Intelligent and Applied Algorithms

Intelligent Algorithms have emerged as pivotal components in modern data science, significantly enhancing capabilities in text modeling, data synthesis, and representation learning. This document presents foundational concepts and practical insights into three prominent Intelligent Algorithms: Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Sequence-to-Sequence (Seq2Seq) models. We highlight their mathematical frameworks, typical use cases, and implementation strategies using Python-based tools.

This document is an extension of the research and lecture notes completed at Johns Hopkins University, Whiting School of Engineering, Engineering for Professionals, Artificial Intelligence Master's Program, Computer Science Master's Program and Data Science Master's Program.

Contents

1 Introduction to Intelligent Algorithms

Intelligent Algorithms represent a powerful subset of machine learning techniques that enable computers to perform complex tasks, previously considered exclusive to human intelligence. Among these algorithms, Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Sequence-to-Sequence (Seq2Seq) models stand out due to their versatility and robustness in addressing various data science problems. These algorithms are particularly adept at generating synthetic yet realistic data, uncovering latent structures within datasets, and effectively managing sequential data transformations. Understanding their core principles and mathematical foundations allows practitioners to leverage their strengths to solve real-world challenges.

The subsequent sections of this document are structured to provide comprehensive insights. The **Types of Intelligent Algorithms Problems** section outlines specific problem areas where these algorithms excel, emphasizing their suitability for various data science tasks. The **Mathematical Foundations for Intelligent Algorithms Representation and Basics** section delves into the underlying theoretical frameworks that define each algorithm, providing clarity on their mathematical formulations. The **Intelligent Algorithms Using Python Packages** section offers practical guidance on leveraging Python libraries for efficient implementation and prototyping of GANs, VAEs, and Seq2Seq models. Lastly, the **Summary** synthesizes key takeaways, reinforcing the significance of mastering these Intelligent Algorithms within contemporary data science workflows.

2 Types of Intelligent Algorithms Problems

Intelligent Algorithms primary problems address text generation, probabilistic modeling, and sequential data transformations. We focus specifically on Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Sequence-to-Sequence (Seq2Seq) models, given their broad applications and relevance in data-driven tasks.

2.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are leveraged to solve data science problems involving realistic data synthesis and augmentation. In GANs, two neural networks, namely a generator and discriminator, are trained simultaneously in an adversarial setup. The generator network synthesizes data to mimic the statistical properties of real datasets, whereas the discriminator aims to classify inputs as real or generated. GANs are particularly beneficial in creating synthetic yet realistic datasets, augmenting limited data sources, and improving the robustness of predictive models in text-based applications.

2.2 Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) address problems centered on probabilistic modeling and representation learning. VAEs encode input data into latent probabilistic distributions and subsequently reconstruct the original input from these latent spaces. This makes VAEs highly valuable for data science tasks involving data compression, dimensionality reduction, anomaly detection, and unsupervised learning scenarios. Specifically, for text data, VAEs facilitate the generation of diverse textual data and uncover meaningful latent representations for downstream analytical tasks.

2.3 Sequence-to-Sequence (Seq2Seq) Models

Sequence-to-Sequence (Seq2Seq) models are essential for problems involving the transformation of sequential data from one form to another. These models utilize encoder-decoder architectures enhanced with attention mechanisms, effectively capturing and modeling sequential relationships within data. Seq2Seq models are fundamental in data science applications such as machine translation, text summarization, dialogue systems, and predictive modeling of sequential events. Their ability to handle context-sensitive transformations makes them invaluable for sophisticated natural language processing tasks encountered in data science practice.

3 Mathematical Foundations for Intelligent Algorithms Representation and Basics

The mathematical underpinnings of Intelligent Algorithms—particularly Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Sequence-to-Sequence (Seq2Seq) models—are essential for their effective implementation and understanding.

3.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks are based on a minimax game involving two neural networks: a generator G and a discriminator D. The objective function of GANs is expressed mathematically as:

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

In this equation, x denotes real data drawn from the data distribution $p_{\text{data}}(x)$, and z represents noise sampled from the prior distribution $p_z(z)$. The discriminator seeks to maximize this objective by accurately differentiating real from synthetic data, while the generator aims to minimize it by producing data indistinguishable from real samples.

3.2 Variational Autoencoders (VAEs)

Variational Autoencoders leverage variational inference and Bayesian statistical methods. Their primary objective is to maximize the Evidence Lower BOund (ELBO), given mathematically as:

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - KL(q(z|x) \| p(z))$$

Here, $p(z)$ denotes the prior distribution over latent variables, $q(z|x)$ represents the approximate posterior distribution learned by the encoder, and $p(x|z)$ indicates the likelihood distribution parameterized by the decoder. The Kullback-Leibler (KL) divergence acts as a regularization term, encouraging meaningful latent representations and high-quality reconstructions.

3.3 Sequence-to-Sequence (Seq2Seq) Models

Sequence-to-Sequence models employ probabilistic methods to model sequential data, specifically by maximizing the conditional probability of generating output sequences $\mathbf{y}$ given input sequences $\mathbf{x}$:

$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p(y_t | y_{<t}, \mathbf{x})$$

In this formulation, y_t represents the token generated at time step t, and $y_{<t}$ refers to the tokens generated in previous steps. Training typically involves minimizing the Cross-Entropy loss function, defined as:

$$\mathcal{L}_{\text{Seq2Seq}} = -\sum_{t=1}^{T} \log p(y_t | y_{<t}, \mathbf{x})$$

This approach effectively allows the model to learn complex sequential dependencies and generate coherent, contextually relevant outputs.

4 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs), introduced by Goodfellow et al. [2], are powerful generative models that learn to produce realistic data by training two competing neural networks: a generator and a discriminator. The generator aims to generate realistic synthetic data, while the discriminator attempts to distinguish synthetic data from real data. In the context of regression analysis and text data modeling, GANs leverage deep sequential models, such as recurrent neural networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformers, to generate coherent and contextually appropriate text sequences.

