AI Decoded - Making Sense of Deep Learning and Generative AI

Yiqiao Yin

Affiliation: Columbia University, University of Chicago - Booth School of Business

Keywords: Neural Networks, Convolutional Neural Networks, Sequential Models, Attention Mechanism, Large Language Models, Foundation Models, Generative AI

May 9, 2024

Failure is instructive. The person who really thinks learns quite as much from his failures as from his successes. – John Dewey

Overview

Artificial Intelligence: A Historical Overview

- AI, as conceived from its inception in the mid-20th century until the 1970s, pertains to the endeavor of constructing machines that can emulate tasks typically necessitating human intellect.
- The early luminaries of the discipline, including Alan Turing, John McCarthy, Marvin Minsky, and Allen Newell, delineated its foundational tenets.
- Turing's seminal proposition in 1950, known as the Turing Test, postulated a criterion wherein a machine's intelligence is gauged by its indistinguishability from human behavior (Gardner, 1987) [6].
- Much of AI's early research was anchored in symbolic systems, aiming to represent human problem-solving methodologies through the manipulation of symbols.
- The aspiration was that intelligence could be abstracted as a set of symbol manipulations, enabling computers to mimic cognitive tasks.
- There existed a keen interest in crafting "Expert Systems"—computational constructs designed to encapsulate and replicate the decision-making process of human specialists in distinct domains.
- The epoch in AI was characterized by a dual pursuit: achieving General AI with comprehensive human-like cognitive abilities, and devising Narrow AI tailored to singular, specialized tasks.
- General AI remained largely aspirational, while Narrow AI saw tangible progress and remains predominant today (Sotala, 2017) [13].

Neural Networks History

- Neural networks, as a concept, trace their origins back to the 1940s.
- The initial idea was inspired by the neural structures in the human brain.
- Warren McCulloch and Walter Pitts introduced the first mathematical model of a neural network in their 1943 paper, "A Logical Calculus of the Ideas Immanent in Nervous Activity.".
- This foundational work proposed simplified neurons as binary threshold units.
- These units could be combined to perform logical functions.

Yann LeCun and Convolutional Neural Networks

- Yann LeCun played a pivotal role in popularizing and advancing neural networks, particularly convolutional neural networks (CNNs) (LeCun Y. B., 1989) [11].
- In the 1980s and 1990s, LeCun introduced the LeNet architecture, which was one of the first CNNs.
- LeNet was successfully applied to the problem of handwritten digit recognition for the United States Postal Service, setting benchmarks in performance (LeCun Y. et. al., 1995) [10].
- LeCun's work highlighted the potential of CNNs in processing grid-like topology data, such as images.
- CNNs can automatically and adaptively learn spatial hierarchies of features, which is revolutionary for image and speech recognition.
- With increased computational power and large datasets, CNNs, backed by LeCun's foundational work, became a cornerstone for deep learning applications in various domains.
- Yann LeCun, along with Geoffrey Hinton and Yoshua Bengio, was awarded the Turing Award in 2018 for their work on deep learning (Schmidhuber, 2022) [12].

Single Layer Perceptron

- The mathematical formula for a single layer perceptron can be represented as follows:

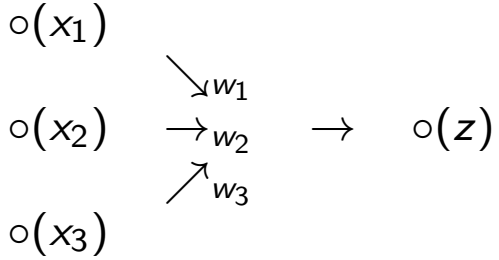
$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$\hat{y} = \sigma(z)$$

- Notations:
  - z - Weighted sum of inputs plus bias
  - w_i - Weight associated with input x_i
  - x_i - Input features
  - b - Bias
  - $\hat{y}$ - Output prediction
  - $\sigma(\cdot)$ - Activation function (e.g., sigmoid, step function, etc.)

Single Layer Perceptron

- Graphical demonstration:



- As portrayed above, one layer has many neurons, each with an arrow passing information towards one neuron in the next layer. Mathematically, we can write

$$z = x_1 w_1 + x_2 w_2 + x_3 w_3$$

- We can also simplify the expression using the summation symbol:

$$z = \sum_{j=1}^{3} x_j w_j$$

- Suppose we choose the sigmoid function as the activation function. The output of the last neuron would produce:

$$\hat{y} = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

Loss Function: Mean Square Error

- The mean square error (MSE) is a commonly used loss function in machine learning and optimization tasks.
- It measures the average squared difference between the predicted values and the actual values in a dataset.
- Mathematically, the MSE is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where:

  - n is the number of samples in the dataset,
  - y_i represents the actual value of the target variable for the i-th sample,
  - $\hat{y}_i$ represents the predicted value of the target variable for the i-th sample.

- The MSE penalizes large errors more heavily due to the squaring operation, making it sensitive to outliers.
- It is often used in regression problems to quantify the goodness of fit of a model.

Explanation of Notations and Partial Derivatives

- In the context of optimization algorithms such as gradient descent, it's essential to understand the notations and partial derivatives involved.
- Let's consider a simple linear regression model with parameters θ_0 and θ_1, where θ_0 represents the intercept and θ_1 represents the slope.
- The predicted value $\hat{y}$ for a given input x can be expressed as:

$$\hat{y} = \theta_0 + \theta_1 x$$

- The objective is to minimize the MSE loss function with respect to the parameters θ_0 and θ_1.
- To do this, we compute the partial derivatives of the MSE with respect to each parameter and update the parameters iteratively using gradient descent.

Taking Derivatives of Mean Square Error Loss Function

- We start with the mean square error (MSE) loss function:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  where y_i are the true values and $\hat{y}_i$ are the predicted values.
- Taking the derivative of MSE with respect to each parameter involves the chain rule and simplification.
- For a simple linear regression with parameters θ_0 and θ_1, we find:

$$\frac{\partial \text{MSE}}{\partial \theta_0} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

$$\frac{\partial \text{MSE}}{\partial \theta_1} = -\frac{2}{n}\sum_{i=1}^{n}x_i(y_i - \hat{y}_i)$$

Final Partial Derivatives with Respect to Each Parameter

- The final partial derivatives with respect to each parameter are obtained by substituting the expressions for $\hat{y}_i$ into the above derivatives.
- For a linear regression model with parameters θ_0 and θ_1, we get:

$$\frac{\partial \text{MSE}}{\partial \theta_0} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - (\theta_0 + \theta_1 x_i))$$

$$\frac{\partial \text{MSE}}{\partial \theta_1} = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (\theta_0 + \theta_1 x_i))$$

- These derivatives are used in gradient descent optimization to update the parameters iteratively.

Loss Function: Binary Cross Entropy

- The binary cross entropy loss function is commonly used in binary classification problems.
- It measures the difference between two probability distributions: the predicted probabilities (output of the model) and the actual labels.
- Mathematically, it is defined as:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

where:

- y_i is the true label for the i-th sample (0 or 1),
- $\hat{y}_i$ is the predicted probability of the positive class for the i-th sample,
- N is the number of samples.

Loss Function: Binary Cross Entropy (Cont'd)

- In the loss function:
  - $y_i \log(\hat{y}_i)$ penalizes the model when the predicted probability of the positive class is low for true positive instances,
  - $(1 - y_i) \log(1 - \hat{y}_i)$ penalizes the model when the predicted probability of the negative class is low for true negative instances.
- The goal is to minimize this loss function to improve the model's ability to correctly classify instances.
- Partial derivatives of the loss function with respect to model parameters are used in gradient-based optimization algorithms like gradient descent to update the parameters during training.

Derivatives of Cross-Entropy Loss Function

- Let's consider the cross-entropy loss function for binary classification:

$$L(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

  where y is the true label (0 or 1) and $\hat{y}$ is the predicted probability.
- To find the derivatives of this loss function with respect to the model parameters, we start by computing the partial derivative of L with respect to $\hat{y}$:

$$\frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1 - y}{1 - \hat{y}}\right)$$

- Next, we compute the partial derivatives of $\hat{y}$ with respect to the model parameters. Depending on the model (e.g., logistic regression), this may involve applying the chain rule.
- Finally, we can compute the final partial derivatives of the loss function with respect to each parameter using the chain rule and the computed derivatives of $\hat{y}$.

Final Partial Derivatives

- For each parameter w_i, where i represents the index of the parameter vector:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_i}$$

- After substituting the expression for $\frac{\partial L}{\partial \hat{y}}$ and $\frac{\partial \hat{y}}{\partial w_i}$, we obtain the final partial derivative with respect to w_i.
- Similarly, we compute the partial derivatives for other parameters in the model, such as biases b or regularization terms.
- These final partial derivatives are crucial for updating the model parameters during the optimization process, such as gradient descent.

