

Deep Generative Models (AI314)

Master of Science in AI



Dr. Nibras Abdullah

*Assistant professor
Faculty of Informatics
Midocean University*



Autoregressive Models

Introduction

Long Short-Term Memory Network (LSTM)

The LSTM Architecture

Working with Text Data

Tokenization

Creating the Training Set

Recurrent Neural Network (RNN) Extensions

PixelCNN

Autoregressive Models

Long Short-Term Memory Network (LSTM)

An LSTM is a particular type of recurrent neural network (RNN). RNNs contain a recurrent layer (or `_cell_`) that is able to handle sequential data by making its own output at a particular timestep form part of the input to the next timestep.

The LSTM Solution

1

Introduced in 1997

LSTM cells were first introduced in 1997 in a paper by Sepp Hochreiter and Jurgen Schmidhuber. The authors described how LSTMs do not suffer from the same vanishing gradient problem experienced by vanilla RNNs and can be trained on sequences that are hundreds of timesteps long.

2

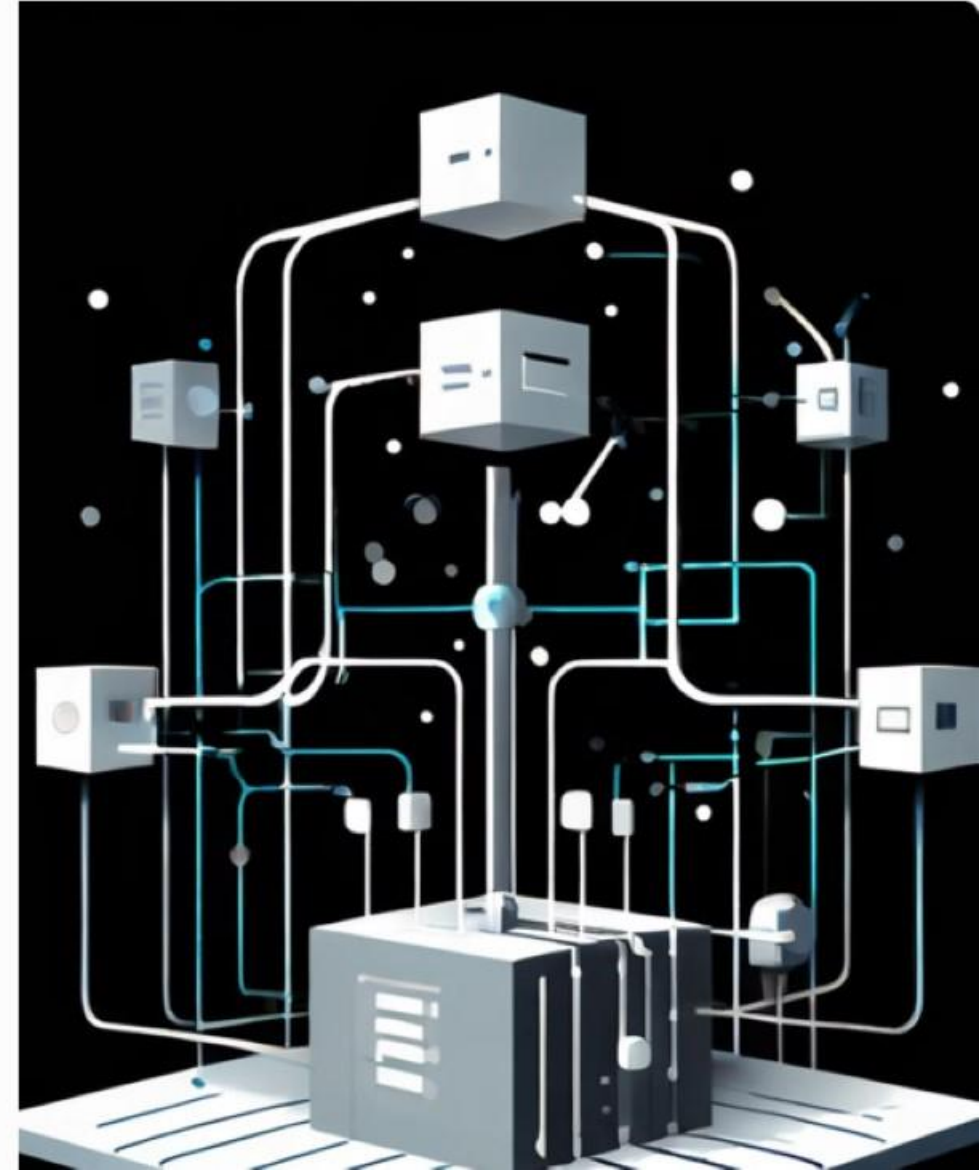
Improved Architecture

Since then, the LSTM architecture has been adapted and improved, and variations such as gated recurrent units (GRUs) are now widely utilized and available as layers in Keras.

3

Wide Range of Applications

LSTMs have been applied to a wide range of problems involving sequential data, including time series forecasting, sentiment analysis, and audio classification. In this chapter, we will be using LSTMs to tackle the challenge of text generation.



Understanding Text Data

Sequential Nature

Text data is fundamentally different from the image data we have seen so far in this class. While images can be processed as a whole, text data is inherently sequential, with each word building upon the previous ones to convey meaning.