4.1 Mathematical Formulations

A Generative Adversarial Network (GAN) comprises two competing neural networks: a generator G and a discriminator D. These networks are trained simultaneously in a competitive, adversarial setting. The training process is formulated as a two-player minimax game, where the generator aims to produce data indistinguishable from real samples, while the discriminator attempts to correctly classify samples as either real or synthetic.

The core minimax objective of a GAN is mathematically expressed as follows:

$$\min_G \max_D \mathbb{E}x \sim p\text{data}(x)[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{1}$$

Here, x represents real data samples drawn from the true data distribution $p_{\text{data}}(x)$, and z represents noise vectors drawn from a prior distribution $p_z(z)$, typically a Gaussian or uniform distribution. The discriminator $D(x)$ outputs a probability indicating the likelihood of the data being real, while $G(z)$ represents the generator's transformation of noise into synthetic data samples.

Specifically, for modeling text data, the generator G transforms the latent noise vector z into sequences of tokens that resemble real textual data. This transformation typically employs advanced sequential architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, or Transformer-based models, which capture temporal dependencies and contextual semantics inherent in language.

Formally, given a latent vector $z \in \mathbb{R}^{d_z}$, the generator outputs a sequence $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_T)$, where each $\tilde{x}_t$ is a token drawn from the vocabulary distribution conditioned on the previously generated tokens:

$$p(\tilde{x}t|\tilde{x}< t, z; \theta_G) \tag{2}$$

The discriminator D, also leveraging sequential architectures, outputs a scalar probability representing the likelihood of the input sequence x being real or synthetic:

$$D(x; \theta_D) = p(y = \text{real}|x; \theta_D) \tag{3}$$

Here, θ_G and θ_D denote the parameters of the generator and discriminator, respectively. The parameters of both models are updated iteratively through gradient descent methods to optimize the aforementioned minimax objective, driving the generator toward producing increasingly realistic sequences of text.

4.2 Algorithm Explanation

Algorithm 5 outlines the training process for GANs specifically adapted to text data. The generator and discriminator employ advanced sequence models such as multi-layer LSTM networks with residual connections and batch normalization or Transformer-based architectures featuring self-attention mechanisms. The discriminator classifies sequences as real or synthetic, guiding the generator towards generating increasingly realistic text.

Algorithm 1 GAN for Modeling Text Data with Deep Sequence Models

Require: Text corpus $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$, noise prior $p_z(z)$, number of training steps T, learning rate α
Ensure: Trained Generator G and Discriminator D

function TrainTextGAN($\mathbf{X}, p_z(z), T, \alpha$)
  Initialize generator parameters θ_G and discriminator parameters θ_D
  Define generator network $G = \text{BuildGenerator}(\text{architecture})$ ▷ e.g., LSTM, Transformer
  Define discriminator network $D = \text{BuildDiscriminator}(\text{architecture})$
  for $t = 1$ to T **do**
    Sample a mini-batch $\{\mathbf{x}^{(i)}\}_{i=1}^m$ from real text data $\mathbf{X}$
    Sample noise vectors $\{z^{(i)}\}_{i=1}^m$ from prior distribution $p_z(z)$
    Generate fake samples: $\text{GenerateSequence}(z^{(i)}, G)$ and assign to $\tilde{\mathbf{x}}^{(i)}$
    // **Discriminator Update**
    Compute discriminator loss:

$$\mathcal{L}_D = -\frac{1}{m}\sum_{i=1}^{m}\left[\log \text{ClassifyText}(\mathbf{x}^{(i)}, D) + \log\left(1 - \text{ClassifyText}(\tilde{\mathbf{x}}^{(i)}, D)\right)\right]$$

    Update discriminator parameters: $\theta_D \leftarrow \theta_D - \alpha\nabla_{\theta_D}\mathcal{L}_D$
    // **Generator Update**
    Sample new noise $\{z^{(i)}\}_{i=1}^m$ from $p_z(z)$
    Generate new fake samples:

$$\tilde{\mathbf{x}}^{(i)} = \text{GenerateSequence}(z^{(i)}, G)$$

    Compute generator loss:

$$\mathcal{L}_G = -\frac{1}{m}\sum_{i=1}^{m}\log \text{ClassifyText}(\tilde{\mathbf{x}}^{(i)}, D)$$

    Update generator parameters: $\theta_G \leftarrow \theta_G - \alpha\nabla_{\theta_G}\mathcal{L}_G$
  end for
  return G, D
end function

Once the GAN is trained, Algorithm 6 generates new synthetic text sequences from the trained generator G. This algorithm converts latent noise vectors into human-readable text by decoding logits or embeddings into tokens, typically employing softmax-based decoding strategies such as greedy decoding or sampling methods to enhance diversity.

Vocabulary $\mathcal{V}$ Definition: The vocabulary $\mathcal{V}$ is defined as a finite set of tokens:

$$\mathcal{V} = \{w_1, w_2, \ldots, w_{|\mathcal{V}|}\}$$

Algorithm 2 BuildGenerator($\cdot$) function call by TrainTextGAN($\cdot$) function

function BuildGenerator(architecture)
    if architecture = `LSTM` **then**
        Build multi-layer LSTM with residual connections and batch normalization
    else if architecture = `Transformer` **then**
        Build Transformer decoder stack with self-attention and feed-forward layers
    end if
    return Generator model G
end function

Algorithm 3 BuildDiscriminator($\cdot$) function call by TrainTextGAN($\cdot$) function

function BuildDiscriminator(architecture)
    if architecture = `LSTM` **then**
        Build bidirectional LSTM with classification head
    else if architecture = `Transformer` **then**
        Build Transformer encoder stack followed by classification layer
    end if
    return Discriminator model D
end function