Introduction

- In the annals of neural network research, LeCun's introduction of the LeNet architecture stands out as a monumental stride. This architecture, one of the inaugural convolutional neural networks, was adeptly applied to the task of handwritten digit recognition. Specifically, its deployment for the United States Postal Service to automate the recognition of handwritten digits was met with laudable accuracy, thus showcasing the pragmatic efficacy of CNNs.
- LeNet's architecture leveraged the spatial hierarchies inherent in image data, enabling the network to learn distinguishing features at varying levels of granularity. This inherent capability of CNNs to automatically and adaptively discern spatial hierarchies of features set them apart and underscored their promise in tasks encompassing image and speech recognition.

Introduction

- LeCun's relentless promotion of neural networks, spanning scholarly publications, lectures, and collaborations, was instrumental in highlighting their potential within the scientific community.
- His efforts transcended theoretical discourse, actively bridging the gap between theory and practical application, thus reigniting interest in neural networks, particularly during periods of skepticism.
- LeCun's pivotal contributions have been acknowledged as pivotal in the rise of deep learning, significantly influencing the course of artificial intelligence research in subsequent years.

Introduction

- Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision with their ability to effectively recognize patterns in images.
- VGG16 is one such model, designed to be both simple and deep for improved performance.
- Imagine teaching a child to recognize cars; you might start with small details like wheels and windows and gradually build up to the full picture. VGG16 works similarly, using layers that focus on small details and combining them as it goes deeper to understand the whole image.
- The "16" in VGG16 refers to the number of layers that have weights; this depth helps the model learn a hierarchy of features, from simple to complex.
- However, the simplicity of VGG16's uniform architecture comes with a cost. Its depth means that it's computationally intensive and requires a lot of memory to operate, which can make it slower to train and use compared to newer models.
- Additionally, because it has so many layers, it's more prone to overfitting, where the model learns the training data too well, including the noise and outliers, which can decrease its performance on new, unseen images.

Convolution Operation

- Continuous Form using Integration:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau)\, d\tau$$

Here, $f(t)$ and $g(t)$ are continuous functions, and the convolution operation $(f * g)(t)$ produces a new function of time t. The integral is taken over all possible values of τ.

- Discrete Form using Summation:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n - m]$$

In the discrete domain, $f[n]$ and $g[n]$ are sequences, typically represented as vectors or matrices in a digital signal processing context. The convolution operation $(f * g)[n]$ computes a new sequence indexed by n. The summation is taken over all possible values of m.

Convolutional Operation in Convolutional Neural Networks

- In convolutional neural networks (CNNs), the convolutional operation is fundamental for feature extraction.
- To understand convolution in CNNs, let's draw upon its statistical background.
- Convolution, in a general sense, is an operation that blends two functions to produce a third, representing how one function modifies the shape of another.
- In the context of image processing, the convolution operation involves sliding a filter (also called a kernel) over an input image.
- At each position, the filter computes the dot product between its weights and the corresponding pixel values in the input image.
- This dot product represents a measure of similarity between the filter and the local image patch, akin to a statistical correlation.
- By sliding the filter across the entire image, we obtain a feature map that highlights regions of similarity between the input image and the filter.
- Through this process, CNNs can automatically learn features hierarchically, capturing increasingly abstract patterns as we move deeper into the network.

Elementwise Matrix Multiplication

- Let's suppose we have two matrices:
  - Matrix A with random integers, size 3×3:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

  - Matrix B with random integers, size 3×3:

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

- The elementwise multiplication of A and B, denoted $A \odot B$, is calculated as follows:

$$A \odot B = \begin{bmatrix} a_{11} \times b_{11} & a_{12} \times b_{12} & a_{13} \times b_{13} \\ a_{21} \times b_{21} & a_{22} \times b_{22} & a_{23} \times b_{23} \\ a_{31} \times b_{31} & a_{32} \times b_{32} & a_{33} \times b_{33} \end{bmatrix}$$

Introduction

- ResNet, short for Residual Network, introduces a novel approach to training deep networks. It employs "residual connections" that allow it to learn from the final outcome and adjust the earlier layers accordingly. This design mitigates the vanishing gradient problem, enabling training of very deep networks with hundreds of layers.
- The strengths of ResNet lie in its ability to be deeper without increasing training difficulty, performing exceptionally well on various image recognition tasks. However, its computational requirements, especially for deeper versions, can be substantial. Additionally, its complex architecture may pose challenges for beginners in the field.

Introduction

- DenseNet, which stands for Densely Connected Convolutional Network, takes inspiration from the fact that having more connections and information paths can improve learning.
- In a school project, for instance, if each student's work is built on top of the others' work, the final project can turn out to be very comprehensive.
- DenseNet applies this by connecting each layer to every other layer in a feed-forward fashion.
- Unlike traditional models where the output of one layer is only passed to the next one, DenseNet reuses features from all previous layers by connecting them to all subsequent layers, significantly improving efficiency.
- This leads to substantial reductions in the number of parameters since there's no need to relearn redundant feature maps.
- DenseNet is particularly good at image classification tasks with fewer training samples and can be more efficient in terms of computation compared to other complex models.
- However, DenseNet's unique architecture leads to a large increase in memory and computational time during training, since each layer's output is concatenated to all subsequent layers.
- This can become problematic with very deep networks or very large images.

Introduction to Inception

- The Inception model is analogous to a multi-lane highway, with each lane having a different speed limit, enabling cars to select the most efficient route.
- Inception networks consist of "Inception modules" designed to perform multiple convolutions in parallel across various scales, ranging from 1x1 to 5x5 filters, and then merge the results.
- This architecture empowers the network to discern the optimal scale for each segment of an image, facilitating the capture of complex patterns at different sizes and resolutions.
- Inception's computational efficiency stems from dimensionality reduction before employing more resource-intensive 3x3 and 5x5 convolutions, making it suitable for resource-constrained environments like mobile devices.
- However, the complexity of Inception can pose challenges for implementation and fine-tuning due to numerous hyperparameters and architectural decisions, which may overwhelm practitioners new to model architecture design.

Common Characteristics of CNN Models

- Use of Convolutional Layers:
  - Foundational concept for feature extraction.
  - Filters capture spatial hierarchies.
- Depth:
  - Deep architectures with many layers.
  - Learn simple to complex features.
- Ability to Handle High-Dimensionality:
  - Designed for high-dimensional input data.
  - Employ techniques to manage complexity.

Common Characteristics of CNN Models (Cont'd)

- Use of Non-Linear Activation Functions:
  - Incorporate functions like ReLU.
  - Crucial for learning complex datasets.
- End-to-End Training:
  - Trained with backpropagation.
  - Maps raw pixel values to class scores.
- Generalization Capabilities:
  - Trained on large-scale datasets.
  - Show remarkable performance on various tasks.

Common Characteristics of CNN Models (Cont'd)

- Transfer Learning:
  - Often used as pre-trained models.
  - Fine-tuning for related tasks.
- Unique Approaches and Trade-offs:
  - Each model has unique processing approaches.
  - Trade-offs in efficiency, performance, and ease of training.
- Architectural Innovations:
  - Responses to challenges of training deeper networks.
  - Skip connections in ResNet, dense connections in DenseNet, etc.

Potential Research Questions

- Scalability and Efficiency:
  - As datasets and tasks' resolutions increase, the scalability of CNNs becomes critical.
  - How to scale CNNs efficiently to handle larger datasets and higher resolution images without a proportional increase in computational cost?
  - Questions of building smaller, more efficient networks without compromising performance (model compression and pruning).
- Generalization Ability:
  - Despite performance on benchmark datasets, CNNs can struggle with generalization due to dataset biases.
  - How to make CNNs more robust to biases and better generalize to new tasks or environments?

Potential Research Questions (Cont'd)

- Explainability and Transparency:
  - Deep CNNs lack interpretability, especially in sensitive fields like healthcare or autonomous driving.
  - How to develop techniques for making CNNs more explainable and transparent?
- Integration with Other Methods:
  - Combining CNNs with other ML approaches like reinforcement learning, GANs, or graph neural networks can lead to improvements.
  - What are the best ways to integrate CNNs with these methods to enhance unsupervised learning, domain adaptation, and few-shot learning?

Introduction to Recurrent Neural Networks (RNNs)

- Neural Networks Outside of Image Data: Recurrent Neural Networks
- Recurrent Neural Networks (RNNs) are designed for recognizing patterns in sequences of data.
- RNNs maintain a "memory" of previous inputs, facilitating the processing of sequences.
- RNNs can exhibit temporal dynamic behavior due to their ability to maintain an internal state.

