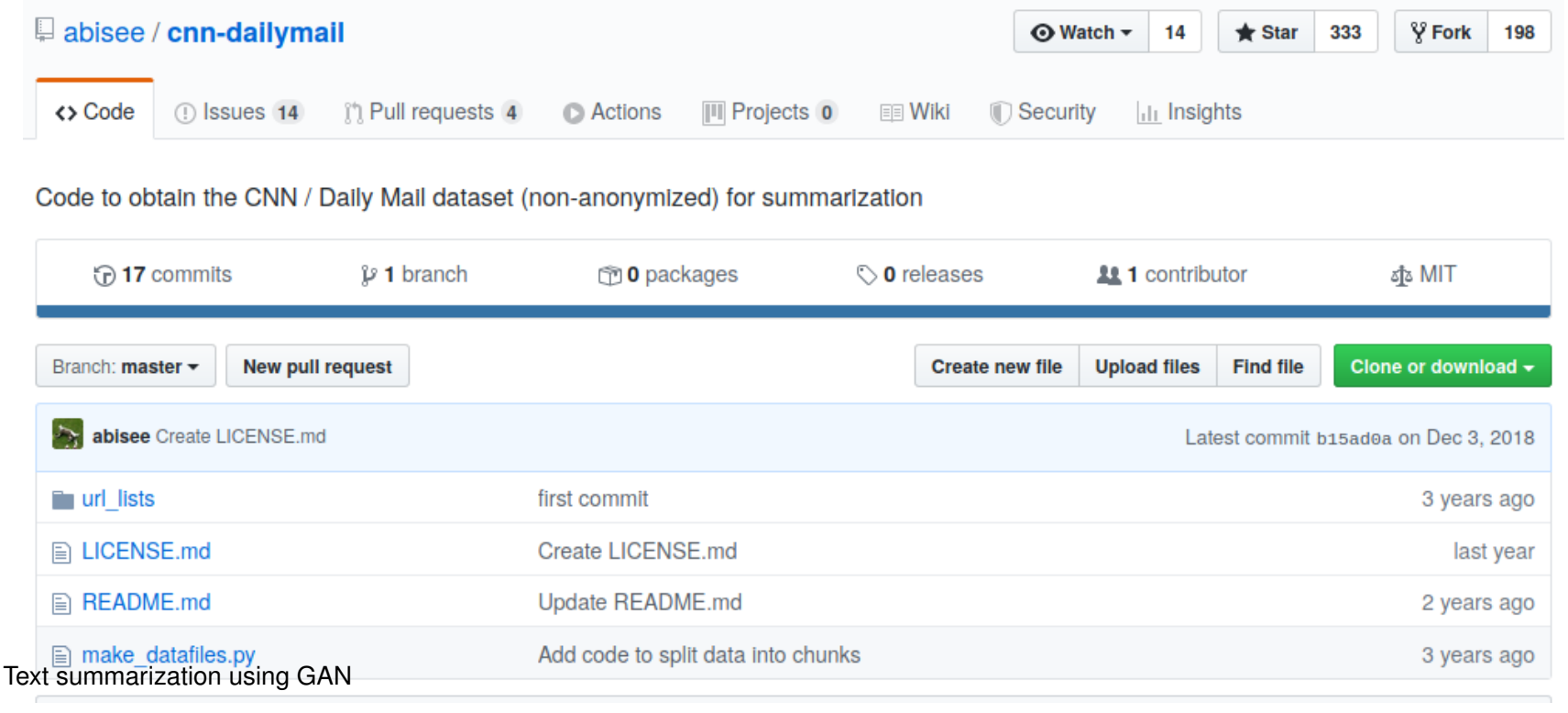
Daily Mail

- 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.