Algorithm 4 GenerateSequence($\cdot$) function call by TrainTextGAN($\cdot$) function

function GenerateSequence(z, G)
    Map latent noise z to token sequence using generator G
    Apply decoding (e.g., greedy, sampling, beam search)
    return $\tilde{\mathbf{x}}$
end function

Algorithm 5 ClassifyText($\cdot$) function call by TrainTextGAN($\cdot$) function

function ClassifyText($\mathbf{x}, D$)
    Compute real/fake probability score using discriminator D
    return $D(\mathbf{x})$
end function

Algorithm 6 Generate Modeled Text Data Using Trained GAN

Require: Trained Generator G, Vocabulary $\mathcal{V}$, Noise distribution $p_z(z)$, Number of samples n
Ensure: Generated text samples $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_n]^T$

function GENERATETEXT($G, \mathcal{V}, p_z(z), n$)
    Initialize empty list $\hat{\mathbf{X}} = [\,]$
    for $i = 1$ to n **do**
        Sample noise vector $z^{(i)} \sim p_z(z)$
        Generate logits or embeddings: $\tilde{\mathbf{x}}^{(i)} = G(z^{(i)})$
        Decode to text: $\hat{\mathbf{x}}^{(i)} = \text{DECODE}(\tilde{\mathbf{x}}^{(i)}, \mathcal{V})$
        Append $\hat{\mathbf{x}}^{(i)}$ to $\hat{\mathbf{X}}$
    end for
    return $\hat{\mathbf{X}}$
end function

function DECODE($\tilde{\mathbf{x}}, \mathcal{V}$)
    Initialize empty string $s \leftarrow$ ""
    for each time step t in $\tilde{\mathbf{x}}$ **do**
        Compute token probabilities: $p_t = \text{Softmax}(\tilde{\mathbf{x}}_t)$
        Select token ID: $j = \arg\max_j p_t[j]$ ▷ Can replace with sampling for diversity
        Lookup token: $w = \mathcal{V}[j]$
        Append token: $s \leftarrow s \,\|\, w$ ▷ Concatenate to string
    end for
    return s
end function

where each w_i is a unique token used in modeling text. Tokens can be words, subwords, or characters depending on the tokenization strategy. The generator outputs a distribution over this vocabulary at each time step, and the DECODE function maps indices to strings using $\mathcal{V}[j]$.

4.3 Implementation Details

The implementation of GAN-based text modeling involves the following critical steps:

Data Preprocessing: Text data must be tokenized and converted to numerical representations (e.g., using word embeddings or subword encoding methods like WordPiece or BPE). Special tokens (`[PAD]`, `[EOS]`, `[UNK]`) should be included in the vocabulary $\mathcal{V}$.

Model Processing: The generator and discriminator architectures are carefully designed to balance complexity and performance, integrating techniques such as residual connections and batch normalization to ensure stable training.

Evaluation: Evaluating GAN-generated text typically involves metrics such as BLEU, ROUGE, or perplexity, supplemented by qualitative analysis or human evaluations due to the subjective nature of text quality.

4.4 Advantages and Limitations

Advantages: GAN-based models for text generation are capable of producing highly realistic and contextually relevant text sequences, which makes them particularly valuable in various natural language processing tasks. These models offer considerable flexibility, as they can effectively leverage advanced neural architectures, such as Long Short-Term Memory (LSTM) networks and

Transformer models, enabling them to capture intricate language structures. Additionally, GANs are beneficial for practical applications, including data augmentation and synthesis tasks, where generating diverse and coherent text data is essential.

Limitations: GAN-based text generation models pose significant challenges during training, primarily due to issues such as mode collapse and training instability, which can result in repetitive or limited outputs. Additionally, without meticulous parameter tuning and regularization, the generated sequences may lack sufficient diversity, potentially leading to monotonous or predictable text. Finally, objectively evaluating the quality and effectiveness of generated text remains a complex and subjective task, as current automated metrics do not always align well with human assessments of naturalness or contextual appropriateness.

4.5 Analysis of the GANs Algorithm

Time Complexity Analysis:

The GAN algorithm for modeling text data using deep sequence models such as LSTMs or Transformers involves multiple iterative training steps. Each training iteration comprises forward and backward passes through both the generator and discriminator networks. For sequence models, assuming input sequences of length T, vocabulary size V, embedding dimension d, and hidden dimension h:

- **Generator Complexity:** An LSTM or Transformer-based generator typically exhibits a complexity of $O(T(h^2 + dh))$ per sequence, with Transformer models exhibiting complexity of approximately $O(T^2 d)$ due to self-attention mechanisms [13].
- **Discriminator Complexity:** Similarly, the discriminator, structured with bidirectional LSTM or Transformer encoders, has complexity on the order of $O(T(h^2 + dh))$ or $O(T^2 d)$.

Hence, the overall training complexity per iteration, for batch size m, is $O(mT^2 d)$ for Transformer-based models and $O(mT(h^2 + dh))$ for LSTM-based architectures.

Analysis of the GAN Algorithms:

GAN algorithms effectively train generators to produce realistic synthetic data by continuously challenging them against discriminators that improve their capacity to distinguish real from generated data. For text modeling, this process leverages sequence architectures capable of capturing long-range dependencies and context-sensitive relationships among tokens, resulting in high-quality text synthesis. However, training stability is a known issue, requiring careful tuning of hyperparameters and architectures to mitigate mode collapse and vanishing gradients [5, 7, 6, 2].

Summary - GANs Algorithm:

The GAN algorithm effectively utilizes deep sequence models, such as LSTMs and Transformers, to generate realistic text sequences. Despite computational complexity and potential training challenges, GANs remain powerful tools for synthesizing text, beneficial in data augmentation and creative content generation tasks.