Contextual Relationships

The meaning of a word in a text can depend heavily on the context in which it appears. LSTMs are well-suited to capturing these contextual relationships and understanding the nuances of language.

Challenges in Text Generation

Generating coherent and meaningful text is a complex task that requires understanding the structure and flow of language. LSTMs can be trained to generate text by learning patterns and relationships from a given corpus of text data.



LSTM Applications in Text Generation

Time Series Forecasting

LSTMs can be used to make predictions on time series data, such as stock prices or weather patterns, by learning patterns and relationships in the data over time.

Sentiment Analysis

LSTMs can be trained to analyze the sentiment expressed in text, such as customer reviews or social media posts, by understanding the contextual meaning of the words and phrases used.

Audio Classification

LSTMs can be applied to audio data, such as speech or music, by processing the sequential nature of the audio signal and learning to classify different types of audio content.

Text Generation

In this chapter, we will focus on using LSTMs to generate new text, such as creative writing or dialogue, by learning the patterns and structures of language from a given corpus of text data.

Creating the Training Set for an LSTM Language Model

To train an LSTM (Long Short-Term Memory) model to predict the next word in a sequence, we need to create a training set that captures the relationships between words. This process involves shifting the input sequence by one token to generate the target variable, allowing the model to learn the patterns in the data.

The LSTM Architecture

Input and Output

The input to the LSTM model is a sequence of integer tokens, and the output is the probability of each word in the 10,000-word vocabulary appearing next in the sequence.

Embedding Layer

The Embedding layer is used to convert the integer tokens into a dense vector representation, which the LSTM can then process more effectively.

LSTM Layer

The LSTM layer is a specialized type of recurrent neural network that can learn long-term dependencies in sequential data, making it well-suited for language modeling tasks.

The Embedding Layer

1 Word Embeddings

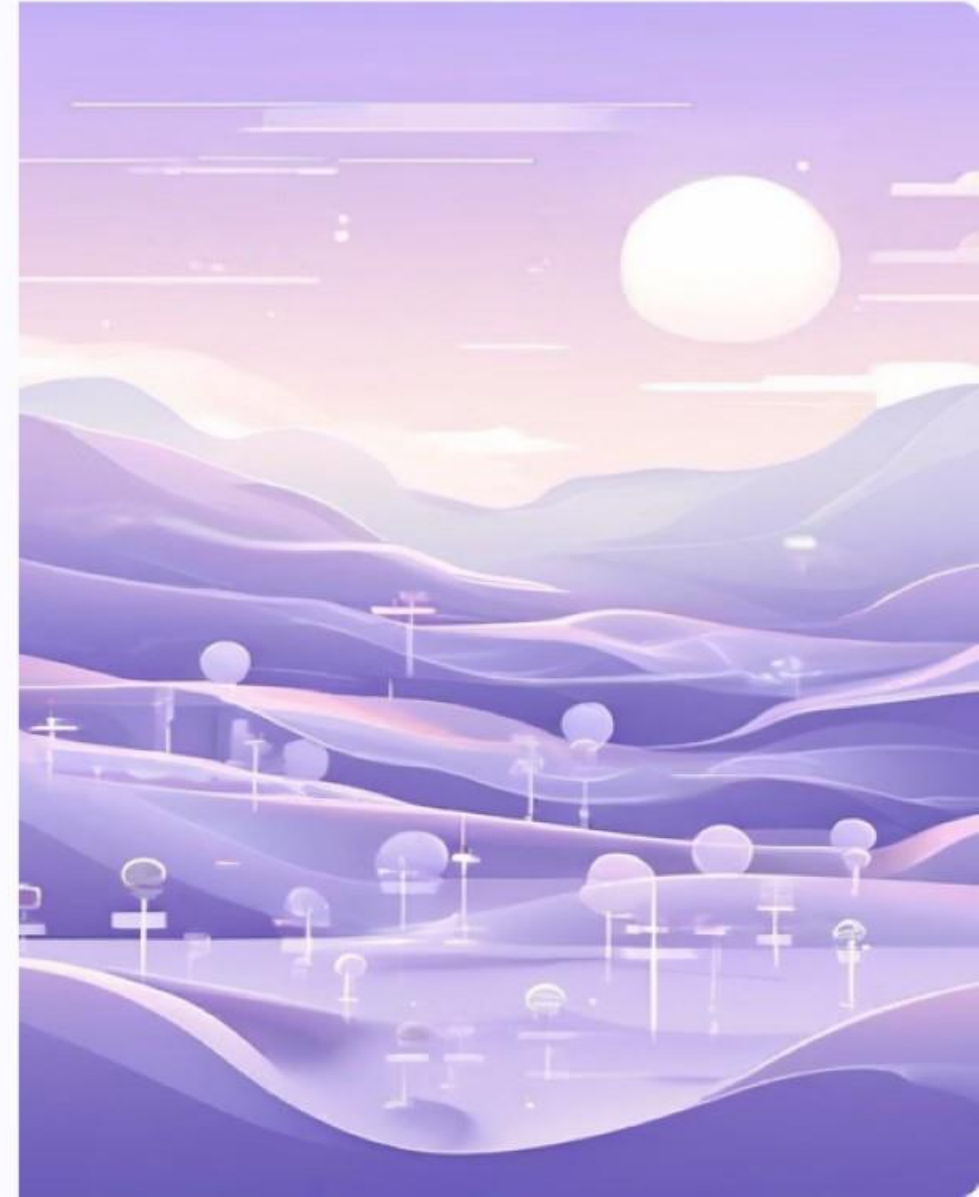
The Embedding layer converts each integer token into a dense vector representation, capturing the semantic and syntactic relationships between words.