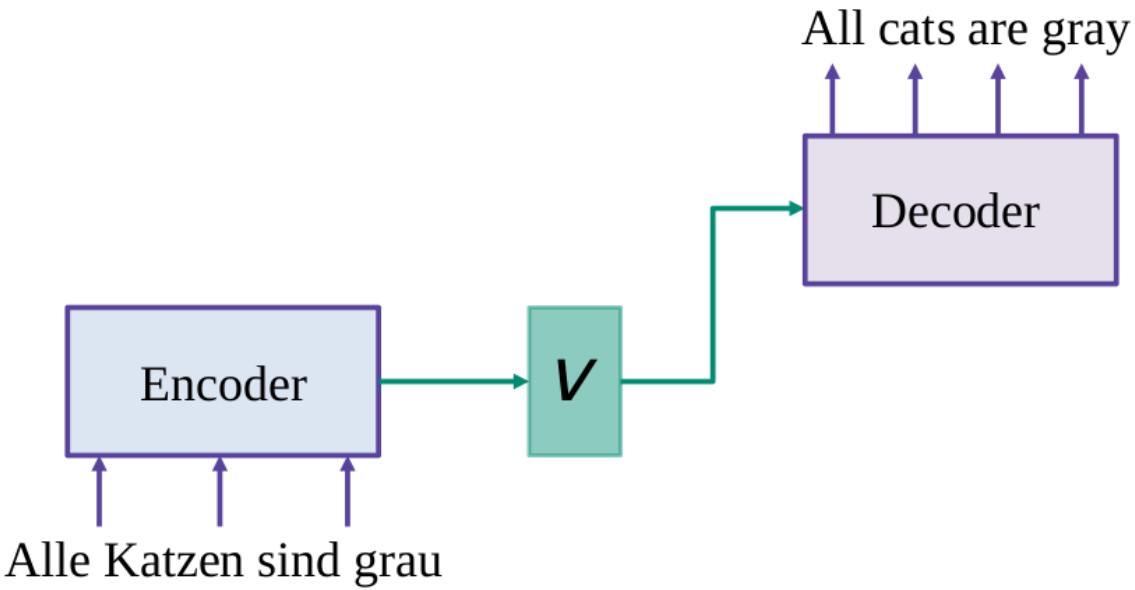
Kyunghyun Cho

CNN / Daily Mail dataset

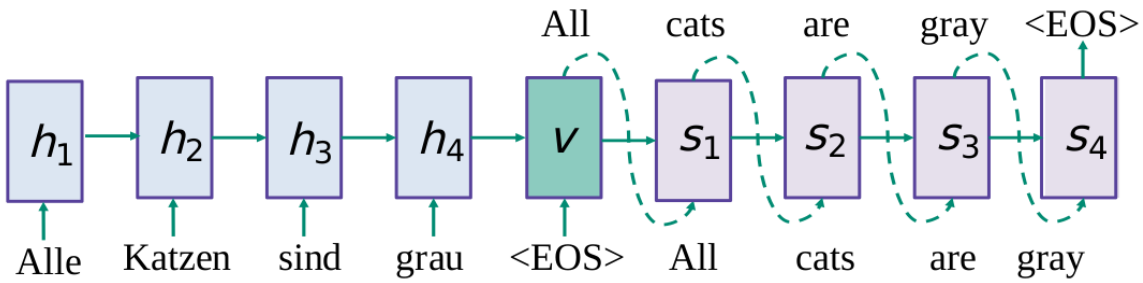


Encode-decoder architectur

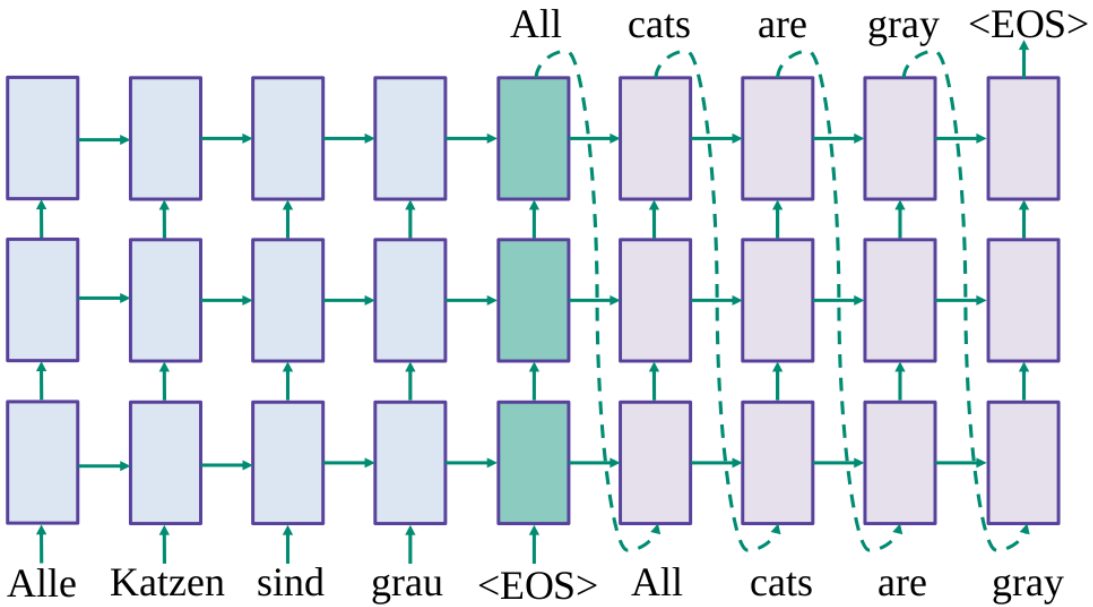
Sequence to sequence



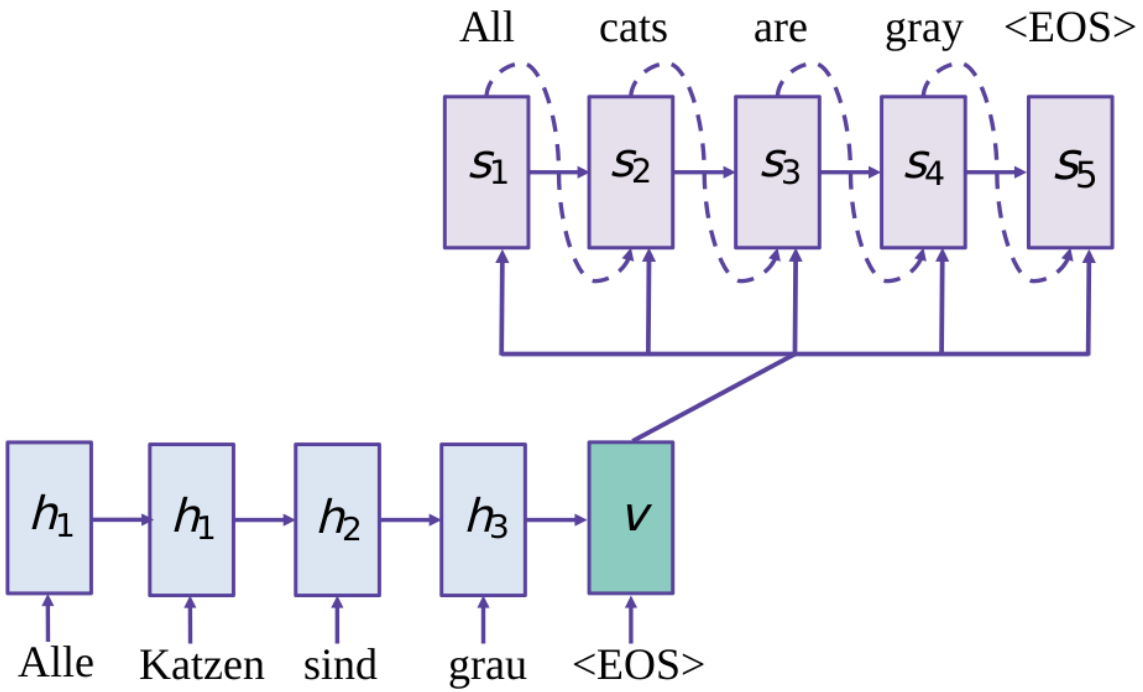
From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.