History and Development of RNNs

- RNNs originated in the 1980s with the work of Rumelhart, Hinton, and Williams (Tyagi, 2022) [15].
- They introduced the backpropagation through time (BPTT) method for training RNNs.
- RNNs addressed limitations of feedforward neural networks in handling sequential data.
- The looping mechanism of RNNs allows them to capture sequential dependencies.

Applications of RNNs

- RNNs are used in natural language processing to understand contextual dependencies.
- They are applied in time series prediction to capture historical trends.
- Examples of applications include speech recognition and financial time series analysis.
- RNNs provide a solution for handling inherently sequential data in various fields.

Recurrent Neural Network Neuron

- Let's denote the input at time step t as x_t and the output of the neuron at time step t as y_t.
- The neuron's computation involves two main transformations:
  - Linear Transformation:
    $$z_t = W_{xz}x_t + W_{yz}y_{t-1} + b_z$$
    Here, W_{xz} and W_{yz} are weight matrices for the input and the previous output respectively, and b_z is the bias term. z_t represents the pre-activation or hidden state of the neuron.
  - Non-linear Transformation (Activation):
    $$y_t = f(z_t)$$
    Where $f(\cdot)$ is an activation function, typically a non-linear function like the sigmoid (σ), tanh, or ReLU (Rectified Linear Unit).

Recurrent Neural Network

- Consider data with time stamp

$$X_{\langle 1 \rangle} \to X_{\langle 2 \rangle} \to \cdots \to X_{\langle T \rangle}$$

and feed-forward architecture pass information through exactly as the following:

$$\begin{array}{ccccccccc} & & Y_{\langle 1 \rangle}, \hat{Y}_{\langle 1 \rangle} & & Y_{\langle 2 \rangle}, \hat{Y}_{\langle 2 \rangle} & & Y_{\langle T \rangle}, \hat{Y}_{\langle T \rangle} & & \\ & & \uparrow & & \uparrow & & \uparrow & & \\ \text{Info. in:} \to & X_{\langle 1 \rangle} & \to & X_{\langle 2 \rangle} & \to & \cdots \to & X_{\langle T \rangle} & \to & \underbrace{Y_{\langle T \rangle}}_{\text{Form beliefs about}} \\ & \uparrow & & \uparrow & & & \uparrow & & \\ & w_{\langle 1 \rangle}, b_{0,\langle 1 \rangle} & & w_{\langle 2 \rangle}, b_{0,\langle 2 \rangle} & & & w_{\langle T \rangle}, b_{0,\langle T \rangle} & & \end{array}$$

Backpropagation Through Time

- Let us clearly define our loss function to make sure we have a proper grip of our mistakes.

$$\mathcal{L} = \sum_t L(\hat{y}_{\langle t \rangle} - y_t)^2$$

and we can compute the gradient

$$\nabla = \frac{\partial \mathcal{L}}{\partial a}$$

and then with respect with parameters w and b

$$\frac{\partial \nabla}{\partial w}, \frac{\partial \nabla}{\partial a}$$

and now with perspective of where we make our mistakes according to our parameters we can go backward

$$\begin{array}{lccccccc} \text{Info. in:} \leftarrow & Y_{\langle 1 \rangle}, \hat{Y}_{\langle 1 \rangle} & & Y_{\langle 2 \rangle}, \hat{Y}_{\langle 2 \rangle} & & Y_{\langle T \rangle}, \hat{Y}_{\langle T \rangle} & & \\ & \downarrow & & \downarrow & & \downarrow & & \\ & X_{\langle 1 \rangle} \leftarrow & & X_{\langle 2 \rangle} \leftarrow & \cdots \leftarrow & X_{\langle T \rangle} & \leftarrow & \underbrace{Y_{\langle T \rangle}}_{\text{Form beliefs about}} \\ & \downarrow & & \downarrow & & \downarrow & & \\ \text{Update:} & w_{\langle 1 \rangle}, b_{0,\langle 1 \rangle} & & w_{\langle 2 \rangle}, b_{0,\langle 2 \rangle} & & w_{\langle T \rangle}, b_{0,\langle T \rangle} & & \end{array}$$

Loss Function of RNN

- The loss function $\mathcal{L}$ of a Recurrent Neural Network (RNN) is typically defined as:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N} y_{i,t}\log(\hat{y}_{i,t})$$

  where T is the sequence length, N is the number of output classes, $y_{i,t}$ is the true label, and $\hat{y}_{i,t}$ is the predicted probability of class i at time step t.
- $\mathcal{L}$ represents the loss function, which measures the discrepancy between the predicted and true labels.
- T denotes the sequence length, indicating the number of time steps in the input sequence.
- N is the number of output classes, indicating the dimensionality of the output vector.
- $y_{i,t}$ represents the true label of class i at time step t.
- $\hat{y}_{i,t}$ represents the predicted probability of class i at time step t.

Taking Partial Derivative

- To optimize the RNN parameters, we need to compute the partial derivative of the loss function with respect to each parameter.
- This involves the application of the chain rule to propagate gradients through time.
- Specifically, we compute the derivative of the loss with respect to each weight matrix and bias vector in the network.
- The process of taking partial derivatives involves backpropagating errors through the network.
- We compute the gradient of the loss function with respect to the output of each time step and then propagate these gradients backward through time.
- Finally, we update the network parameters using an optimization algorithm such as gradient descent.

Derivative of Loss Function in RNNs

- The partial gradient of the loss function with respect to the parameters of a Recurrent Neural Network (RNN) is crucial for training the model.
- Let L denote the loss function and θ represent the parameters of the RNN.
- The derivative of the loss function L with respect to θ is computed using backpropagation through time (BPTT) or other optimization algorithms.
- The partial gradient $\frac{\partial L}{\partial \theta}$ represents the sensitivity of the loss function to changes in the parameters θ of the RNN.
- This gradient is used in the optimization process, such as gradient descent, to update the parameters of the RNN and improve its performance on the task at hand.
- The attention mechanism emerged as a salient advancement in deep learning, particularly in sequence-to-sequence tasks like machine translation.
- It dynamically weighs the significance of different input elements in relation to a given output, addressing limitations of fixed-size intermediate representations in traditional neural network architectures.

Introduction to Attention Mechanism (Cont'd)

- The attention mechanism allows the decoder to "attend" to different parts of the source sentence at each step of output generation, providing a dynamic, adaptive memory.
- Its significance extends beyond machine translation, influencing tasks in natural language processing, computer vision, and beyond, laying the groundwork for models like OpenAI's GPT and Google's BERT.
- The attention layer is crucial to GPT-based models due to its deep integration with the Transformer architecture.
- The Transformer architecture, introduced in "Attention Is All You Need" by Vaswani et al. (2017), revolutionized sequence modeling by leveraging attention mechanisms.
- Unlike traditional models relying on recurrent structures, Transformers use attention to capture dependencies in input data without sequential processing.

Importance of Attention Layer in GPT-based Models (cont.)

- The attention mechanism enables each item in the input data to focus on different parts, facilitating the modeling of various relationships and dependencies.
- Transformers overcome limitations of recurrent architectures by mitigating long-term dependency challenges and offering computational advantages.
- Transformers, with their attention layers, have become a cornerstone in deep learning, driving advancements in natural language processing and machine translation.

Autoencoders (AE)

- **Autoencoder Concept:** An autoencoder is a type of artificial neural network used for unsupervised learning. It consists of an encoder network that compresses the input data into a lower-dimensional representation, called the latent space, and a decoder network that reconstructs the original input from this representation. The objective of an autoencoder is to learn a compact and efficient representation of the input data.
- **Need for Autoencoders:** Autoencoders are utilized in various applications for several reasons. Firstly, they can be used for data compression and denoising, where they learn to filter out noise from input data and reconstruct clean versions. Secondly, autoencoders are valuable for feature learning and representation, enabling the discovery of meaningful patterns and structures in the data. Additionally, they serve as powerful generative models, capable of generating new data samples similar to the training data.
- **Hidden Connection with PCA:** There exists a fundamental connection between autoencoders and Principal Component Analysis (PCA). Both methods aim to capture the most important features or components of the data while reducing its dimensionality. In fact, under certain conditions, an autoencoder with a linear activation function and mean squared error loss is equivalent to performing PCA. This highlights the ability of autoencoders to perform nonlinear dimensionality reduction, capturing complex relationships in the data beyond what PCA can achieve.

Autoencoders (AE): Mathematical Formulation

- An autoencoder is a neural network architecture used for unsupervised learning that aims to learn efficient representations of input data.
- Let X represent the input data, which typically resides in a high-dimensional space.
- The autoencoder consists of two main components: an encoder f and a decoder g.
- The encoder f maps the input data X to a lower-dimensional latent space representation z, i.e., $z = f(X)$.
- The decoder g then maps the latent space representation z back to the original input space, producing the reconstructed data X', i.e., $X' = g(z)$.
- Mathematically, the autoencoder can be represented as follows:

$$X' = g(f(X))$$

- Here, X' represents the reconstructed data, f denotes the encoder function, and g denotes the decoder function.