2 Learned Representations

The Embedding layer's weights are learned during the training process, allowing the model to develop its own internal representations of the input words.

3 Improved Performance

Using an Embedding layer can significantly improve the performance of the LSTM model, as it allows the model to better understand the context and meaning of the input sequence.



The LSTM Layer

Memory Cell

The LSTM layer maintains a memory cell that can selectively remember and forget information, allowing it to capture long-term dependencies in the input sequence.

Sequential Processing

The LSTM layer processes the input sequence one token at a time, updating its internal state and producing an output at each step, which is then used to predict the next word in the sequence.



Gating Mechanism

The LSTM layer uses a gating mechanism to control the flow of information into and out of the memory cell, enabling it to learn complex patterns in the data.

The LSTM Cell

Forget Gate

The forget gate determines which information from the previous time step should be retained in the memory cell, allowing the LSTM to selectively remember or forget relevant information.

Output Gate

The output gate decides which information from the memory cell and current input should be used to produce the output for the current time step, allowing the LSTM to generate relevant predictions.

Input Gate

The input gate controls which new information from the current input and previous hidden state should be added to the memory cell, enabling the LSTM to learn and update its internal

Cell State

The cell state is the long-term memory of the LSTM, which is updated at each time step based on the input, forget, and output gates, enabling the model to maintain and update its understanding of the sequence.

Training the LSTM

1

Backpropagation

The LSTM model is trained using backpropagation through time, which allows the gradients of the loss function to be computed with respect to the model's parameters, enabling the model to learn from the training data.

2

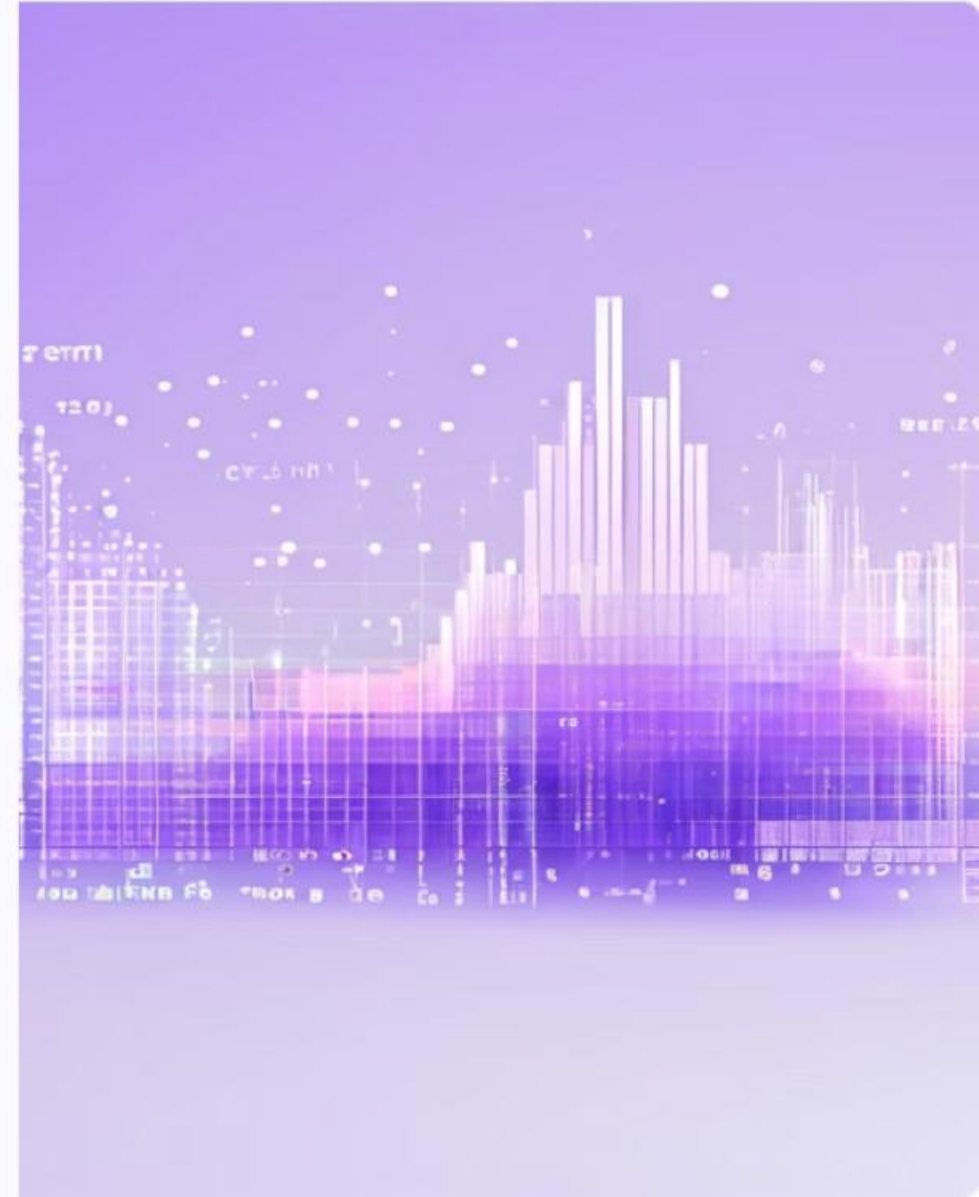
Optimization

The model's parameters are updated using an optimization algorithm, such as Stochastic Gradient Descent (SGD) or Adam, to minimize the loss function and improve the model's performance on the training data.