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



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



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

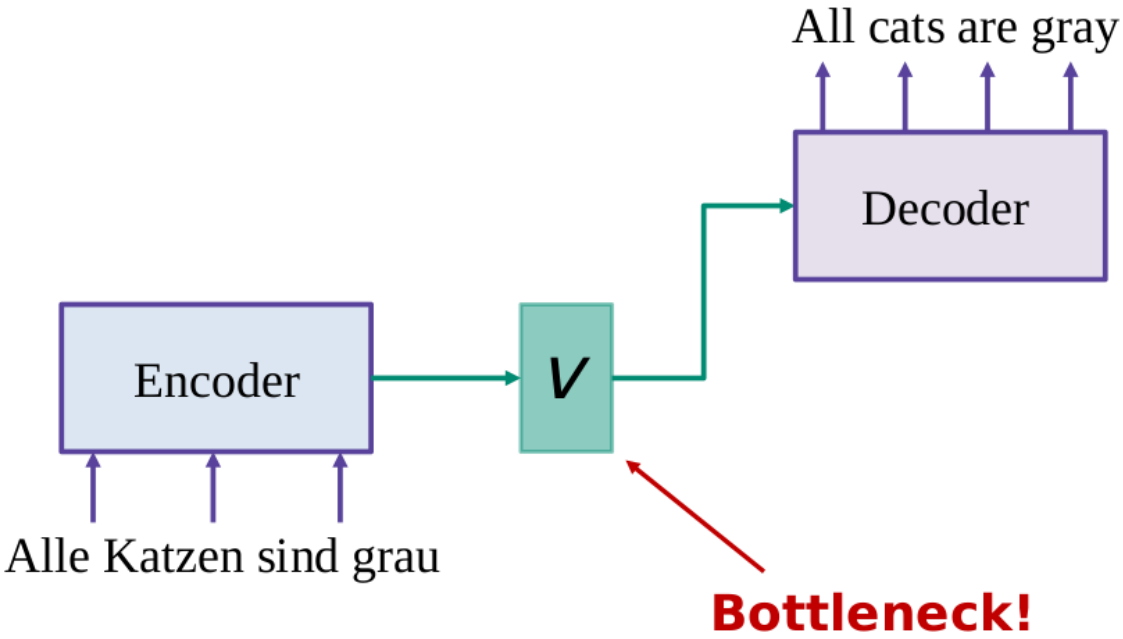
Sequence modeling task

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

where y_i and x_i are sequences of words.

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

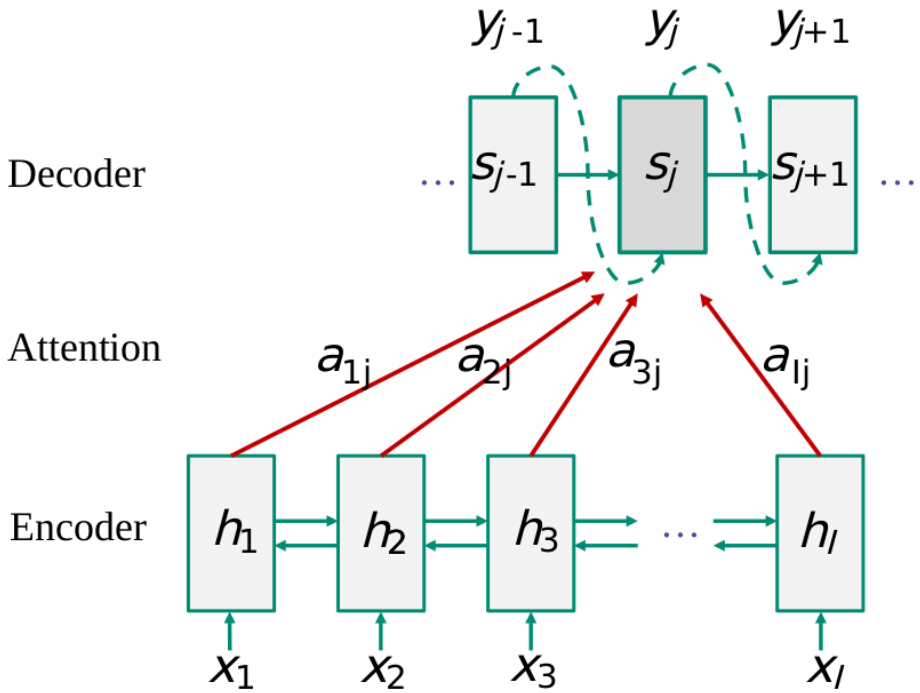
From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.



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

Attention mechanism

Attention mechanism



Encoder that has *h* states and decoder that has some *s* states. a_{ij} denotes some weights that will say us whether it is important to look there or here.

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

- 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 (α_{ij} is an average of encoder states):

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

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

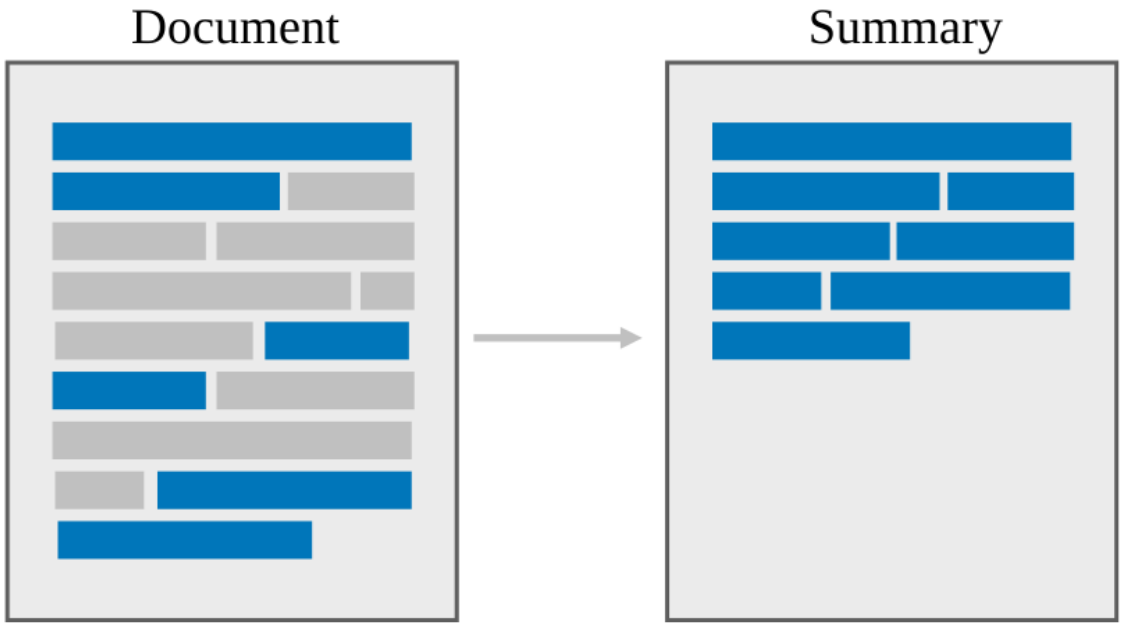
$$\mathbb{P}\left[y_1, \ldots y_J | x_1, \ldots, x_I\right] = \prod_{j=1}^{J} \mathbb{P}\left[y_j | v_j, y_1, \ldots, y_{j-1}\right]$$