Motivation of Variational Autoencoder

- The motivation of variational autoencoders (VAEs) lies in their ability to generate new data samples while also learning useful representations.
- **Autoencoder Mathematical Form**:
  - Encoder: $z = f_{\text{enc}}(X)$
  - Decoder: $\hat{X} = f_{\text{dec}}(z)$
  - Putting everything together: $\hat{X} = f_{\text{dec}}(f_{\text{enc}}(X)$
- **Notation**:
  - X: High-dimensional data (e.g., images with noise)
  - z: Latent representation (lower-dimensional space)
  - $\hat{X}$: Reconstructed data
- **Use Case**: Reconstructing Images from Noisy Inputs
  - Input: X (noisy images)
  - Output: $\hat{X}$ (reconstructed images)

Motivation and Overview of U-Net

- Motivation for Image-to-Image Models:
  - Image-to-image models aim to solve various tasks such as image segmentation, image denoising, and image super-resolution.
  - These tasks require mapping an input image to an output image, preserving important features and structures.
- Why Use U-Net?
  - U-Net is a popular architecture for image-to-image tasks due to its effectiveness and efficiency.
  - It utilizes a U-shaped architecture with skip connections, inspired by residual networks.
  - Skip connections allow for the preservation of fine-grained details and help overcome the vanishing gradient problem.
- Variations of U-Net:
  - There are several variations of U-Net tailored for specific tasks and performance improvements.
  - Variations may include changes in network depth, width, and incorporation of additional modules like attention mechanisms.
- Mathematical Representation of U-Net:
  - Let X be the input image and Y be the corresponding output image.
  - U-Net can be represented as a function $F : X \to Y$ mapping input images to output images.
  - The network consists of an encoder-decoder architecture with skip connections, enabling high-resolution feature reconstruction.

Introduction

- Motivation of Vision Transformer:
  - Traditional Convolutional Neural Networks (CNNs) have been the backbone of many computer vision tasks.
  - However, as images grow in resolution and complexity, CNNs face challenges in capturing long-range dependencies.
  - Vision Transformer (ViT) aims to address this limitation by leveraging self-attention mechanisms inspired by Transformer models, originally designed for natural language processing tasks.
- Mechanism of Generating Small Patches:
  - The original image is divided into smaller patches using a sliding window approach.
  - Each patch is then flattened into a 1D vector and linearly projected to embed spatial information.
- Utilizing Attention Mechanism:
  - These small patches are passed through an attention mechanism to learn intricate relationships between all the patches.
  - Attention mechanisms allow the model to focus on relevant patches while considering global context.
- Application of Attention-Based CNN and Vision Transformer:
  - Attention-based CNNs, such as the popular models like ResNet and DenseNet, have shown remarkable performance in various computer vision tasks.
  - Vision Transformer (ViT) extends this idea by applying attention mechanisms directly to image patches, demonstrating competitive performance in image classification, object detection, and other tasks.

Vision Transformer Sketch

- Graphically, we can portray the concept using the following sketches:

$$X_{\text{as image}} \rightarrow \begin{matrix} \text{patch}_1 & \text{patch}_2 \\ \text{patch}_3 & \text{patch}_4 \end{matrix} \rightarrow \text{Attention}[\text{softmax}(\cdot)] \rightarrow \mathbb{P}(\text{patch}_j) \rightarrow \hat{y}$$

- Denote X as the original image. We first partition X into small patches. For simplicity, we suppose there are 4 patches. We send the patches into attention mechanism with multiple softmax architecture inside. Then we are able to create the probability distribution of these patches. Last, we map these probabilities into the prediction $\hat{y}$.

Introduction

- Prior to the 1990s, NLP primarily relied on hand-crafted rules and heuristic-based systems, deeply entrenched in the symbolic approach.
- The exploration and development of NLP systems revolved around crafting explicit grammatical and syntactical rules, which proved to be labor-intensive and inherently limited.

Introduction (Cont'd)

- The exploration and development of NLP systems revolved around crafting explicit grammatical and syntactical rules, which proved to be labor-intensive and inherently limited.

Early Literature in Natural Language Processing

- The advent of Natural Language Processing (NLP) was underpinned by the work of numerous dedicated researchers who navigated through the complexities of linguistics and computational models.
- One pivotal figure in the development of NLP was Noam Chomsky, a linguist whose theories laid a significant groundwork for understanding syntactical structures. Chomsky's introduction of the concept of transformational-generative grammar in the 1950s, elucidated in his groundbreaking work "Syntactic Structures" (1957), indirectly influenced computational approaches to language [3].
- In the computational realm, researchers such as Terry Winograd brought considerable attention to the intricacies and challenges of developing computer programs that could understand and generate human language. Winograd's SHRDLU, developed in the late 1960s and early 1970s, demonstrated an early capacity for a machine to process and respond to natural language inputs, albeit in a highly restricted domain.
- Moreover, the development of the ELIZA program by Joseph Weizenbaum in the mid-1960s marked an emblematic moment in NLP. Although ELIZA simulated conversational abilities through relatively simplistic pattern-matching and substitution methodologies, the program ignited debates and explorations regarding machine understanding of language.

Early Literature in Natural Language Processing (Cont'd)

- During the 1980s, the limitations of hand-crafted, rule-based approaches in handling the variability and ambiguity of natural language became increasingly evident. The field began to witness a shift towards data-driven methods and machine learning, with researchers such as Fred Jelinek championing this transition.
- In this context, the work of these researchers, among others, not only brought the multifaceted challenges of NLP to the forefront but also sowed the seeds for the evolution of the field, paving the way from rule-based methodologies towards the data-driven, machine-learning approaches that would come to define NLP in the latter part of the 20th century and beyond. Their seminal papers provided foundational perspectives, theoretical frameworks, and practical insights that have continued to inform and shape the trajectory of research and development in NLP.

Literature in Natural Language Processing Pre-2000s

- Between 1990 and 2000, the challenges of Natural Language Processing (NLP) were not entirely solved, but the decade saw significant advancements and a pivotal shift in methodologies that addressed some of the limitations of previous rule-based approaches.
- The key transformation during this period was the increasing adoption of statistical and data-driven methods, heavily influenced by the emergence of machine learning technologies, which provided new avenues to explore and develop NLP applications.
- Fred Jelinek and his team at IBM were pivotal figures during this time. Jelinek's work on statistical language models for speech recognition showcased the potential of utilizing large corpora and statistical methods to manage language complexities (Jelinek F. M., 1991) [7].

Literature in Natural Language Processing Pre-2000s (contd.)

- The late 1980s and early 1990s marked the emergence and popularization of machine learning models in NLP.
- Decision trees, Hidden Markov Models, and probabilistic context-free grammar became fundamental models for various language processing tasks (Kotsiantis, 2013) [8]
- Additionally, during the 1990s, the increasing availability and use of larger textual corpora enabled the development and refinement of statistical methods, providing richer data upon which to train and evaluate models (Sharoff, 2013) [5].
- The introduction of the Maximum Entropy (MaxEnt) model in the late 1990s marked a shift towards models that could handle multiple overlapping dependencies, accommodating a higher level of linguistic complexity and contextuality (Chen, 2003) [2].

Literature in Natural Language Processing Pre-2000s (contd.)

- Furthermore, the rise of the internet during this period and the subsequent proliferation of online text provided an expansive and continually growing source of data for training and developing NLP models.
- The data available from web pages, online forums, and later, social media platforms, supplied a rich and diverse linguistic dataset that enabled the development of more robust and versatile NLP applications.

Literature that Gave Birth to the Attention Mechanism

- The attention mechanism was introduced in the paper "Neural Machine Translation by Jointly Learning to Align and Translate" by Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, published in 2014 [1].
- It enhances the performance of Neural Machine Translation (NMT) systems by enabling the model to focus on different parts of the input sequence when producing the output.
- This concept mimics human attention, which selectively focuses on different parts of the input when understanding language or performing tasks.
- The attention mechanism builds upon advancements in machine learning and NLP from the period of 1990-2000.

Literature that Gave Birth to the Attention Mechanism (Cont'd)

- During the 1990s, NLP transitioned towards data-driven approaches, with statistical models and machine learning algorithms playing a pivotal role.
- Advancements in computational power enabled the development of models leveraging large-scale textual data for NLP tasks.
- In the early 2000s, neural networks, including Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, became prominent in NLP.
- Encoder-Decoder architectures, utilizing RNNs or LSTMs, emerged for sequence-to-sequence tasks like machine translation.