3

Validation

The model's performance is evaluated on a separate validation set during training, allowing the model to be tuned and preventing overfitting to the training data.



Analysis of the LSTM



Interpretability

Analyzing the LSTM model's internal states and activations can provide insights into how the model is processing the input sequence and making its predictions, enabling a better understanding of the model's behavior.



Performance Evaluation

Evaluating the LSTM model's performance on a held-out test set can help assess its ability to generalize to new, unseen data, and identify areas for further improvement.



Iterative Refinement

The insights gained from analyzing the LSTM model can be used to refine the model architecture, hyperparameters, or training process, leading to improved performance and a better understanding of the underlying language patterns.



Shifting the Sequence for Target Generation

Input Sequence

grilled chicken with boiled

chicken with boiled potatoes

with boiled potatoes

Target Sequence

chicken with boiled potatoes

with boiled potatoes

boiled potatoes

By shifting the input sequence by one token, we can create the target sequence that the LSTM model will be trained to predict. This allows the model to learn the relationships between words and generate appropriate next words in a sequence.

An abstract graphic on the left side of the slide featuring a network of glowing nodes connected by thin lines, set against a purple and blue background with bokeh light effects.

In summary, the process of creating the training set for an LSTM language model involves shifting the input sequence by one token to generate the target variable. This allows the LSTM to learn the patterns and dependencies in the data, enabling it to predict the next word in a sequence. The LSTM architecture, with its Embedding and LSTM layers, is well-suited for this task, and analyzing the model's performance and internal states can provide valuable insights for further refinement and improvement.



Memory Cell

The core of an LSTM cell is the memory cell, which is responsible for storing and updating the state of the network as it processes sequential data.



Gates

LSTM cells use a series of gates to control the flow of information into and out of the memory cell, allowing the network to selectively remember and forget information as needed.



Input

The input to an LSTM cell is a combination of the current input data and the previous hidden state, which allows the network to maintain context and understanding as it processes sequential data.



Output

The output of an LSTM cell is a combination of the current state of the memory cell and the current hidden state, which can be used to make predictions or generate new text.

Long Short-Term Memory Networks for Text Generation

1

Capturing Context

Long short-term memory (LSTM) networks are a type of recurrent neural network that excel at modeling sequential data, such as text. By maintaining an internal state and selectively updating it, LSTMs are able to capture long-range dependencies and context, which is crucial for generating coherent and natural-sounding text.

2

Iterative Generation

To generate text, an LSTM model is trained on a corpus of text data. During generation, the model predicts the next token in the sequence conditioned on the previous tokens, iteratively building up the output text. This sequential, autoregressive approach allows the model to produce coherent and contextually-appropriate text.

3

Applications

LSTMs have been successfully applied to a wide range of text generation tasks, from creative writing to language translation and dialogue systems. Their ability to capture long-term dependencies and generate fluent, contextual output makes them a powerful tool for modeling and generating natural language.

Training LSTMs for Text Generation

1

Data Preparation

The first step in training an LSTM for text generation is to prepare the text data, which may involve tokenizing the text, creating a vocabulary, and encoding the text as numerical sequences that the LSTM can process.

2

Model Architecture

The LSTM model architecture must be designed to effectively capture the patterns and relationships in the text data, including the number and size of LSTM layers, the use of dropout and other regularization techniques, and the output layer for generating new text.

3

Training Process

The LSTM model is then trained on the prepared text data, using techniques such as teacher forcing and beam search to generate new text that is coherent and consistent with the patterns learned from the training data.



Evaluating LSTM Text Generation

Metric	Description
Perplexity	A measure of how well the model predicts the next word in a sequence, with lower values indicating better performance.
BLEU Score	A metric that compares the generated text to a reference corpus, evaluating the similarity of the generated text to the reference.
Human Evaluation	Ultimately, the quality of the generated text must be evaluated by human judges, who can assess the coherence, creativity, and overall quality of the output.



Conclusion [LSTMs]

LSTMs have proven to be a powerful tool for tackling a wide range of problems involving sequential data, including text generation. By understanding the structure and flow of language, LSTMs can be trained to generate coherent and meaningful text that can be used in a variety of applications, from creative writing to dialogue generation. As the field of natural language processing continues to evolve, the capabilities of LSTMs and other deep learning techniques will only continue to grow, opening up new and exciting possibilities for the future of text generation.

From Latent Variables to Sequential Modeling

Generative models like variational autoencoders (VAEs) and generative adversarial networks (GANs) have relied on the introduction of latent variables to capture the underlying structure of the data. In contrast, autoregressive models take a different approach, treating the generative process as a sequential task. Rather than decoding from a latent variable, autoregressive models condition their predictions on the previous values in the sequence, explicitly modeling the data generating distribution.

Advantages of Autoregressive Models

1 Explicit Modeling

Autoregressive models directly model the data generating distribution, rather than approximating it through a latent variable. This can lead to more accurate and stable generation, as the model is not constrained by the limitations of the latent space.