4.6 Correctness Proof of the GANs Algorithm

Correctness Proof:

The correctness of GAN algorithms relies on the theoretical foundation established by the minimax optimization problem described by Goodfellow et al. [2]. Specifically, under optimal conditions, the discriminator reaches an equilibrium where it can no longer distinguish between real and synthetic data, thereby making its predictive accuracy $D(x) = 0.5$ for all inputs x. At this equilibrium, the generator distribution $p_g(x)$ perfectly matches the real data distribution $p_{\text{data}}(x)$, i.e., $p_g(x) = p_{\text{data}}(x)$.

Formally, the optimal discriminator $D^(x)$ for a fixed generator G is given by:

$$D^(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \tag{4}$$

Substituting this optimal discriminator into the GAN objective leads to minimizing the Jensen-Shannon divergence between the generator's distribution and the true data distribution. Minimizing this divergence ensures convergence of the generator distribution to the true distribution [2].

Summary for Correctness Proof:

The correctness of GAN algorithms, including those using deep sequential architectures, is theoretically grounded in their minimization of Jensen-Shannon divergence, ensuring the generator distribution aligns closely with the real data distribution at equilibrium. This fundamental principle guarantees the algorithm's validity in generating realistic and accurate synthetic text data.

4.7 Summary - Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are powerful generative models comprising two neural networks—a generator and a discriminator—trained adversarially. The generator creates synthetic data aiming to mimic real data distributions, while the discriminator evaluates the authenticity of data samples. Employing advanced sequential models, including recurrent neural networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer architectures, GANs effectively generate coherent and contextually relevant text sequences. Despite challenges such as training instability, mode collapse, and complex evaluations, GANs provide robust capabilities in realistic text synthesis, data augmentation, and creative content generation. Their theoretical underpinning in minimizing Jensen-Shannon divergence ensures convergence toward accurately replicating the underlying data distribution, making GANs indispensable tools in contemporary text modeling tasks.

5 Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) are generative models designed for learning probabilistic representations of data by encoding inputs into a latent space and subsequently decoding them back into the original domain. Proposed by Kingma and Welling [4], VAEs are particularly useful for text generation tasks due to their capability to model sequential data effectively, making them suitable for regression and generative analyses in textual contexts.

5.1 Mathematical Formulations

The foundational principle of VAEs is to maximize the Evidence Lower Bound (ELBO), defined as:

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - KL(q(z|x)||p(z)) \tag{5}$$

where $p(z)$ is a prior distribution over latent variables (typically Gaussian), $q(z|x)$ is the approximate posterior distribution modeled by the encoder, and $p(x|z)$ is the likelihood distribution modeled by the decoder. The KL-divergence term ensures that the latent space maintains regularization properties, promoting meaningful latent representations.

5.2 Algorithm Explanation

The VAE algorithm for text data leverages deep sequential models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Transformers. The encoder network processes input sequences to generate latent representations, while the decoder reconstructs the sequences from these latent variables. Algorithm 7 details the training process, incorporating encoding, sampling via the reparameterization trick, decoding, and parameter updates guided by the ELBO loss function.

Algorithm 8 describes the process of generating new text data from a trained VAE decoder. It involves sampling from the latent space, decoding latent variables to embeddings or logits, and transforming these embeddings into readable text through a decoding function that uses the softmax operation and vocabulary mapping.

5.3 Implementation Details

Implementing VAEs for text modeling begins with data preprocessing. This phase includes tokenizing the input text, converting tokens into numerical representations through techniques like embeddings or one-hot encoding, and padding sequences to ensure uniform length across batches. Additionally, special tokens such as end-of-sequence (EOS) and padding (PAD) tokens are introduced to handle variable-length sequences effectively.

The next step, model processing, involves selecting suitable architectures for the encoder and decoder networks. Common choices include recurrent structures like Long Short-Term Memory (LSTM) networks or Transformer-based architectures, each offering unique advantages in capturing sequential dependencies and contextual nuances. Optimization during training utilizes gradient descent methods guided by the Evidence Lower Bound (ELBO), balancing accurate reconstruction with meaningful latent-space regularization.

Algorithm 7 Training Variational Autoencoder (VAE) for Text Data

Require: Text corpus $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$, latent dimension d_z, epochs E, learning rate α
Ensure: Trained Encoder parameters ϕ, Decoder parameters θ

function TRAINTEXTVAE($\mathbf{X}, d_z, E, \alpha$)
  Initialize encoder parameters ϕ and decoder parameters θ
  Define Encoder network: $q_\phi(z|x)$
  Define Decoder network: $p_\theta(x|z)$
  for $e = 1$ to E **do**
    for mini-batch $\{\mathbf{x}^{(i)}\}_{i=1}^m$ from $\mathbf{X}$ **do**
      Compute encoder outputs $(\mu_z^{(i)}, \sigma_z^{(i)}) \leftarrow q_\phi(z|\mathbf{x}^{(i)})$
      Sample latent variable $z^{(i)} \leftarrow \mu_z^{(i)} + \sigma_z^{(i)} \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$
      Compute decoder outputs $\hat{\mathbf{x}}^{(i)} \leftarrow p_\theta(x|z^{(i)})$
      Compute loss:

$$\mathcal{L}_{\text{VAE}} = \frac{1}{m} \sum_{i=1}^{m} \Big[-\log p_\theta(\mathbf{x}^{(i)}|z^{(i)}) + KL(q_\phi(z|\mathbf{x}^{(i)})||p(z)) \Big]$$