Literature that Gave Birth to the Attention Mechanism (Cont'd)

- However, these models faced limitations due to information loss and challenges in handling long sequences.
- The introduction of the attention mechanism in 2014 addressed these limitations by enabling dynamic "attention" to different parts of input sequences.
- The Transformer model, introduced in 2017, relied wholly on attention mechanisms, defining the state of the art in NLP tasks.
- Models like BERT, GPT, and their successors have since demonstrated the profound impact of the attention mechanism on the field of NLP.

Contemporary Literature of Natural Language Processing

- The evolution of natural language processing models, especially within deep learning and neural networks, from the inception of the attention layer to the development and deployment of models like GPT (Generative Pre-trained Transformer) has been marked by various milestones (Lajkó, 2022) [14].
- The attention mechanism, introduced by Bahdanau, Cho, and Bengio in 2014, provided a way for models to selectively focus on different parts of an input sequence, significantly improving the performance of sequence-to-sequence models.
- A significant milestone after the attention mechanism's introduction was the release of the paper "Attention is All You Need" by Vaswani et al in 2017, which introduced the Transformer model. This architecture utilized self-attention mechanisms and completely did away with recurrence in the model architecture, enabling more scalable and parallel processing of inputs (Vaswani A. S., 2017) [16].

Introduction

- An attention mechanism in a neural network can be represented mathematically as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where:

- Q, K, and V are the query, key, and value matrices, respectively.
- d_k is the dimensionality of the key vectors.
- The softmax function is applied row-wise to normalize the attention scores.

Attention Mechanism: Mathematical Form and Example

- The attention mechanism in neural networks is a way to focus on specific parts of input data, assigning different weights to different elements.
- Mathematically, given a query Q, a set of keys K, and a set of values V, the attention mechanism computes a weighted sum of the values, where the weights are determined by the compatibility between the query and the keys.
- Let's consider a simple example with a small matrix:

$$Q = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

$$K = \begin{pmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

$$V = \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{pmatrix}$$

- Let's walk through the computation:
  1. Compute the compatibility scores using dot product: score $= Q \cdot K^T$
  2. Apply softmax to obtain attention weights: weights $=$ softmax(score)
  3. Compute the weighted sum of values: output $=$ weights $\cdot V$

Attention Mechanism: Mathematical Computation (Cont'd)

- Continuing from the previous slide:
  1. Compute the compatibility scores using dot product: $\text{score} = Q \cdot K^T$
  2. Apply softmax to obtain attention weights: $\text{weights} = \text{softmax}(\text{score})$
  3. Compute the weighted sum of values: $\text{output} = \text{weights} \cdot V$
- Let's perform the computations:
  1. Dot product:
  $$\text{score} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 7 & 10 \\ 5 & 8 & 11 \\ 6 & 9 & 12 \end{pmatrix}^T$$
  2. Softmax:
  $$\text{weights} = \text{softmax}(\text{score})$$
  3. Weighted sum:
  $$\text{output} = \text{weights} \cdot V$$

Attention Mechanism: Mathematical Computation (Cont'd)

- Continuing from the previous slide:
  1. Compute the compatibility scores using dot product: score $= Q \cdot K^T$
  2. Apply softmax to obtain attention weights: weights $=$ softmax(score)
  3. Compute the weighted sum of values: output $=$ weights $\cdot V$
- Let's perform the computations:
  1. Dot product:

$$\text{score} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 7 & 10 \\ 5 & 8 & 11 \\ 6 & 9 & 12 \end{pmatrix}^T$$

$$= \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

$$= \begin{pmatrix} 4+14+30 & 5+16+33 & 6+18+36 \end{pmatrix}$$

$$= \begin{pmatrix} 48 & 54 & 60 \end{pmatrix}$$

  2. Softmax:

$$\text{weights} = \text{softmax}(\text{score}) = \text{softmax}(\begin{pmatrix} 48 & 54 & 60 \end{pmatrix})$$

  3. Weighted sum:

$$\text{output} = \text{weights} \cdot V$$

Attention Mechanism: Mathematical Computation (Cont'd)

- Continuing from the previous slide:
  1. Dot product:

$$\text{score} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 7 & 10 \\ 5 & 8 & 11 \\ 6 & 9 & 12 \end{pmatrix}^T$$

$$= \begin{pmatrix} 4+14+30 & 5+16+33 & 6+18+36 \end{pmatrix}$$

$$= \begin{pmatrix} 48 & 54 & 60 \end{pmatrix}$$

  2. Softmax:

$$\text{weights} = \text{softmax}(\text{score})$$

$$= \text{softmax}(\begin{pmatrix} 48 & 54 & 60 \end{pmatrix})$$

  3. Weighted sum:

$$\text{output} = \text{weights} \cdot V$$

- Let's perform the softmax computation:

$$\text{softmax}(\begin{pmatrix} 48 & 54 & 60 \end{pmatrix}) = \begin{pmatrix} \frac{e^{48}}{e^{48}+e^{54}+e^{60}} & \frac{e^{54}}{e^{48}+e^{54}+e^{60}} & \frac{e^{60}}{e^{48}+e^{54}+e^{60}} \end{pmatrix}$$

- Finally, let's compute the weighted sum:

$$\text{output} = \text{weights} \cdot V = \begin{pmatrix} \text{softmax}(48) & \text{softmax}(54) & \text{softmax}(60) \end{pmatrix} \cdot \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{pmatrix}$$

$$= \text{omitted}$$

Contemporary Literature of Natural Language Processing (Cont'd)

- Another key development was the introduction of BERT (Bidirectional Encoder Representations from Transformers) by researchers at Google in 2018. BERT exploited the Transformer architecture for pre-training a model on a large corpus of text and demonstrated remarkable performance on various tasks (Devlin, 2018) [4].
- OpenAI introduced GPT (Generative Pretrained Transformer) in 2022, leveraging the Transformer architecture with a different pre-training approach. GPT was trained as a language model to predict the next word in a sentence and showcased the potency of transfer learning in NLP (Lajkó, 2022) citelajko2022fine.
- Building upon the success of GPT, OpenAI introduced GPT-2 in 2019, significantly larger and trained on a more diverse corpus of text. GPT-2 generated considerable attention due to its capability to generate coherent and contextually relevant text (Lajkó, 2022) [9].

Contemporary Literature of Natural Language Processing (Cont'd)

- GPT-3, introduced in 2020, further scaled up the architecture and training data, resulting in a model with 175 billion parameters. It demonstrated the capability to perform specific NLP tasks without task-specific training data, using a few-shot learning approach (Brown, 2020).
- In summary, the journey from the attention layer to GPT-3 showcased rapid advancements in NLP, progressing from enhancing sequence-to-sequence models to developing massively scaled, pre-trained models capable of various NLP tasks with little to no task-specific training. This trajectory highlights the growing capability and potential of language models.

Challenges of Modern Tasks in NLP (Part 1)

- Developing and deploying large-scale language models like BERT and GPT-3 entails numerous challenges and requires substantial resources.
- Training state-of-the-art language models requires immense computational resources, involving powerful GPUs and TPUs. The training process can take weeks or even months, entailing substantial electricity consumption and resulting in significant carbon emissions.
- These models require extensive and diverse training data to develop a broad understanding of language.
- Ensuring the data is representative, unbiased, and of high quality is a non-trivial task, necessitating sophisticated data collection, cleaning, and preprocessing pipelines.

Challenges of Modern Tasks in NLP (Part 2)

- Language models tend to inherit the biases present in their training data, which can perpetuate harmful stereotypes and produce outputs that are prejudiced or offensive.
- The potential misuse of large language models for malicious purposes, such as generating disinformation or abusive content, is a significant concern.
- Understanding why large language models produce specific outputs and deciphering their internal workings is challenging due to their size and complexity, impeding efforts to debug, refine, and trust these models.
- While these models exhibit remarkable generalization in numerous scenarios, they can sometimes produce nonsensical or inappropriate outputs, especially in cases that deviate from their training data or involve nuanced understanding.

Convert Sentence to Numbers

- Sentence: "I love learning machine learning"
- Word Dictionary: {i, learning, love, machine}
- Word Dictionary Index: {0, 1, 2, 3} (ordered alphabetically)
- Binary Representation:
  - "I" or "i": 1000
  - "learning": 0100
  - "love": 0010
  - "machine": 0001
- Matrix Representation of the phrase "i love learning machine learning" is:

$$\begin{pmatrix} "i" & "love" & "leanring" & "machine" & "learning" \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

- Optional: 1D Vector (Word Index): [0, 2, 1, 3, 1]

Compute Similarity between Sentences

- Sentences: "hello world" and "hi world"
- Unique Words: {hello, hi, world}
- Dictionary Index: {0, 1, 2} (ordered alphabetically)
- Vector Representation:
  - "hello world": 101 or in vector form we write $< 1, 0, 1 >$
  - "hi world": 011 or in vector form we write $< 0, 1, 1 >$
- Cosine Similarity:

$$\text{similarity}(\mathbf{v}_1, \mathbf{v}_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|}$$

$$\text{similarity}(\mathbf{v}_1, \mathbf{v}_2) = \frac{1 \times 0 + 0 \times 1 + 1 \times 1}{\sqrt{2} \times \sqrt{2}} = \frac{1}{2} = 0.5$$

- In words, the cosine similarity is calculated as the ratio of dot product of the two vectors over the product of the magnitudes of the two vectors. This final numerical value 0.5 kind of makes sense becase "hello world" and "hi world" have two words in each sentences and they are one word different.