2 Flexibility

Autoregressive models can be applied to a wide range of data types, from text to images and beyond. Their sequential, context-aware approach makes them a versatile tool for generative modeling tasks.

3 Interpretability

The step-by-step, autoregressive generation process of these models can provide more interpretability and insight into how the model is generating the output, compared to more opaque latent variable models.

4 Controllability

Autoregressive models allow for fine-grained control over the generation process, as the output can be conditioned on specific inputs or constraints. This makes them useful for applications that require precise control over the generated content.



Challenges and Limitations

Computational Complexity

The sequential nature of autoregressive models can make them computationally intensive, as each output token or pixel must be generated conditioned on all the previous ones. This can limit their scalability and efficiency, especially for large-scale generation tasks.

Exposure Bias

Autoregressive models are trained on ground truth data, but during generation, they rely on their own predictions as input. This can lead to a compounding of errors, a phenomenon known as exposure bias, which can degrade the quality of the generated output over time.

Lack of Parallelism

Unlike models that generate output in a single pass, autoregressive models must generate each output element sequentially. This can make them slower and less efficient than parallel generation approaches, especially for tasks that can be easily parallelized.

Sensitivity to Priors

Autoregressive models can be sensitive to the choice of prior distributions and hyperparameters, which can significantly impact the quality and diversity of the generated output. Careful tuning and experimentation may be required to achieve optimal performance.



Pushing the Boundaries



Advancing the State-of-the-Art

Researchers are continuously exploring ways to improve the performance and capabilities of autoregressive models, such as developing more efficient architectures, addressing exposure bias, and exploring hybrid approaches that combine autoregressive modeling with other generative techniques.



Interdisciplinary Collaboration

The development of autoregressive models often involves collaboration across disciplines, including machine learning, natural language processing, computer vision, and more. This cross-pollination of ideas and techniques can lead to innovative breakthroughs in generative modeling.

Diverse Applications



Autoregressive models have found applications in a wide range of domains, from creative content generation to scientific data analysis. As the field continues to evolve, we can expect to see even more exciting and impactful applications of this powerful approach to generative modeling.

The Future of Generative Modeling



Autoregressive models represent a significant step forward in the field of generative modeling, and their continued development and integration with other techniques will likely shape the future of this rapidly advancing field.



Conclusion: Embracing the Power of Autoregressive Models

1

Explicit Modeling

Autoregressive models directly capture the data generating distribution, leading to more accurate and stable generation.

2

Flexibility

Autoregressive models can be applied to a wide range of data types, making them a versatile tool for generative modeling.

3

Interpretability

The step-by-step generation process of autoregressive models provides more insight into how the model generates output.



Working with Text Data

There are several key differences between text and image data that mean many of the methods that work well for image data are not so readily applicable to text data. Text data is composed of discrete chunks, such as characters or words, whereas pixels in an image are points in a continuous color spectrum. Text data also has a time dimension but no spatial dimension, the opposite of image data. Additionally, text data is highly sensitive to small changes, and has a rules-based grammatical structure, unlike image data.

Tokenization



1

Word Tokens

Tokenizing by words has pros and cons. It may require converting all text to lowercase, handling sparse words, and deciding how to treat punctuation.

2

Character Tokens

Character tokenization can generate new words outside the training vocabulary, and requires decisions about handling capitalization and punctuation.

3

Choosing the Right Approach

The choice of word or character tokenization will depend on the goals of the text generation model and the characteristics of the text data.

Tokenization Example

Pad Punctuation

Punctuation marks are tokenized separately by adding spaces around them.

Create TensorFlow Dataset

The text data is converted to a TensorFlow Dataset for efficient processing.

TextVectorization Layer

A Keras layer is used to convert the text to lowercase, map the 10,000 most common words to integer tokens, and trim or pad the sequences to a fixed length.

Challenges with Text Data

1

Discrete Chunks

Text data is composed of discrete words or characters, unlike the continuous pixel values in images.

2

Time Dimension

Text data has a sequential time dimension, unlike the spatial dimensions of images.

3

Sensitivity to Changes

Small changes to text can drastically alter the meaning, unlike with image data.

Grammatical Structure

Rules-Based

Text data follows strict grammatical rules, unlike the unconstrained pixel values in images.

Semantic Rules

Text data also has complex semantic rules that are difficult to model, such as the meaning and context of words.

Challenges for Generation

These grammatical and semantic rules make it very difficult to train models to generate coherent text, as every word is vital to the overall meaning.



Applying LSTMs to Text



Capture Dependencies

LSTMs can capture long-term sequential dependencies in text data, unlike feedforward neural networks.



Overcome Discreteness

LSTMs provide a way to work with the discrete nature of text data, where backpropagation is not as straightforward as with image data.



Handle Time Dimension

The sequential nature of text data, with a time dimension but no spatial dimension, is well-suited for LSTM architectures.



Preprocessing Text Data

1

Tokenization

The first step is to clean up and tokenize the text, splitting it into individual units like words or characters.

2

Vocabulary Mapping

The tokens are then mapped to a vocabulary of unique words or characters, often limiting the size to the most common ones.

3

Sequence Padding

The sequences of tokens are then trimmed or padded to a fixed length, so they can be efficiently processed by the LSTM model.