      Update parameters:

$$\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_{\text{VAE}}, \quad \theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{VAE}}$$

    end for
  end for
  return ϕ, θ

Algorithm 8 Generate Text Data Using Trained VAE

Require: Trained Decoder parameters θ, vocabulary $\mathcal{V}$, latent dimension d_z, number of samples n
Ensure: Generated text samples $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_n]^T$

function GENERATEVAETEXT($\theta, \mathcal{V}, d_z, n$)
  Initialize empty list $\hat{\mathbf{X}} = []$
  for $i = 1$ to n **do**
    Sample latent vector $z^{(i)} \sim \mathcal{N}(0, I_{d_z})$
    Decode latent vector to logits or embeddings: $\tilde{\mathbf{x}}^{(i)} = p_\theta(x|z^{(i)})$
    Decode to text: $\hat{\mathbf{x}}^{(i)} = \text{DECODETOTEXT}(\tilde{\mathbf{x}}^{(i)}, \mathcal{V})$
    Append generated text to list $\hat{\mathbf{X}}$
  end for
  return generated text samples $\hat{\mathbf{X}}$
end function

function DECODETOTEXT($\tilde{\mathbf{x}}, \mathcal{V}$)
  Initialize empty string $s \leftarrow$ ""
  for each time step t in $\tilde{\mathbf{x}}$ **do**
    Compute token probabilities: $p_t = \text{Softmax}(\tilde{\mathbf{x}}_t)$
    Select token ID: $j = \arg\max_j p_t[j]$ ▷ or sampling for diversity
    Lookup token: $w = \mathcal{V}[j]$
    Append token to string: $s \leftarrow s||w$
  end for
  return decoded string s
end function
  $=0$

Finally, the evaluation phase assesses the quality of the generated text. Reconstruction quality is commonly measured using quantitative metrics such as perplexity and BLEU scores, providing objective insights into model performance. Complementing these quantitative metrics, qualitative analysis—often through human judgment—is essential to fully understand the coherence, diversity, and contextual relevance of the generated textual outputs.

5.4 Advantages and Limitations

Advantages: VAEs provide a robust probabilistic modeling framework, enabling generation of diverse and coherent text. They facilitate interpretable latent representations, beneficial for downstream tasks such as data augmentation and unsupervised representation learning.

Limitations: VAEs may struggle with generating highly detailed and contextually nuanced text. Balancing reconstruction quality and regularization terms is challenging and may require extensive hyperparameter tuning. The generated text quality can sometimes be overly generic or lack specificity compared to GAN-based methods.

5.5 Analysis of the VAEs Algorithm

Analysis of the VAEs Algorithm for Text, VAEs for Modeling Text Data:

Variational Autoencoders (VAEs) model text data by employing deep sequential architectures, typically Long Short-Term Memory (LSTM) or Transformer-based encoders and decoders. The algorithm encodes input text sequences into a probabilistic latent representation and decodes these representations to reconstruct the original input. This architecture effectively captures both the semantic and syntactic characteristics of the data, enabling high-quality text generation and meaningful latent-space explorations [3, 12].

Time Complexity Analysis:

The computational complexity of training VAEs for text data primarily depends on the choice of encoder and decoder architectures. For LSTM-based models, complexity typically scales as $O(mT(h^2 + dh))$ per batch, where m is the batch size, T is the sequence length, h is the hidden state size, and d is the embedding dimension. For Transformer-based models, complexity typically scales as $O(mT^2d)$ due to self-attention mechanisms [12]. Consequently, training complexity can vary significantly based on the chosen architecture and hyperparameters.

Analysis of the VAEs Algorithm:

VAEs employ variational inference, optimizing the Evidence Lower BOund (ELBO) objective. This approach inherently provides a structured probabilistic framework, balancing reconstruction accuracy with latent regularization. The training involves iterative parameter updates via gradient descent, enabling stable convergence to an approximate posterior distribution that captures complex data relationships.

Summary - VAEs Algorithm:

The VAEs algorithm provides a robust probabilistic approach to text generation and representation learning. Its inherent balance of reconstruction fidelity and latent representation regularization allows for the generation of diverse and contextually relevant text, facilitating numerous applications in natural language processing tasks.

5.6 Correctness Proof of the VAEs Algorithm

Correctness Proof:

The correctness of the VAE algorithm for text data is anchored in variational inference theory. Specifically, the ELBO serves as a rigorous lower bound for the log-likelihood of the observed data:

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - KL(q(z|x)|p(z)). \tag{6}$$

Maximizing this bound ensures that the approximate posterior $q(z|x)$ converges toward the true posterior, thereby guaranteeing that the reconstructed output $p(x|z)$ accurately represents the true data distribution [3, 8].

The iterative gradient descent optimization method ensures monotonic improvement or convergence of the ELBO. Hence, as training progresses, the algorithm improves its generative and reconstructive capabilities, verifying the correctness and theoretical soundness of the VAE training process.

Summary for Correctness Proof:

The correctness and validity of the VAEs algorithm hinge on maximizing the ELBO, a rigorous lower bound of data log-likelihood. Through iterative optimization, the algorithm ensures convergence to a probabilistic representation that effectively captures the underlying data distribution, thereby validating its theoretical foundations and practical effectiveness.

5.7 Summary - Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) provide a robust probabilistic framework for modeling textual data by learning latent representations through encoder-decoder architectures. They effectively balance reconstruction accuracy and latent regularization via optimization of the Evidence Lower Bound (ELBO). Leveraging deep sequential models such as LSTMs and Transformers, VAEs capture semantic and syntactic intricacies, facilitating diverse and coherent text generation. Despite challenges related to hyperparameter tuning and generating highly nuanced content, VAEs excel in tasks requiring interpretable latent representations, data augmentation, and unsupervised learning. The theoretical foundation based on variational inference ensures stable convergence and accurate approximation of complex data distributions, underscoring VAEs' versatility and effectiveness in text modeling applications.