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

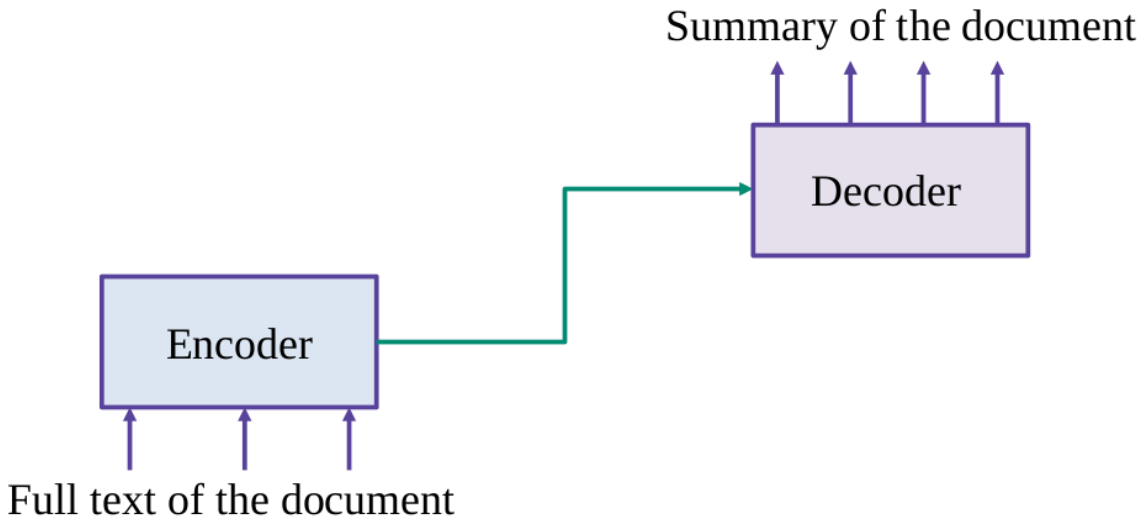
From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Summarization with pointer-generator networks