Technical Requirements (Part 1)

- Access to high-performance computing clusters with multiple GPUs or TPUs is pivotal.
- These models entail billions of parameters and require considerable memory and computational power for training and inference.
- Efficient and scalable machine learning libraries and frameworks, such as TensorFlow or PyTorch, are crucial for model development.

Technical Requirements (Part 2)

- Adeptness in handling distributed computing to manage the training of large-scale models across multiple GPUs or nodes is essential.
- Sizable and diverse text corpora are requisite for training language models.
- Developing these models requires expertise in machine learning, natural language processing, and deep learning.

Technical Requirements (Part 3)

- Understanding the intricacies of model architecture, training dynamics, and task-specific considerations is pivotal.
- Financial resources to access or acquire the requisite hardware (either through purchase or cloud computing rental) are crucial.
- Ensuring the responsible and ethical development and deployment of language models necessitates oversight from individuals or committees versed in the ethical, social, and legal implications of AI technologies.

Tokenizers in Natural Language Processing

- **Word Tokenizer**
  - Splits text into words, the most common form of tokenization.
  - Useful for preparing text for parsing or text analysis.
  - Sample Python Code: [See notebooks.]
- **Sentence Tokenizer**
  - Breaks text into sentences, useful for processing at the sentence level.
  - Sample Python Code: [See notebooks.]

Tokenizers in Natural Language Processing (Cont'd)

- **Subword Tokenizer**
  - Divides words into smaller parts based on subword occurrence.
  - Useful for handling out-of-vocabulary words in languages like English.
  - Sample Python Code: [See notebooks.]
- **Character Tokenizer**
  - Separates text into individual characters, useful for character-level tasks.
  - Sample Python Code: [See notebooks.]

Tokenizers in Natural Language Processing (Cont'd)

- **Regexp Tokenizer**
  - Uses regular expressions to control the tokenization of text.
  - Highly customizable.
  - Sample Python Code: [See notebooks.]
- **Space Tokenizer**
  - Tokenizes based on spaces, a simple tokenizer.
  - May not work well for texts with punctuation.
  - Sample Python Code: [See notebooks.]

Tokenizers in Natural Language Processing (Cont'd)

- **Treebank Word Tokenizer**
  - A tokenizer that uses regular expressions to tokenize text as in Penn Treebank.
  - A sophisticated tokenizer dealing with complex word tokenization rules.
  - Sample Python Code: [See notebooks.]
- **Installation Note**
  - When using these tokenizers, especially those from external libraries like NLTK or Hugging Face's tokenizers, you may need to install the libraries first using pip if you haven't already.
  - Specific tokenizers like the ByteLevelBPETokenizer require pre-training on a dataset before use.

Future of Tokenizers

- Multimodality:
  - Tokenizers may evolve to handle multiple modes of input more seamlessly, including text, voice, images, and potentially other data types.
  - This could lead to tokenizers that are not solely focused on language but also on other forms of data representation.
- Efficiency:
  - There is a push for more efficient tokenization methods that can handle longer sequences of text with less computational overhead.
  - Byte Pair Encoding (BPE) and its variants have been popular, but new methods may emerge that provide better trade-offs between efficiency and performance.
- Character-Level Tokenization:
  - Character-level tokenization can be more parameter-efficient and could handle out-of-vocabulary words better.
  - It's possible that future tokenizers will use a hybrid approach that combines the best aspects of word-level and character-level tokenization.

Future of Tokenizers (Cont'd)

- Contextual Tokenization:
  - Tokenizers might become more context-aware, adjusting the tokenization process based on the surrounding text.
  - This could improve the handling of homographs and other language nuances.
- Language Agnosticism:
  - Advances may be made towards creating tokenizers that are truly language-agnostic, being able to handle any language with little to no performance loss.
  - This could eliminate the need for language-specific training.
- Unsupervised and Semi-supervised Learning:
  - There might be a shift towards unsupervised and semi-supervised methods that require less annotated data for training.
  - This could be particularly beneficial for under-resourced languages.

Introduction

- **Embedding Layers in NLP**:
  - Word embeddings are crucial in NLP, converting tokens into continuous vectors for machine learning models.
  - Various famous embedding layers include Word2Vec, GloVe, FastText, BERT, ELMo, PaLM, OpenAI Embedding, and AWS's Bedrock Embedding.

Introduction (continued)

- **Sample Python Code**:
  - Each embedding layer comes with sample Python code for implementation.
  - These codes demonstrate how to use each embedding layer in Python for NLP tasks. [See notebooks.]

Future Potential of Natural Language Processing

- Future potential of Natural Language Processing (NLP) extends far beyond current capabilities:
  - Reflects an amalgamation of technical advancement, refined linguistic understanding, and enhanced user experience in human-computer interaction.
  - Models like ChatGPT have considerably advanced our capacity to generate human-like text and comprehend language.
- One poignant direction in the future of NLP is toward achieving genuine understanding and generation of language:
  - Advancing models to a point where they can comprehend context, nuance, and semantics in a manner analogous to humans poses a considerable frontier.
  - Involves not just managing syntactical and lexical aspects but truly grasping the underlying meaning, intent, and emotion conveyed through language.

Future Potential of Natural Language Processing (cont'd)

- Enhancing the ethical, unbiased, and fair operation of NLP systems is paramount:
  - Future models will necessitate sophisticated mechanisms to identify, understand, and neutralize biases.
  - Ensuring that these models can comprehend and communicate in a plethora of languages and dialects is crucial to uphold linguistic and cultural diversity.
- Future could see an escalation in the confluence of NLP with other AI domains:
  - Enabling NLP systems to comprehend and generate language in conjunction with understanding visual, auditory, or sensory inputs could revolutionize human-computer interaction.
  - This could impact various fields, such as developing virtual reality environments where users can communicate naturally with entities.

Future Potential of Natural Language Processing (cont'd)

- Ensuring the responsible and ethical advancement of NLP is pivotal:
  - Future NLP technologies must embed ethical considerations, privacy, and security in their design and operation.
  - Developing mechanisms to mitigate the misuse of NLP technologies will be crucial to protect societal and individual well-being.

Introduction to Vector Stores and Databases

- Vector stores or databases are specialized storage systems designed for efficient handling of vector data.
- Vector data is commonly used in similarity search or nearest neighbor search in machine learning and NLP tasks.
- Here are some well-known vector databases and libraries:
  - Annoy (Approximate Nearest Neighbors Oh Yeah)
  - Elasticsearch with the ‘elastiknn‘ plugin
  - Milvus
  - ScaNN (Scalable Nearest Neighbors)
  - Hnswlib

Introduction to Vector Stores and Databases (Cont'd)

- Annoy (Approximate Nearest Neighbors Oh Yeah): Annoy is a C++ library with Python bindings for Approximate Nearest Neighbors (ANN) search. It is designed to efficiently find the nearest neighbors of a given query point in high-dimensional spaces. Annoy was developed by Spotify.
- Elasticsearch with the 'elastiknn' plugin: Elasticsearch is a distributed, RESTful search and analytics engine built on top of Apache Lucene. The 'elastiknn' plugin extends Elasticsearch to support similarity search and nearest neighbor queries. It is developed by Open Distro for Elasticsearch, an open-source distribution of Elasticsearch.
- Milvus: Milvus is an open-source vector database designed for storing and processing high-dimensional vectors, such as those used in machine learning and deep learning applications. It provides efficient indexing and similarity search capabilities. Milvus is developed by Zilliz, an AI technology company.
- ScaNN (Scalable Nearest Neighbors): ScaNN is a library for efficient approximate nearest neighbor search in high-dimensional spaces. It is designed to be scalable and suitable for large-scale applications. ScaNN is developed by Google.
- Hnswlib: Hnswlib is a C++ library for approximate nearest neighbor search based on the Hierarchical Navigable Small World (HNSW) algorithm. It provides efficient indexing and search capabilities for high-dimensional data. Hnswlib is developed by Yandex.