6 Sequence-to-Sequence (Seq2Seq)

Sequence-to-Sequence (Seq2Seq) models, initially proposed by Sutskever et al. [11], employ encoder-decoder architectures to translate sequences from one domain or language to another. In the context of regression analysis and text modeling, these models leverage deep recurrent networks such as LSTMs or attention-based architectures like Transformers to effectively capture context and long-term dependencies within sequential data.

6.1 Mathematical Formulations

The fundamental structure of Seq2Seq models includes two primary components: an encoder and a decoder. Given input sequences $\mathbf{x}$ and output sequences $\mathbf{y}$, the model estimates the conditional probability:

$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{<t}, \mathbf{x}), \tag{7}$$

where y_t is the t-th token in the sequence, and $y_{<t}$ represents all tokens preceding y_t. The model is typically trained using Cross-Entropy loss, defined as:

$$\mathcal{L}\text{Seq2Seq} = -\sum t = 1^T \log p(y_t|y_{<t}, \mathbf{x}). \tag{8}$$

6.2 Algorithm Explanation

Algorithm 9 outlines the training process for a Seq2Seq model specific to text data. This algorithm involves encoding the input text sequences into intermediate hidden representations, which are subsequently decoded into output sequences. The parameters of both encoder and decoder networks are optimized iteratively through gradient descent methods to minimize the Cross-Entropy loss.

Algorithm 10 describes the process of generating new text data using a trained Seq2Seq model. It begins by encoding input sequences to obtain context-rich hidden states, from which the decoder generates corresponding output sequences. The generated outputs are transformed into human-readable text through a decoding step involving token probability computations and vocabulary lookups.

6.3 Implementation Details

Implementing Seq2Seq models for text begins with data preprocessing, a crucial step that ensures the data is appropriately structured for sequential modeling. This stage involves tokenization, which breaks down the raw text into smaller, meaningful units called tokens. Subsequently, tokens are transformed into numerical representations through embedding layers, enabling the model to capture semantic relationships effectively. Additionally, sequences are padded to maintain uniform lengths within batches, and special tokens, such as End-of-Sequence (EOS) and Padding (PAD), are introduced to handle sequence boundaries and ensure consistent input formatting.

The next critical stage, model processing, involves selecting suitable architectures and configuring their parameters to achieve optimal performance. Commonly employed architectures include Long

Algorithm 9 Training Sequence-to-Sequence (Seq2Seq) Model for Text Data

Require: Text corpus $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$, number of epochs E, learning rate α
Ensure: Trained Encoder parameters ϕ, Decoder parameters θ

function TrainSeq2Seq($\mathbf{X}, E, \alpha$)
  Initialize encoder parameters ϕ and decoder parameters θ
  Define Encoder network: $q_\phi(z|x)$
  Define Decoder network: $p_\theta(y|z)$
  for $e = 1$ to E **do**
    for mini-batch $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^m$ from $\mathbf{X}$ **do**
      Compute encoder hidden states $h^{(i)} \leftarrow q_\phi(\mathbf{x}^{(i)})$
      Compute decoder outputs $\hat{\mathbf{y}}^{(i)} \leftarrow p_\theta(\mathbf{y}^{(i)}|h^{(i)})$
      Compute loss:

$$\mathcal{L}_{\text{Seq2Seq}} = \frac{1}{m} \sum_{i=1}^{m} \text{CrossEntropy}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$$

      Update parameters:

$$\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_{\text{Seq2Seq}}, \quad \theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{Seq2Seq}}$$

    end for
  end for
  return ϕ, θ
end function

Algorithm 10 Generate Text Data Using Trained Seq2Seq Model

Require: Trained Encoder parameters ϕ, Decoder parameters θ, vocabulary $\mathcal{V}$, input sequences $\mathbf{X}_{\text{input}} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$
Ensure: Generated text samples $\hat{\mathbf{Y}} = [\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_n]^T$

function GenerateSeq2SeqText($\phi, \theta, \mathcal{V}, \mathbf{X}$input)
  Initialize empty list $\hat{\mathbf{Y}} = []$
  for $i = 1$ to n **do**
    Encode input sequence: $h^{(i)} = q\phi(\mathbf{x}^{(i)})$
    Generate decoder outputs (logits or embeddings): $\tilde{\mathbf{y}}^{(i)} = p_\theta(y|h^{(i)})$
    Decode to text: $\hat{\mathbf{y}}^{(i)} = \text{DecodeToText}(\tilde{\mathbf{y}}^{(i)}, \mathcal{V})$
    Append generated text to list $\hat{\mathbf{Y}}$
  end for
  return generated text samples $\hat{\mathbf{Y}}$
end function

function DecodeToText($\tilde{\mathbf{y}}, \mathcal{V}$)
  Initialize empty string $s \leftarrow$ ""
  for each time step t in $\tilde{\mathbf{y}}$ **do**
    Compute token probabilities: $p_t = \text{Softmax}(\tilde{\mathbf{y}}_t)$
    Select token ID: $j = \arg\max_j p_t[j]$ ▷ or sampling for diversity
    Lookup token: $w = \mathcal{V}[j]$
    Append token to string: $s \leftarrow s||w$
  end for
  return decoded string s
end function

Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRU), and Transformer-based models, each offering distinct advantages depending on the complexity and dependencies within the data. Key hyperparameters, such as learning rate, hidden dimensions, and the number of layers, must be carefully tuned. The training process generally involves optimizing the model parameters through gradient-based methods, specifically using the Cross-Entropy loss function, which evaluates the accuracy of the generated sequences relative to the target outputs.

The final stage encompasses the evaluation of the trained Seq2Seq models to determine their effectiveness and quality. This evaluation combines quantitative and qualitative approaches. Quantitative assessments utilize standard metrics such as BLEU, ROUGE, and perplexity scores to objectively measure aspects like fluency, accuracy, and overlap with reference texts. These measures are supplemented with qualitative analyses, often involving human judgment, to evaluate the coherence, context relevance, and overall meaningfulness of the generated sequences, thus providing a comprehensive understanding of the model's capabilities and limitations.