Generating Text with LSTMs

Challenges	Strategies
Discrete Tokens	Use a softmax output layer to predict the next token
Maintaining Coherence	Leverage the LSTM's ability to capture long-term dependencies
Avoiding Repetition	Implement techniques like beam search or top-k sampling

While text data presents unique challenges compared to image data, the sequential nature and discrete structure of text can be effectively modeled using LSTM architectures. By carefully preprocessing the text data and leveraging the capabilities of LSTMs, we can build powerful text generation models that can produce coherent and contextually relevant output.



Convolutional Neural Networks for Image Generation

Convolutional neural networks (CNNs) have proven to be incredibly powerful for feature extraction and image recognition tasks. However, applying them directly to image generation in an autoregressive manner is challenging, as CNNs do not inherently have a sense of ordering or directionality. This section introduces two key extensions to CNNs that enable their use for generative tasks: Masked Convolutional Layers and Residual Blocks.

PixelCNN: Autoregressive Image Generation

Modeling Pixel Dependencies

While LSTMs excel at text generation, PixelCNN is an autoregressive model designed for image generation. PixelCNN models the joint distribution of pixels in an image by predicting each pixel conditioned on all the previous pixels in a raster-scan order. This allows the model to capture the complex dependencies between pixels and generate realistic-looking images.

Efficient Generation

Unlike models that generate images from a latent space, PixelCNN generates images pixel-by-pixel in a sequential manner. This makes the generation process more efficient and allows for fine-grained control over the output. PixelCNN can also be used to generate images of arbitrary size, making it a versatile tool for image synthesis.

Applications

PixelCNN has been applied to a variety of image generation tasks, from generating natural images to synthesizing high-resolution medical images. Its ability to capture complex pixel dependencies and generate realistic-looking output makes it a powerful tool for generative modeling in the visual domain.

Masked Convolutional Layers

1 Ordering Pixels

To apply CNNs to image generation, we must first impose an ordering on the pixels, such as top-left to bottom-right.

2 Masking Filters

The convolutional filters are then masked so that each pixel's output is only influenced by the pixels that precede it in the chosen ordering.

3 Two Mask Types

There are two types of masks: Type A, where the central pixel is masked, and Type B, where the central pixel is not masked.

Residual Blocks

Skip Connections

Residual blocks include a "skip connection" that allows the input to bypass the intermediate layers and be added to the output. This makes it easier for the network to learn an identity mapping.

Optimal Transformations

If the optimal transformation is to keep the input the same, the network can simply zero the weights of the intermediate layers, rather than having to learn an identity mapping.

Building Blocks

Residual blocks are the core building blocks used to construct the PixelCNN architecture, which applies CNNs to image generation in an autoregressive manner.

Training the PixelCNN

1

Autoregressive Design

The PixelCNN is designed to generate images one pixel at a time, with each pixel's value predicted based on all the preceding pixels in the image.

2

Output Layer

The original PixelCNN paper used a 256-filter convolutional output layer with softmax activation to predict the exact pixel value. This can be slow to train.

3

Simplified Output

In our implementation, we use a 4-filter convolutional output layer to simplify the task and speed up training, at the cost of reduced color fidelity.



Mixture

Pixel Value Challenges

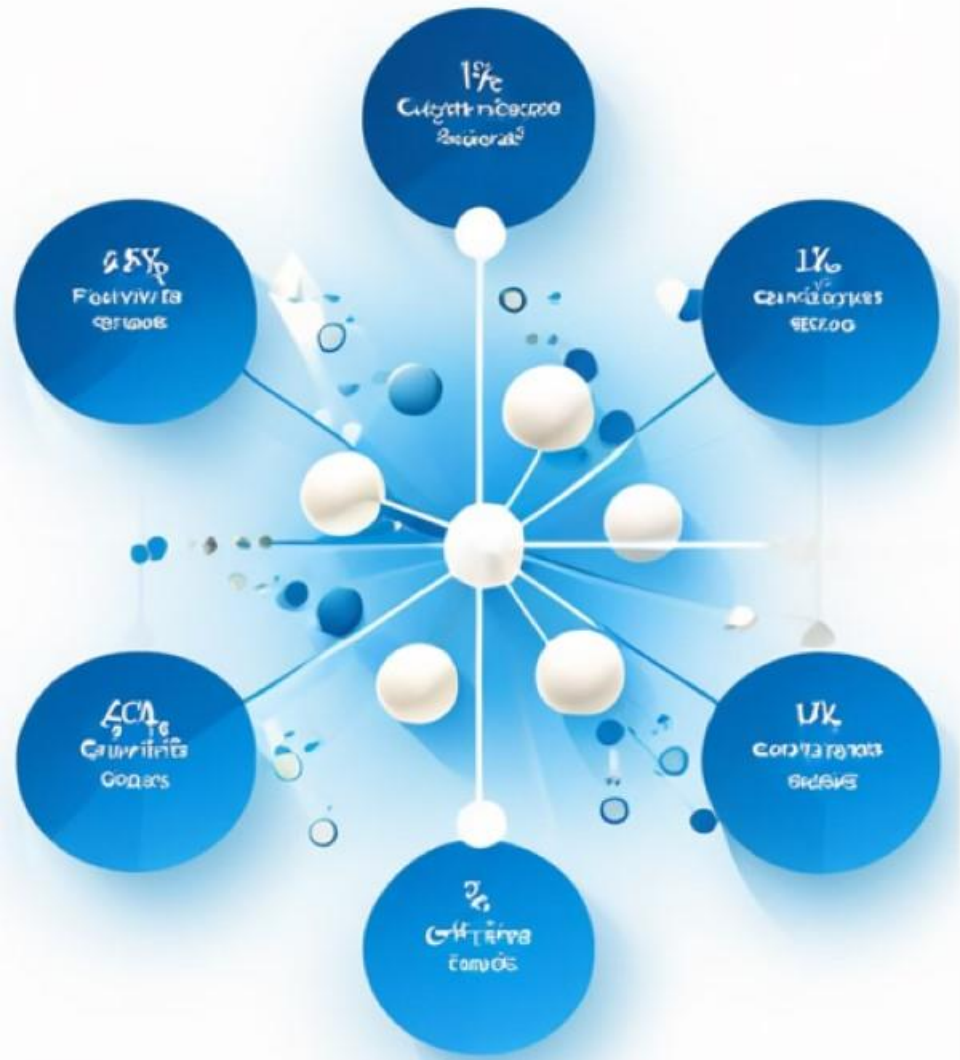
Treating each pixel value as an independent categorical distribution can be slow to train, as the network must learn the full distribution over 256 possible values.