Introduction to Vector Stores and Databases (Cont'd)

- NGT (Neighborhood Graph and Tree for Indexing High-dimensional Data): Developed by Yahoo! Japan Research, NGT is a high-speed approximate nearest neighbor search library designed to efficiently handle high-dimensional data. It provides a scalable and memory-efficient solution for nearest neighbor search tasks in large datasets, making it suitable for various applications such as recommendation systems and data clustering.
- TileDB: Developed by TileDB Inc., TileDB is a universal storage engine that efficiently manages multidimensional array data. It offers a versatile and scalable solution for storing and querying diverse data types, including genomic data, geospatial data, and sensor data. TileDB's flexible architecture and cloud-native design make it suitable for a wide range of use cases across industries.
- Weaviate: Developed by SeMI Technologies, Weaviate is an open-source vector search engine that specializes in handling high-dimensional data and performing semantic searches. It enables developers to build intelligent applications capable of understanding natural language queries and retrieving relevant information from large datasets. Weaviate's graph-based approach and schema-less design make it well-suited for applications in AI, knowledge graphs, and recommendation systems.

Introduction to Vector Stores and Databases (Cont'd)

- Pinecone: Developed by Pinecone Systems Inc., Pinecone is a cloud-native vector database optimized for similarity search tasks. It provides fast and accurate nearest neighbor search capabilities for high-dimensional vector data, such as embeddings generated by deep learning models. Pinecone's managed service and auto-scaling features make it easy to deploy and scale similarity search applications in production environments.
- Vald: Developed by Platformer Inc., Vald (Vector-Advanced Library for Deep learning) is an open-source distributed vector similarity search engine designed to handle large-scale high-dimensional data. It offers efficient indexing and retrieval of vector embeddings, making it suitable for applications such as image search, recommendation systems, and anomaly detection. Vald's distributed architecture and GPU acceleration support enable fast and scalable similarity search operations.

Considerations for Choosing a Vector Store or Database

- These systems excel in specific scenarios based on:
  - Data scales
  - Latency requirements
  - Integration with existing machine learning workflows
- Choosing the right system depends on factors such as:
  - Size of the dataset
  - Dimensionality of the vectors
  - Required precision and recall
  - Latency constraints
  - Scalability needs

Introduction to Fine-tuning Large Language Models

- Large Language Models (LLMs) have revolutionized natural language processing, but fine-tuning them for specific tasks is challenging due to high computational costs and storage requirements.
- Parameter-Efficient Fine-Tuning (PEFT) techniques have been developed to address this challenge by achieving high task performance with fewer trainable parameters.
- PEFT methods balance computational efficiency and task performance, enabling fine-tuning of even the largest LLMs without compromising quality.

Pretrained LLMs and Fine-tuning

- Pretrained LLMs are trained on vast general-domain data, allowing them to capture rich linguistic patterns and knowledge.
- Fine-tuning involves adapting pretrained models to specific tasks by training them on task-specific datasets, adjusting parameters to optimize performance.
- Fine-tuning enables leveraging the knowledge of pretrained models for specialized tasks, enhancing task performance.

Parameter Efficient Fine-tuning (PEFT)

- PEFT methods reduce the number of trainable parameters while maintaining computational efficiency and task performance.
- PEFT offers practical benefits such as reduced memory usage, storage cost, and inference latency.
- However, PEFT may require additional training time compared to traditional methods and could be sensitive to hyperparameter choices.

Introduction

- Types of PEFT:
  - Various PEFT methods have been developed to cater to different requirements and trade-offs.
  - Some notable PEFT techniques include T-Few, which attains higher accuracy with lower computational cost, and AdaMix. This general method tunes a mixture of adaptation modules for better performance across different tasks.
  - T-Few: This method uses $(IA)^3$, a new PEFT approach that rescales inner activations with learned vectors. It achieves super-human performance and uses significantly fewer FLOPs during inference than traditional fine-tuning.

Introduction (contd.)

- AdaMix:
  - A general PEFT method that tunes a mixture of adaptation modules, like Houlsby or LoRA, to improve downstream task performance for fully supervised and few-shot tasks.
- MEFT:
  - A memory-efficient fine-tuning approach that makes LLMs reversible, avoiding caching intermediate activations during training and significantly reducing memory footprint.
- QLORA:
  - An efficient fine-tuning technique that uses low-rank adapters injected into each layer of the LLM, greatly reducing the number of trainable parameters and GPU memory requirement.

Introduction to Early AI Milestones

- Turing introduced the concept of a machine exhibiting human-like intelligence through the Turing Test (Tavani, 1950).
- Rosenblatt's perceptron, a foundational model for machine learning and neural networks, emerged in 1958 (Rosenblatt, 1958).
- The General Problem Solver (GPS), developed by Newell in 1961, aimed to demonstrate human problem-solving capabilities in a computer.

Introduction to Early AI Milestones (contd.)

- Weizenbaum's ELIZA: An early chatbot developed by Joseph Weizenbaum in 1966 at the MIT Artificial Intelligence Laboratory. It emulated a Rogerian psychotherapist, engaging users in conversation by responding with scripted patterns based on keywords.
- Shortliffe's MYCIN: A rule-based expert system for medical diagnosis developed by Edward Shortliffe in the early 1970s at Stanford University. It was one of the earliest applications of artificial intelligence in healthcare, using a set of rules to recommend treatments based on patient data and medical knowledge.
- McCarthy's challenge: John McCarthy, one of the pioneers of artificial intelligence, highlighted the challenge of representing changes in the world within AI and symbolic logic in his paper "Epistemological Problems of Artificial Intelligence" published in 1981. He emphasized the difficulty of capturing dynamic aspects of the real world in symbolic systems.
- The A* algorithm: A search algorithm introduced by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968 at Stanford Research Institute. It is foundational for reinforcement learning and widely used in pathfinding and graph traversal. The A* algorithm efficiently finds the optimal path between nodes in a graph by using heuristics to guide the search.

Introduction

- Reinforcement Learning (RL) has its roots in the study of optimal control and decision making, which dates back to the early 20th century.
- Among the list provided, the General Problem Solver (GPS) by Allen Newell and Herbert A. Simon is the closest in spirit to reinforcement learning.
- While GPS itself is not a reinforcement learning algorithm, the desire for more general approaches to learning and problem-solving set the stage for the development of reinforcement learning.

Introduction (cont'd)

- Q-learning is an off-policy learner, meaning it learns the value of the optimal policy irrespective of the actions taken, using a process called bootstrapping.
- Over time, Q-learning has been foundational to the development of many other RL algorithms and has been applied to numerous real-world problems.
- One of the most famous applications of Q-learning is in teaching agents to play games, especially the suite of games provided by the Atari 2600 console.

Introduction (cont'd)

- Neural networks, particularly deep neural networks, provide a powerful function approximation tool that can handle high-dimensional inputs and represent complex functions.
- Consider the example of the Atari games. The input is the raw pixel data of the game screen.
- Neural networks allow for end-to-end learning directly from raw inputs (like images or sounds) to outputs (like actions).

Introduction (cont'd)

- The integration of neural networks with reinforcement learning, particularly Q-learning, was pioneered by several researchers.
- Another notable figure in the realm of reinforcement learning is Richard Sutton, who, along with Andrew Barto, wrote the foundational book "Reinforcement Learning: An Introduction."
- While DeepMind popularized the combination of deep learning and reinforcement learning with their DQN algorithm, the idea of combining neural networks with reinforcement learning dates back to at least the 1990s.

Introduction

- From 2000s and Onward
  - Post-1990s and especially from the 2000s onward, reinforcement learning (RL) experienced significant growth, both in terms of theoretical advancements and practical applications. Several researchers contributed to this progress, and numerous models and algorithms were proposed.
- Prominent Figures
  - Richard Sutton is one of the foundational figures in Reinforcement Learning (RL). He co-authored the widely used textbook "Reinforcement Learning: An Introduction" alongside Andrew Barto and has made substantial contributions across various RL topics. David Silver is another prominent figure, known for his pioneering work at DeepMind. He played a crucial role in fusing deep learning with RL, leading to the development of groundbreaking models like DQN, AlphaGo, and their subsequent versions. Pieter Abbeel has gained recognition for his significant contributions to robotics and deep reinforcement learning. His work has spanned areas such as deep imitation learning and meta-learning. Sergey Levine is a central figure at the confluence of deep learning, robotics, and RL. He has been instrumental in the development of algorithms like the soft actor-critic and has significantly impacted real-world robotic applications. John Schulman is renowned for his work at OpenAI. His expertise lies in policy gradient methods, and he has made notable contributions to algorithms like Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO).