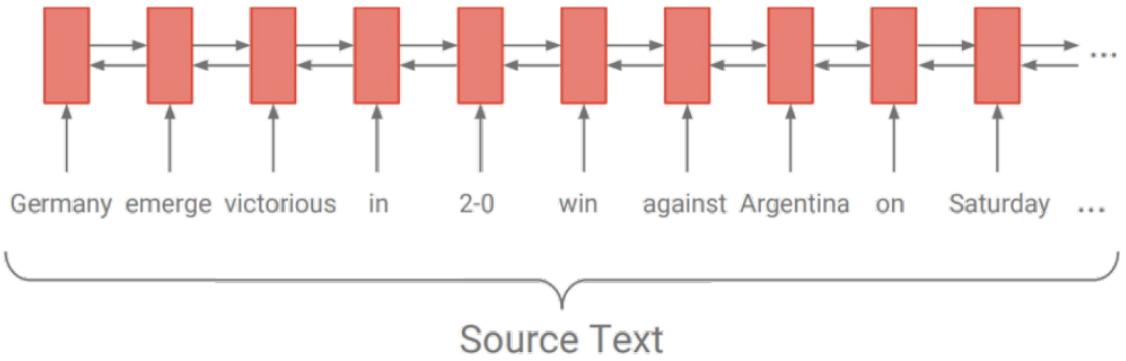
Summarization



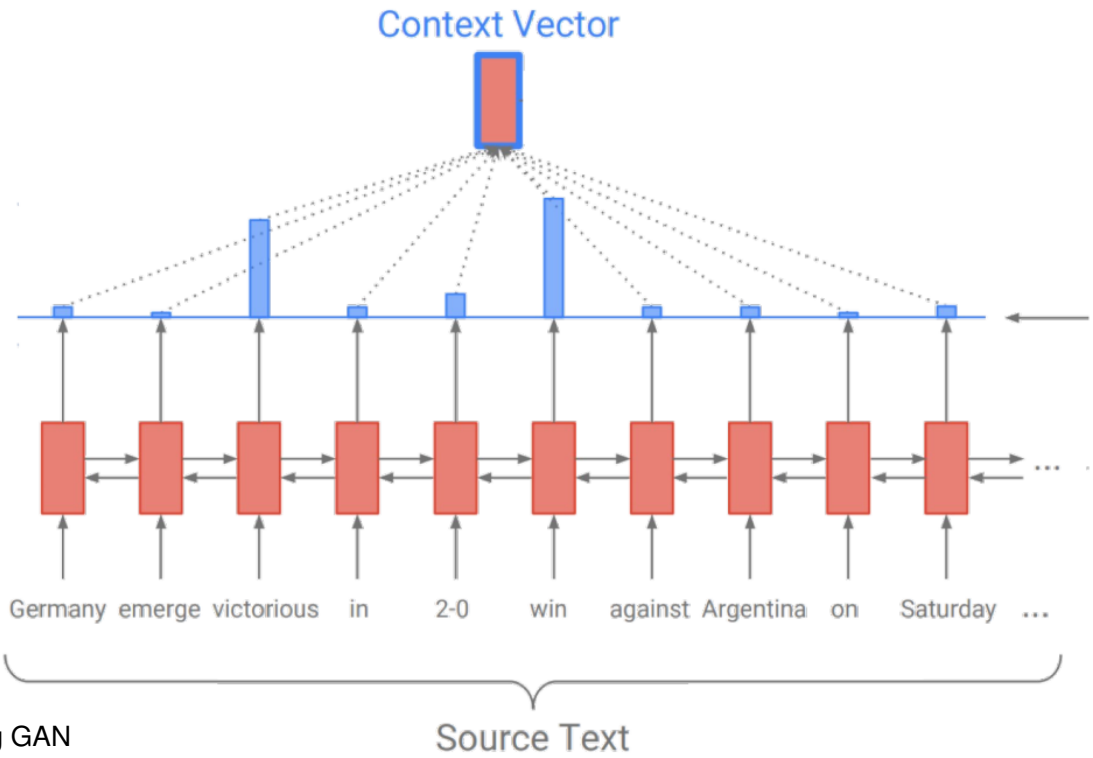
Summarization



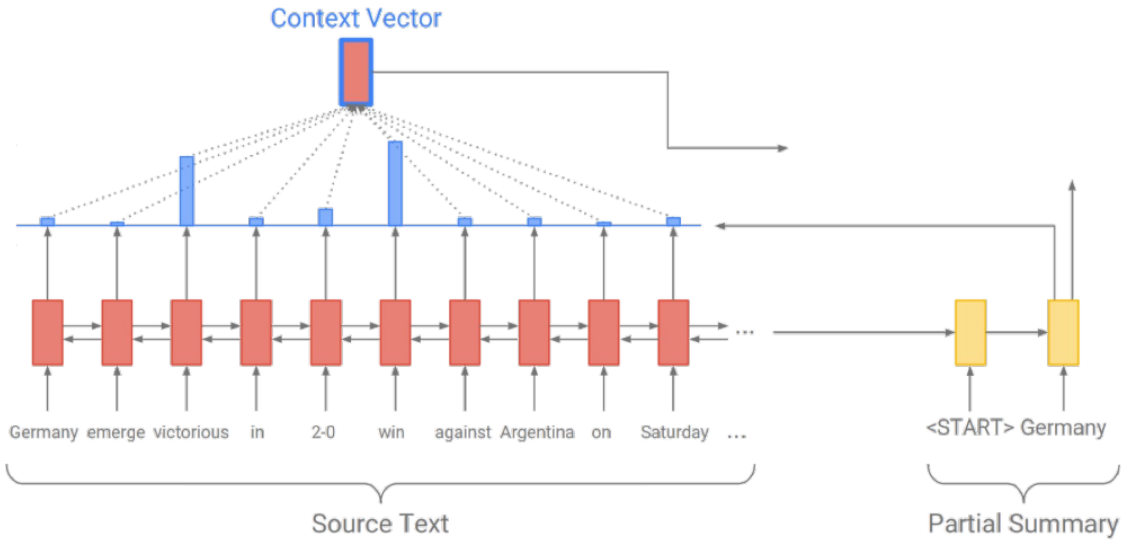
Seq2seq + attention



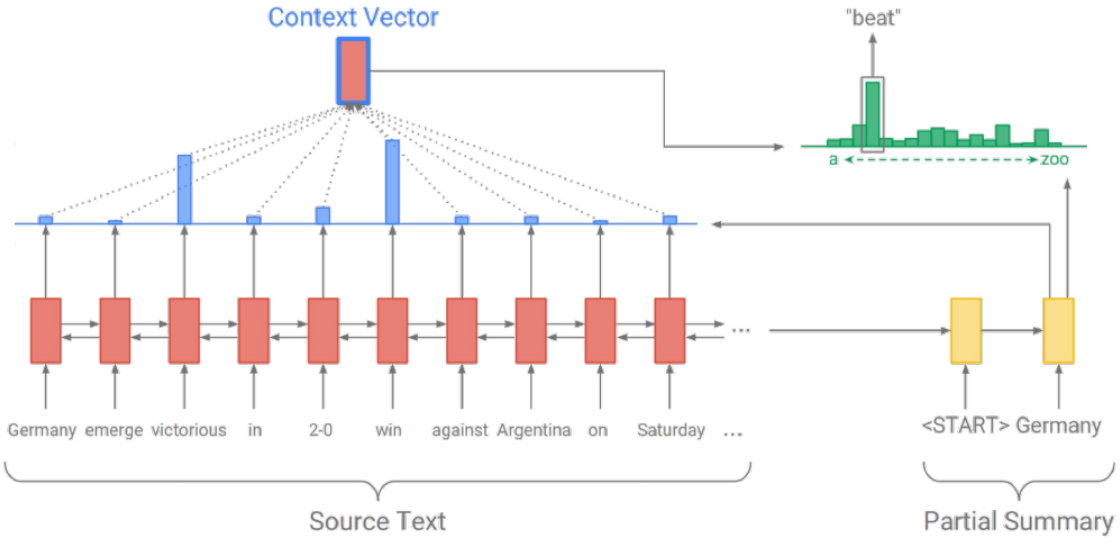
Seq2seq + attention



Seq2seq + attention



Seq2seq + attention



1. Attention distribution (over source positions):

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

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

2. Vocabulary distribution (generative model):

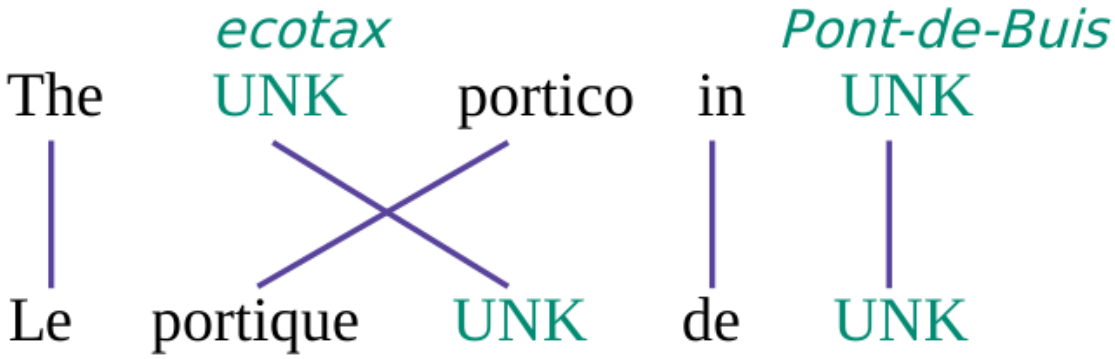
$$v_j = \sum_i p_i^j h_i \text{ (context vector)}$$

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

where V', V are some transformation

Copy mechanism

What do we do with OOV words?

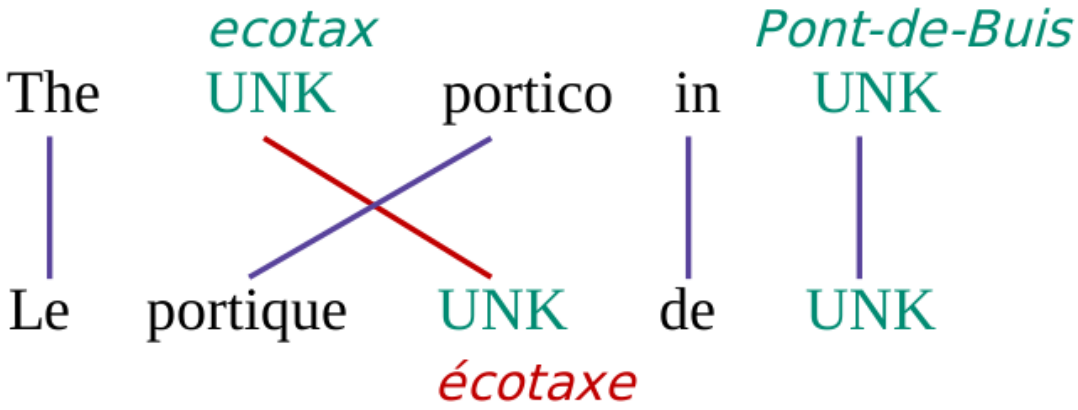


From "Natural Language Processing", A. Potapenko et.al. HSE, 2019.

Copy mechanism

What do we do with OOV words?