Mixture Distributions

To address this, we can use a mixture distribution, which combines multiple probability distributions (e.g., logistic distributions) with a categorical distribution to choose between them.

Improved Flexibility

Mixture distributions provide more flexibility and can better capture the continuous nature of pixel values, leading to faster and more effective training.



PixelCNN Extensions



Stacked PixelCNNs

Stacking multiple PixelCNN layers can improve the model's ability to capture complex dependencies in the data.



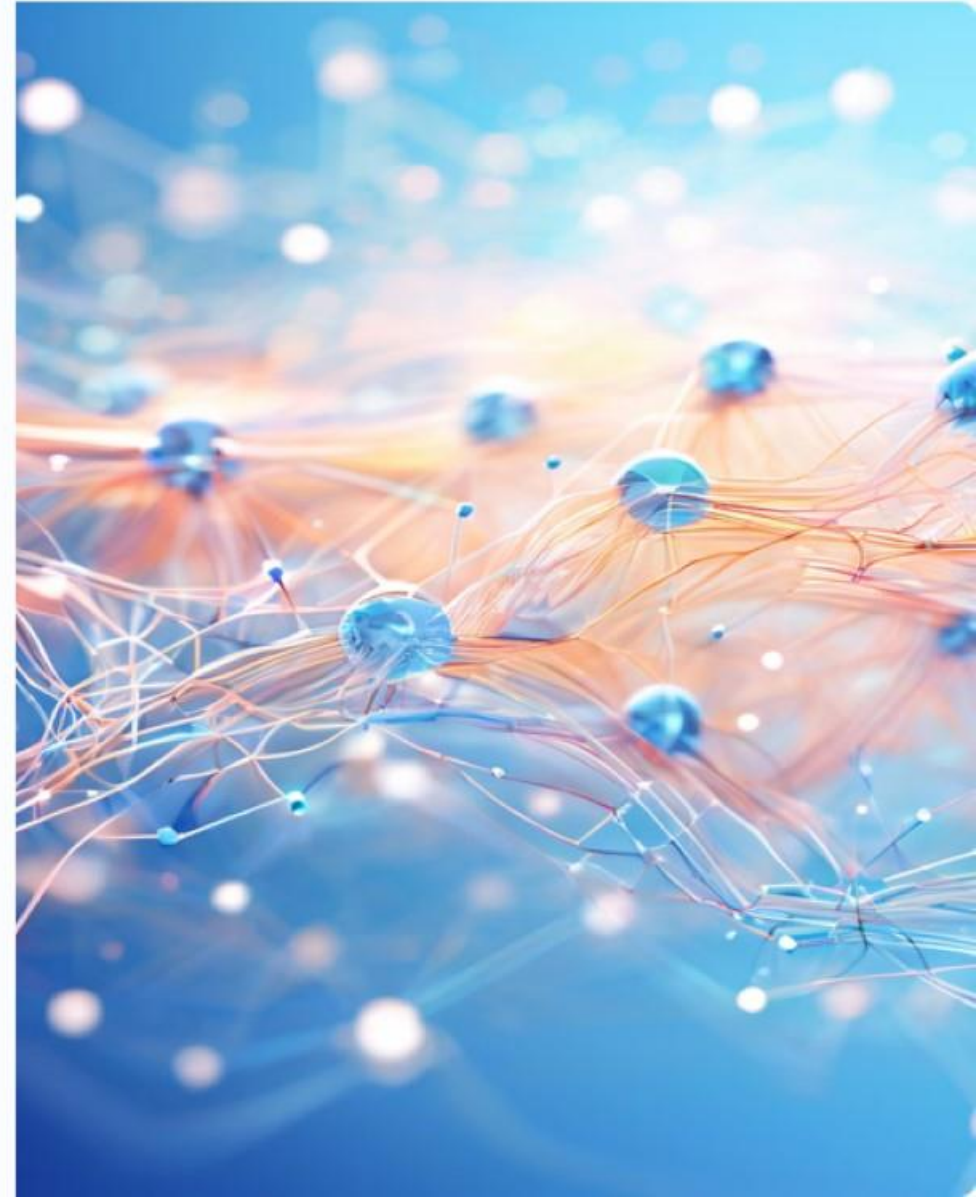
Conditioning

Conditioning the PixelCNN on additional information, such as class labels or other inputs, can help guide the generation process.