6.4 Advantages and Limitations

Advantages: Seq2Seq models are highly effective for tasks involving sequential transformations, such as machine translation, summarization, and dialogue generation. Their flexible architecture allows incorporation of attention mechanisms, significantly improving their capability to handle long sequences and complex dependencies.

Limitations: These models can suffer from training instability, vanishing gradients, and difficulties in capturing extremely long-term dependencies. Furthermore, they often require extensive computational resources and careful hyperparameter tuning to achieve optimal performance.

6.5 Analysis of the Seq2Seq Algorithm

Analysis of the Seq2Seq Algorithm for Text, Seq2Seq for Modeling Text Data:

Sequence-to-Sequence (Seq2Seq) models, as proposed by Sutskever et al. [10], use encoder-decoder structures to convert input sequences into output sequences, effectively modeling complex sequential dependencies in text data. These models are particularly useful for machine translation, summarization, and dialogue generation tasks. The encoder captures semantic and syntactic context, transforming input sequences into latent representations, which the decoder then uses to generate coherent and contextually appropriate outputs.

Time Complexity Analysis:

The computational complexity of Seq2Seq models depends significantly on the chosen encoder-decoder architecture. For Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)-based models, the complexity typically scales as $O(mT(h^2 + dh))$ per batch, with m representing batch size, T sequence length, h hidden dimension size, and d embedding dimension size. Transformer-based architectures have a higher complexity of $O(mT^2d)$ due to their self-attention mechanisms, although they typically achieve better parallelization efficiency and faster training convergence [12].

Analysis of the Seq2Seq Algorithms:

The Seq2Seq model optimization leverages Cross-Entropy loss, optimizing the conditional probabilities of generating each token given preceding tokens. This approach effectively encourages the model to maximize the likelihood of the correct output sequences during training, promoting coherent and contextually meaningful outputs. Attention mechanisms further enhance model capability by dynamically weighting different input segments, greatly improving accuracy and fluency in generated texts [1].

Summary - Seq2Seq Algorithm:

The Seq2Seq algorithm is highly effective for sequential text generation tasks due to its robust ability to model dependencies within data. Through the strategic use of encoder-decoder architectures and attention mechanisms, Seq2Seq effectively captures linguistic nuances, producing coherent, contextually rich textual outputs suitable for diverse NLP applications.

6.6 Correctness Proof of the Seq2Seq Algorithm

Correctness Proof:

The correctness of Seq2Seq models is theoretically grounded in maximizing the conditional probability of the target sequence given the input sequence. Formally, given input sequence $\mathbf{x}$ and output sequence $\mathbf{y}$, the Seq2Seq model maximizes:

$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{<t}, \mathbf{x}). \tag{9}$$

Optimization of this probability is performed by minimizing the Cross-Entropy loss:

$$\mathcal{L}\text{Seq2Seq} = -\sum t = 1^T \log p(y_t|y_{<t}, \mathbf{x}). \tag{10}$$

Through iterative gradient descent methods, the model parameters converge toward optimal values, ensuring the predicted probability distribution aligns closely with the true data distribution. Therefore, correctness and convergence are guaranteed through continuous minimization of the Cross-Entropy loss, as proven by standard results in statistical learning theory [10].

Summary for Correctness Proof:

The correctness of Seq2Seq algorithms is assured by optimizing the conditional likelihood of generated sequences, ensuring convergence to a robust approximation of the underlying data distribution. This theoretical foundation validates the algorithm's ability to consistently produce accurate and contextually relevant textual outputs.

6.7 Summary - Sequence-to-Sequence (Seq2Seq)

Sequence-to-Sequence (Seq2Seq) models effectively handle tasks involving the translation and transformation of sequential data, such as machine translation, summarization, and conversational modeling. These models utilize encoder-decoder architectures, frequently enhanced with attention mechanisms, to accurately capture long-term dependencies and contextual nuances within sequences. While Seq2Seq models demonstrate robust performance in generating coherent and contextually relevant textual outputs, they require careful management of computational resources

and hyperparameter optimization to overcome challenges like training instability and complexity in handling extensive dependencies. The theoretical foundation of these models, grounded in maximizing conditional probabilities and minimizing Cross-Entropy loss, ensures their efficacy and correctness across diverse text-based applications.

7 Recent Advances in Intelligent Algorithms

Recent developments in intelligent algorithms have significantly expanded beyond Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Sequence-to-Sequence (Seq2Seq) models. These advancements include models and frameworks that have dramatically improved capabilities across multiple data science tasks.

7.1 Transformers and Attention-Based Models

The advent of Transformer architectures, first proposed by Vaswani et al. [12], has revolutionized intelligent algorithms, especially in natural language processing (NLP). Transformers effectively model long-range dependencies without recurrence or convolution, leveraging self-attention mechanisms. Recent variants, such as BERT, GPT, and their successors (GPT-3, GPT-4), have set new benchmarks for understanding and generating text with impressive contextual sensitivity and coherence.

7.2 Diffusion Models

Diffusion probabilistic models represent a recent breakthrough in generative modeling, particularly for continuous data like images and audio. Proposed by Sohl-Dickstein et al. [9], diffusion models iteratively add and remove noise from data to generate realistic samples. Their capability for high-fidelity generation has been demonstrated through models such as DALL·E 2 and Stable Diffusion, significantly advancing the quality and control of generated content.

7.3 Reinforcement Learning with Human Feedback

Reinforcement learning with human feedback (RLHF) integrates human evaluative judgments directly into reinforcement learning algorithms, enhancing decision-making quality and alignment with human values. Popularized by its successful application in training models like ChatGPT, RLHF allows algorithms to optimize performance not only based on predefined metrics but also human subjective evaluations, leading to safer and more useful AI systems.