Look-up in a dictionary

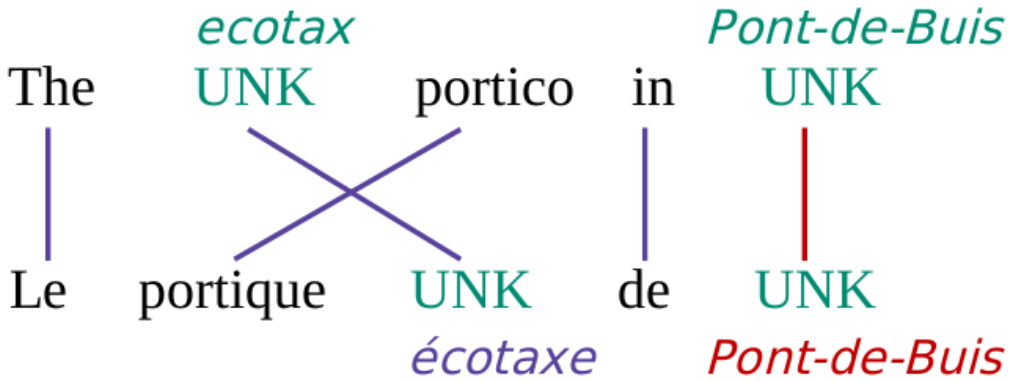


From "Natural Language Processing", A. Potapenko et.al. HSE, 2018

Copy mechanism

What do we do with OOV words?

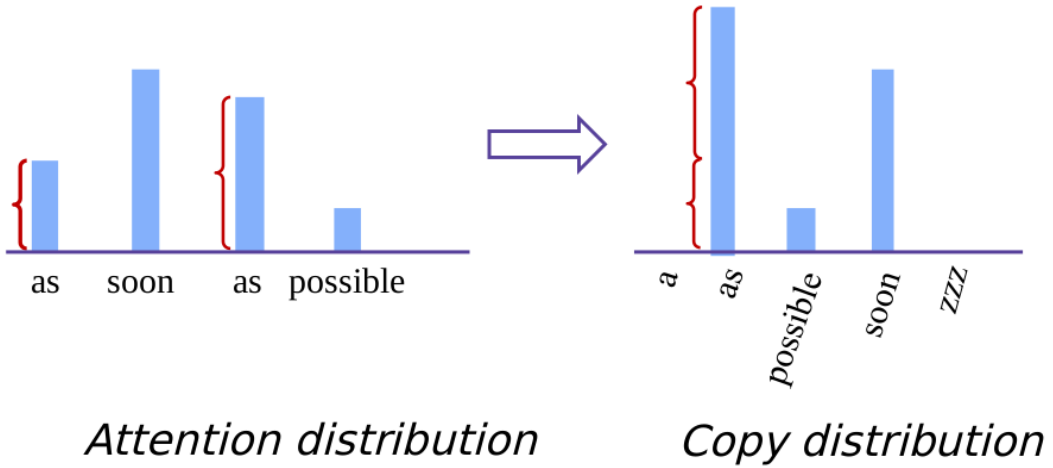
Look-up in a dictionary **Copy name**



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

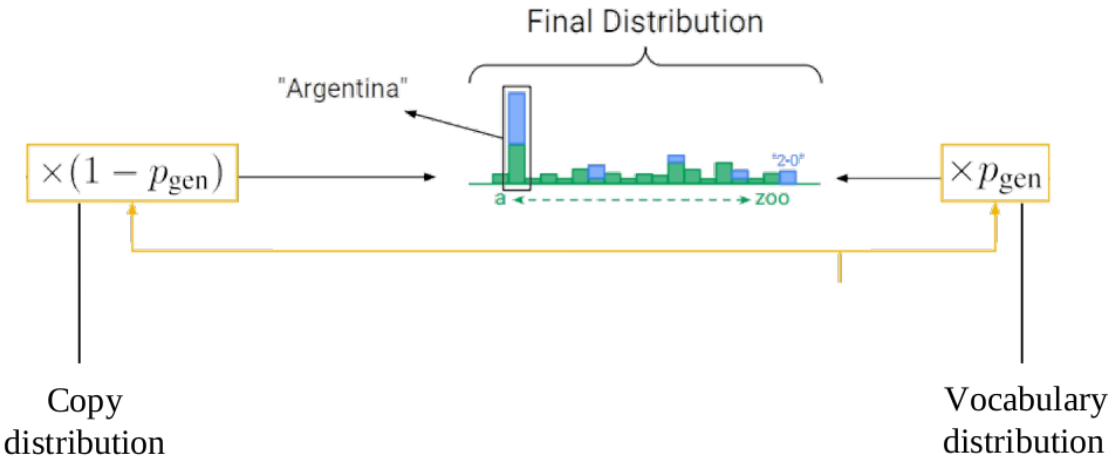
3. Copy distribution (over words from source):

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



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

Pointer-generator network



4. Final distribution:

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

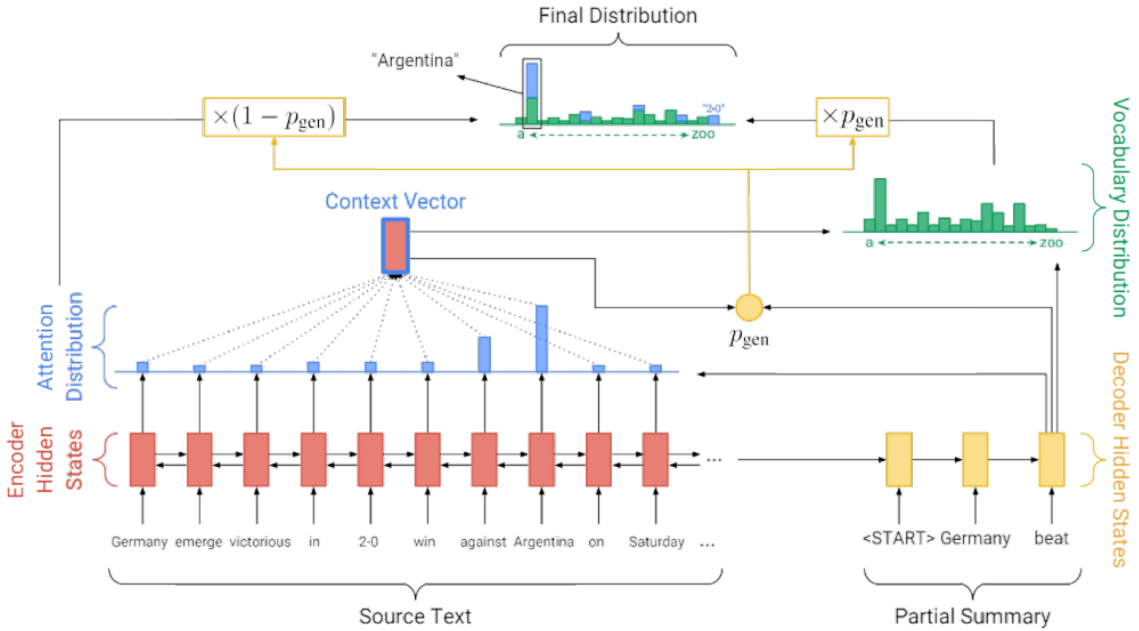
$$p_{gen} = sigmoid(w_v^T v_j + w_s^T s_j + w_x^T y_{j-1} + b_{gen}),$$

where v_j is the context vector, s_j - decoder states, y_{j-1} - current inputs to the decoder