Hierarchical Models

Hierarchical PixelCNN models that generate images in a coarse-to-fine manner can improve efficiency and quality.



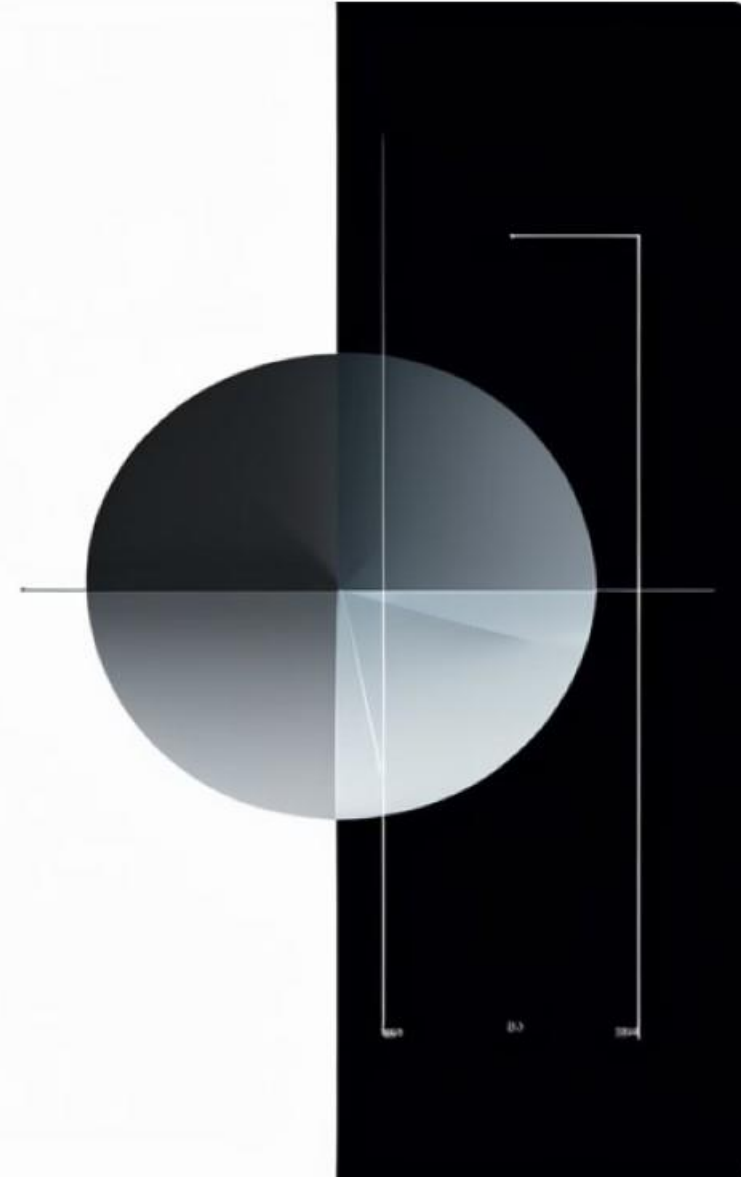
The Limitations of Early RNNs

1 Vanishing Gradient Problem

When RNNs were first introduced, recurrent layers were very simple and consisted solely of a tanh operator that ensured that the information passed between timesteps was scaled between -1 and 1 . However, this approach was shown to suffer from the vanishing gradient problem and didn't scale well to long sequences of data.

2 Difficulty with Long Sequences

The simple recurrent layers in early RNNs struggled to handle long sequences of data, as information would gradually fade over time, making it difficult for the network to maintain context and make accurate predictions.



Recurrent Neural Network (RNN)

Extensions

1

Sequence Modeling

Recurrent neural networks (RNNs) are well-suited for modeling sequential data, such as text, due to their ability to maintain a hidden state that captures dependencies over time.

2

Limitations

However, standard RNNs can struggle with long-range dependencies and may have difficulty capturing complex patterns in the data.

3

Extensions

Various extensions to the basic RNN architecture, such as LSTMs and GRUs, have been developed to address these limitations and improve the model's performance on a wide range of sequence modeling tasks.



PixelCNN vs. RNNs

PixelCNN

The PixelCNN applies convolutional layers with masked filters to generate images one pixel at a time in an autoregressive manner, capturing spatial dependencies.

RNNs

Recurrent neural networks (RNNs) are well-suited for modeling sequential data, such as text, by maintaining a hidden state that captures dependencies over time.

Comparison

While both models can be used for generative tasks, they differ in their underlying architectures and the types of data they are best suited for.

Conclusion

In this chapter, we have explored two key extensions to convolutional neural networks that enable their use for image generation tasks: Masked Convolutional Layers and Residual Blocks. These techniques, combined with the PixelCNN architecture and the use of mixture distributions, provide a powerful framework for generating images one pixel at a time in an autoregressive manner. We have also briefly discussed how recurrent neural networks (RNNs) and their extensions can be applied to sequence modeling tasks, and how the PixelCNN and RNN approaches differ in their underlying architectures and the types of data they are best suited for.

Thank You