7.4 Neural Architecture Search (NAS)

Neural Architecture Search has emerged as an automated approach to discovering efficient neural network architectures, thus reducing the extensive manual effort in network design. NAS methods, including evolutionary algorithms and reinforcement learning techniques, systematically explore architectural hyperparameters to optimize performance, efficiency, and scalability, as illustrated by successful implementations such as NASNet and EfficientNet.

7.5 Federated Learning

Federated Learning represents a significant advance in decentralized machine learning, enabling multiple clients to collaboratively train models without sharing their data explicitly. This approach addresses critical privacy concerns, facilitating robust model development across distributed datasets. Federated learning algorithms have seen widespread adoption, particularly in domains like healthcare and finance, where data privacy is paramount.

In summary, these recent advances illustrate the dynamic nature and continuous evolution of intelligent algorithms, emphasizing the ongoing integration of sophisticated techniques that significantly enhance model performance, efficiency, and applicability across diverse data science challenges.

8 Intelligent Algorithms using Python Packages

Python provides robust libraries and frameworks that significantly streamline the implementation of Intelligent Algorithms such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Sequence-to-Sequence (Seq2Seq) models. These packages facilitate rapid prototyping, training, evaluation, and deployment of advanced models.

8.1 Generative Adversarial Networks (GANs)

For implementing GANs, popular Python packages include:

- **PyTorch**: Provides intuitive APIs and dynamic computation graphs for custom GAN architectures and training loops.
- **TensorFlow**: Offers comprehensive GAN frameworks through TensorFlow-GAN (TF-GAN) for easy implementation of standard and customized GAN models.
- **Keras**: A user-friendly high-level API built on top of TensorFlow, suitable for quickly setting up and training GAN models.

Example (PyTorch GAN):

```
import torch
import torch.nn as nn

class Generator(nn.Module):
def init(self):
super(Generator, self).init()
self.model = nn.Sequential(
nn.Linear(100, 256),
nn.ReLU(),
nn.Linear(256, 784),
nn.Tanh()
)

def forward(self, z):
    return self.model(z)
```

8.2 Variational Autoencoders (VAEs)

Key Python packages for implementing VAEs include:

- **PyTorch**: Enables flexible and efficient implementation of complex VAE architectures, leveraging dynamic computational graphs.
- **TensorFlow Probability (TFP)**: Provides built-in probabilistic modeling capabilities essential for creating and optimizing VAE models.
- **Keras**: Facilitates rapid VAE prototyping and deployment with easy-to-use API layers and comprehensive examples.

Example (PyTorch VAE):

```
import torch
import torch.nn as nn

class VAE(nn.Module):
def init(self):
super(VAE, self).init()
self.encoder = nn.Sequential(nn.Linear(784, 400), nn.ReLU())
self.mu = nn.Linear(400, 20)
self.log_var = nn.Linear(400, 20)
self.decoder = nn.Sequential(nn.Linear(20, 400), nn.ReLU(), nn.Linear(400, 784), nn.Sigmoi

def reparameterize(self, mu, log_var):
    std = torch.exp(0.5 * log_var)
    eps = torch.randn_like(std)
    return mu + eps * std

def forward(self, x):
    h = self.encoder(x)
    mu, log_var = self.mu(h), self.log_var(h)
    z = self.reparameterize(mu, log_var)
    return self.decoder(z), mu, log_var
```

8.3 Sequence-to-Sequence (Seq2Seq) Models

Python packages widely used for Seq2Seq model implementations are:

- **PyTorch**: Supports detailed customization of Seq2Seq architectures, including attention mechanisms, with robust control over training routines.
- **TensorFlow and Keras**: Offer integrated high-level APIs specifically tailored for building and training Seq2Seq models efficiently.
- **OpenNMT**: A specialized open-source toolkit explicitly designed for sequence-to-sequence modeling tasks, offering extensive capabilities for machine translation and text generation.

Example (PyTorch Seq2Seq):

```
import torch
import torch.nn as nn

class Encoder(nn.Module):
def init(self, input_dim, hidden_dim):
super(Encoder, self).init()
self.rnn = nn.GRU(input_dim, hidden_dim)

def forward(self, src):
```

```
    outputs, hidden = self.rnn(src)
    return hidden

class Decoder(nn.Module):
def init(self, hidden_dim, output_dim):
super(Decoder, self).init()
self.rnn = nn.GRU(hidden_dim, output_dim)

def forward(self, hidden):
    outputs, hidden = self.rnn(hidden)
    return outputs
```

9 Summary

This document provided an overview and detailed analysis of three key Intelligent Algorithms—Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Sequence-to-Sequence (Seq2Seq) models—highlighting their mathematical underpinnings, practical applications, and implementation strategies. By exploring these algorithms' theoretical frameworks and illustrating their usage through Python examples, we emphasize their critical role in contemporary data science workflows. Mastery of these Intelligent Algorithms equips data scientists and researchers with powerful tools for data modeling, synthesis, and transformative analysis tasks.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[2] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[3] Diederik P Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[4] Diederik P. Kingma and Max Welling. "Auto-encoding variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2013.

[5] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG]. URL: https://arxiv.org/abs/1511.06434.

[6] Alec Radford et al. *Improving language understanding by generative pre-training*. 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[7] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog* 1 (2019), p. 9.

[8] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *arXiv preprint arXiv:1401.4082* (2014).

[9] Jascha Sohl-Dickstein et al. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2015, pp. 2256–2265.

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in Neural Information Processing Systems* 27 (2014), pp. 3104–3112.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. 2014.

[12] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. 2017.

[13] Ashish Vaswani et al. "Attention Is All You Need". In: vol. 30. 2017.