5. Training (cross-entropy loss):

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

Pointer-generator network

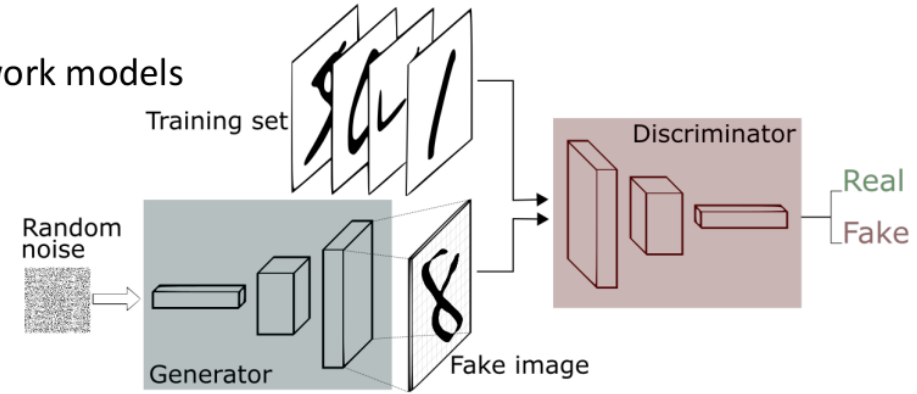


Comparison of the models

	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**

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.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

From

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

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

Some problems in GANs

- Vanishing gradient. (When the discriminator is perfect, we are guaranteed with D(x)=1,xpr and D(x)=0,xpg. Therefore the loss function L falls to zero and we end up with no gradient to update the loss during learning iterations.) Solution: reverse the loss function from min to max.
- Mode collapse. During the training, the generator may collapse to a setting where it always produces same outputs. (The probability distribution of real data is multymodal. Reason: the main purpuse of the generator to learn to fool the disc(fake/real), it is not have to be diverse)
- Low dimensional supports

Wasserstein Generative Adversarial Networks

Improving the GAN training-Use Better Metric of Distribution Similarity

Essentially the loss function of GAN quantifies the similarity between the generative data distribution and the real sample distribution by JS divergence when the discriminator is optimal and equal to 1/2.

$$L(G, D^*) = 2D_{JS}(p_r \| p_g) - 2\log 2$$

This metric fails to provide a meaningful value when two distributions are disjoint.
Wasserstein metric is proposed to replace JS divergence because it has a much smoother value space.

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

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

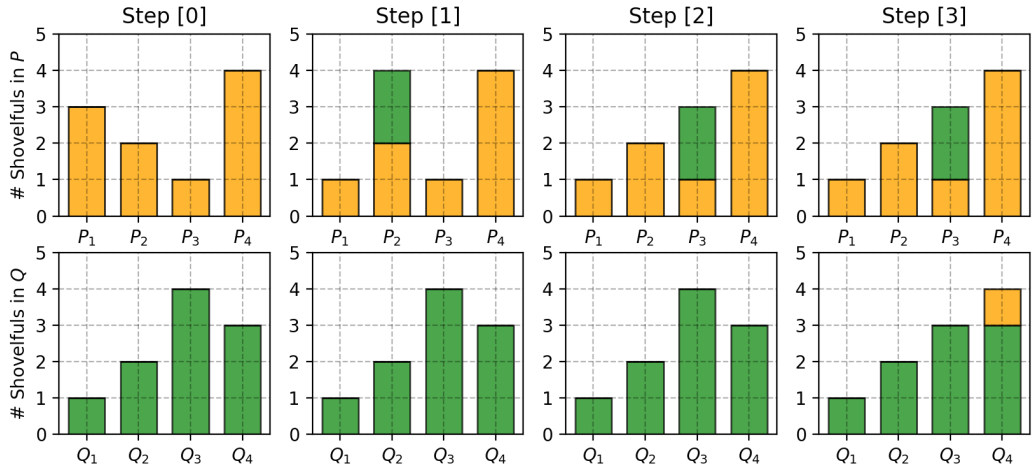
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

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:

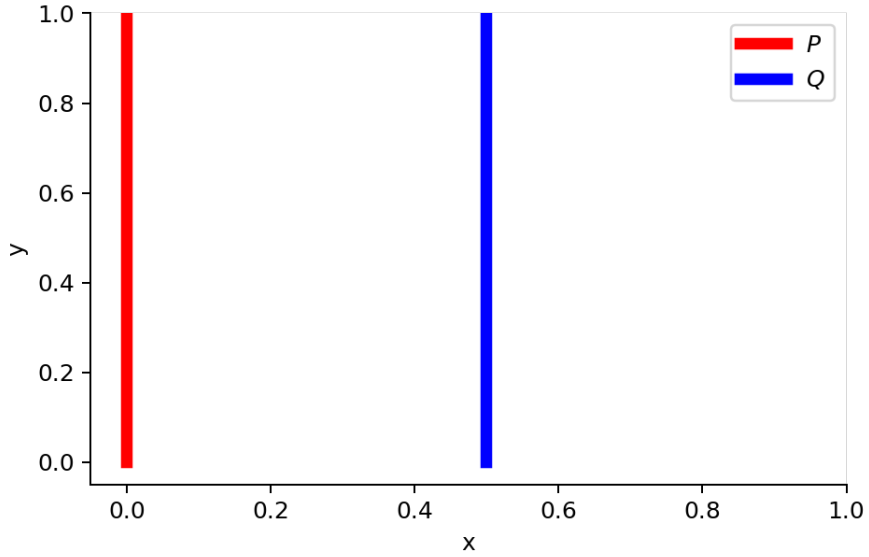
$$W(p^*(x), p(z)) = \inf_{\gamma \sim \Pi(p^*(x), p(z))} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2}$$

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:

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



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

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

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

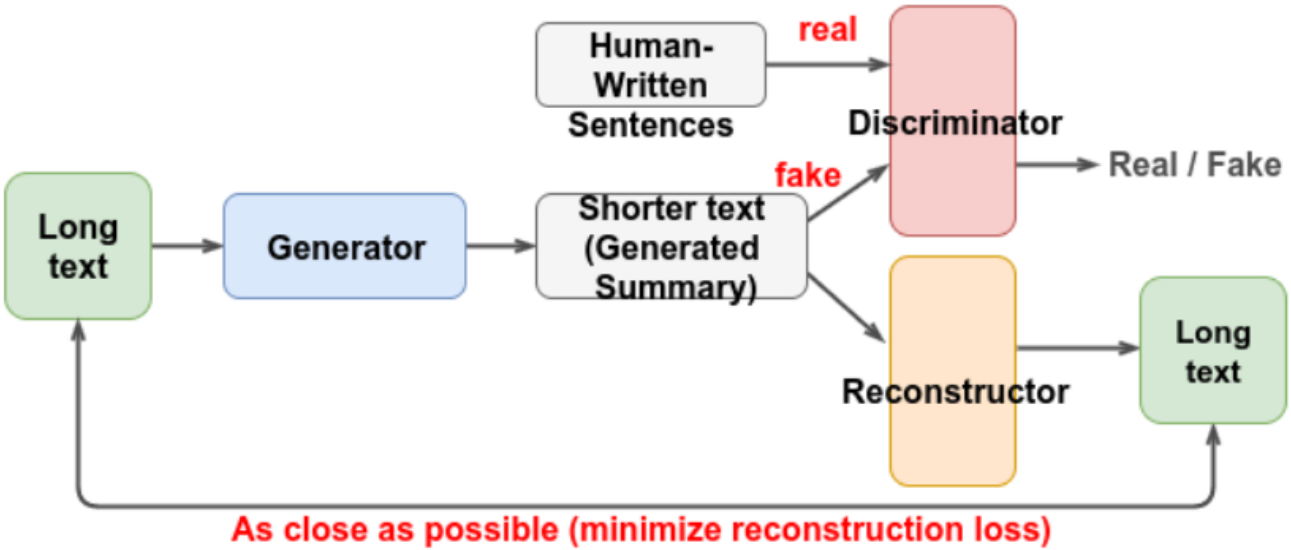
From "https://arxiv.org/pdf/1904.08994.pdf".

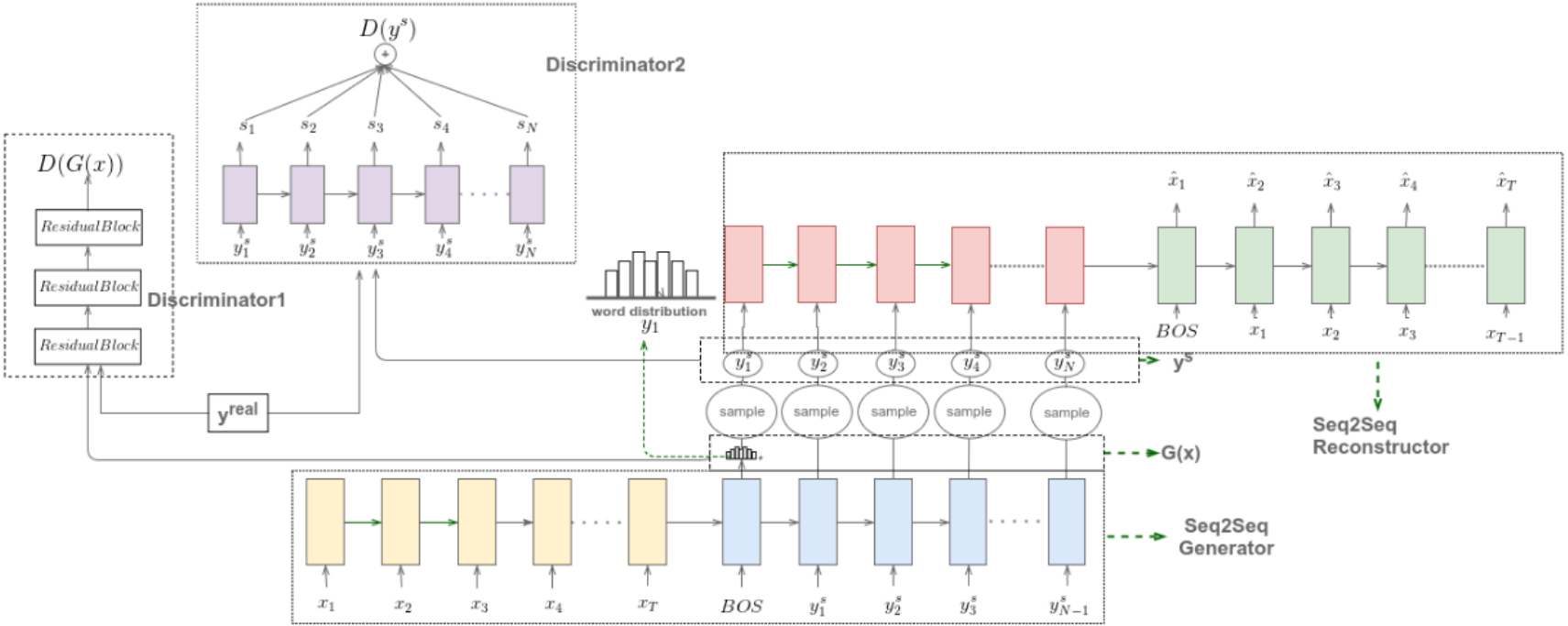
The differences in implementation for the WGAN

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


Human-Written Sentences
real
Discriminator
Real / Fake
Long text
Generator
Shorter text (Generated Summary)
fake
Reconstructor
Long text
As close as possible (minimize reconstruction loss)


$D(y^s)$
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
y^{real}
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_1^s y_2^s y_3^s y_4^s y_N^s
y^s
sample sample sample sample sample
G(x)
Seq2Seq
Reconstructor
x_1 x_2 x_3 x_4 x_T BOS y_1^s y_2^s y_3^s y_{N-1}^s
Seq2Seq
Generator

Code

Paper results

- Union of the best models: WGAN(IWGAN) is an improved version of GAN which solves problems with training and stability; Pointer-network is one of the best models for a summarization task.
- GAN setup gives a possibility to train the model in an unsupervised way. Therefore it is possible to use big datasets in order to increase quality.
- Pointer-network did not gives guarantee that:
  - summary will be human readable and that it will be key information
  - we need to configure parameters in order to get short summaries
- Discriminator should solve the problem being human readable
- Reconstructor solves the problem making summaries short (If the information will be not the key one than we will not be able to reconstruct it correctly, like in AutoEncoder)

Key learnings from the paper

- Embed, encode, attend, predict: The powerful deep learning formula for state-of-the-art NLP models.
- 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.
- Human readable code using code patterns speed up development and understanding.

Critics of authors

- Paper is very bad detailed, there are missed parts.
- Code provided by authors is hardly reproducible and not documented.
- The final results evaluation is made with many assumptions which does not give a possibility to compare it with baseline results.
- contribution from the paper is not significant enough to the field.

Future work

- Make an architecture more simple which will speed up training and future modification for the model:
  - Check whether model performs better with another type of network for Generator and Reconstructor.
  - Measure performance with Reconstructor and without.

Thank you for your attention

Any questions?

References