Introduction (Cont'd)

- Key Models and Algorithms
  - In terms of models and algorithms, the DQN, or Deep Q Networks, stands out for integrating deep learning with Q-learning, enabling the handling of high-dimensional inputs such as raw pixels from video games. TRPO, or Trust Region Policy Optimization, represents an advanced policy gradient approach that emphasizes stable updates by limiting drastic changes to the policy during each update. PPO, or Proximal Policy Optimization, refines the TRPO concept, achieving comparable performance with a simpler design, making it one of the go-to algorithms for many practical applications. The Soft Actor-Critic algorithm is an off-policy actor-critic method that emphasizes exploration through entropy regularization. DeepMind's contributions to the field are further highlighted by models like AlphaGo and its successors, AlphaZero and MuZero.
- Significance and Applications
  - The significance of these advancements in RL is manifold. Modern RL algorithms, particularly when synergized with deep learning, have showcased the ability to learn and generalize across a diverse spectrum of tasks, often leveraging raw sensory data. Practical applications of RL have been widespread across gaming, robotics, automotive, natural language processing, finance, healthcare, and energy optimization sectors.

Deep Q-Network (DQN)

- Reward Function in DQN:
  - The reward function in the deep Q-network (DQN) is a crucial component that guides the learning process.
  - It assigns a numerical value to the agent's actions in an environment, indicating how favorable or unfavorable they are.
  - Typically denoted as $R(s, a, s')$, where s is the current state, a is the action taken, and s' is the resulting state after taking action a.
  - The goal of the reward function is to incentivize the agent to take actions that lead to desirable outcomes and discourage actions that lead to undesirable outcomes.
- Optimization Strategy:
  - In DQN, the optimization strategy revolves around approximating the optimal action-value function $Q^*(s, a)$.
  - This is achieved by iteratively updating a neural network to minimize the difference between the predicted Q-values and the target Q-values.
  - The target Q-values are calculated based on the Bellman equation, incorporating rewards obtained from transitions between states.
  - The optimization process often employs techniques like stochastic gradient descent (SGD) or its variants to update the neural network parameters.
  - By continually adjusting the network parameters to better approximate the optimal Q-values, the DQN learns to make more informed decisions in the environment.

Introduction

- Proximal Policy Optimization (PPO) is a reinforcement learning algorithm designed to improve upon traditional policy gradient methods.
- PPO operates in the space of policy gradient methods, aiming to combine the efficiency of value-based methods with the robustness of trust region methods.
- The main idea behind PPO is to clip the policy update, preventing drastic changes from the old policy and avoiding instability in training.

Introduction (contd.)

- In the context of language models like ChatGPT, PPO is used for iterative refinement based on human feedback, optimizing the model's parameters to produce responses closer to human preferences.
- PPO is employed after initial supervised training and collection of comparison data, aiming to align the model's outputs more closely with human preferences.
- This iterative process ensures that the model's responses evolve to match human preferences, enhancing its effectiveness in generating high-quality responses.

Proximal Policy Optimization (PPO)

- Reward Function in PPO: PPO is motivated by the same question as TRPO: how can we take the biggest possible improvement step on a policy using the data we currently have, without stepping so far that we accidentally cause performance collapse? Where TRPO tries to solve this problem with a complex second-order method, PPO is a family of first-order methods that use a few other tricks to keep new policies close to old. PPO methods are significantly simpler to implement, and empirically seem to perform at least as well as TRPO.
- **PPO-Penalty** approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient over the course of training so that it's scaled appropriately.
- **PPO-Clip** doesn't have a KL-divergence term in the objective and doesn't have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

Proximal Policy Optimization (PPO) (contd.)

- PPO-clip updates policies via

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k,\theta)]$$

typically taking multiple steps of (usually minibatch) SGD to maximize the objective. Here L is given

$$L(s,a,\theta_k,\theta)\min\Big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), \text{clip}\big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\big)A^{\pi_{\theta_k}(s,a)}\Big)$$

in which ϵ is a (small) hyperparameter which roughly says how far away the new policy is allowed to go from the old.

- Notations:
  - θ_{k+1}: Updated policy parameters.
  - θ_k: Current policy parameters.
  - $\pi_\theta(a|s)$: Probability of taking action a in state s under policy π_θ.
  - $A^{\pi_{\theta_k}}(s,a)$: Advantage function for action a in state s under policy π_{θ_k}.
  - ϵ: Hyperparameter controlling the extent of policy change.

Proximal Policy Optimization (PPO) (contd.)

- The optimization strategy involves maximizing the objective function L using stochastic gradient descent (SGD).
- This is done by taking multiple steps (usually minibatch) to update the policy parameters θ.
- The objective function L incorporates a clipped surrogate objective to ensure stable updates and prevent large policy changes.
- The hyperparameter ϵ controls the magnitude of the clipping, balancing exploration and exploitation in the learning process.

Threats and Challenges of AI in Business and Society

- AI ethics refers to the moral principles and practices guiding the development, deployment, and use of AI technologies.
- It encompasses issues like transparency, fairness, privacy, and the distribution of benefits and harms.
- Ethical considerations include responsibility, accountability, and socio-economic impacts on individuals and communities.
- In organizations, AI's ethical implications revolve around workplace transformation, decision-making, and competitive landscapes.

Threats and Challenges of AI in Business and Society (cont'd)

- Ethically aligned AI can enhance productivity, foster innovation, and contribute to unbiased decision-making.
- However, mismanaged AI may perpetuate biases, reduce decision-making transparency, and lead to job displacement.
- Societal implications of AI ethics are pronounced, impacting areas like healthcare, law enforcement, and education.
- Ethical AI use demands scrutiny to prevent erosion of privacy, autonomy, and social cohesion.
- Balancing technological advancement with human dignity and equitable society is crucial.

Ethical Issues in Artificial Neural Network Models (Part 1)

- Opaqueness, often referred to as the "black box" problem, is a primary ethical concern with Artificial Neural Networks (ANNs) (Carabantes, 2020; Guidotti, 2018; Starke, 2022).
- ANNs' complex and layered structure makes it challenging to understand how they arrive at specific decisions or predictions (Tschider, 2020).
- Lack of transparency can be problematic in high-stakes domains like criminal justice or healthcare, where understanding the decision-making process is crucial.

Ethical Issues in Artificial Neural Network Models (Part 2)

- ANNs have the potential to perpetuate and amplify biases present in their training data (Rad, 2023).
- Biases, inaccuracies, or stereotypes in training data can be learned and perpetuated by ANNs, leading to discrimination and social inequality (Pessach, 2020).
- The inability to discern the impact of specific features on target variables in ANN models raises ethical concerns (Abdullah, 2021).

Ethical Issues in Artificial Neural Network Models (Part 3)

- Understanding which features are most influential in predictions is essential in many applications.
- The "black box" nature of ANNs makes it hard to identify and correct issues, potentially leading to decisions that adversely affect individuals without clear justification or recourse (Lee, 2020).
- Ongoing research into methods for making ANN models more interpretable and fairer is imperative.

Bias-Variance Trade-Off and AI Systems

- The bias-variance trade-off is a fundamental concept in machine learning that deals with finding the right balance between bias and variance when developing predictive models.
- Bias refers to the error introduced by approximating a real-world problem with a simplified model. High bias models tend to oversimplify the underlying patterns in the data and may underfit.
- Variance, on the other hand, measures the model's sensitivity to fluctuations in the training data. High variance models capture noise in the training data and may overfit, performing well on training data but poorly on unseen data.
- The trade-off arises because reducing bias often increases variance and vice versa. Finding the optimal trade-off is crucial for developing models that generalize well to unseen data.
- Understanding the bias-variance trade-off is essential for AI systems because it helps developers make informed decisions about model complexity, feature selection, and dataset size.
- In the context of AI, biases can manifest in various forms, including dataset biases, algorithmic biases, and societal biases encoded in training data. By understanding the bias-variance trade-off, developers can mitigate the impact of biases and build more fair and robust AI systems.

Bias-Variance Trade-off Derivation

- When we assess the performance of a machine learning model, one common metric we use is the Mean Square Error (MSE), which quantifies the average squared difference between the predicted values and the actual values:

$$\text{MSE} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

- To understand the trade-off between bias and variance, let's expand the full square for bias in the MSE equation:

$$\begin{aligned}\text{MSE} &= (\text{Bias})^2 + \text{Variance} + \text{Irreducible Error} \\ &= \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \text{Irreducible Error}\end{aligned}$$

- Here, $\hat{f}(x)$ represents the estimated function, and $f(x)$ is the true function. Expanding the full square allows us to decompose the MSE into components related to bias, variance, and irreducible error.

Ethical Issues in Computer Vision (Part 1)

- Computer vision, powered by Convolutional Neural Networks (CNNs), has enabled unprecedented advancements in how machines process and analyze images.
- However, this capability comes with significant ethical considerations. The accuracy and efficiency of CNNs in image recognition and classification tasks can lead to privacy violations, as surveillance systems become capable of identifying individuals without consent.
- The proliferation of surveillance could erode privacy rights and heighten the risks of mass data collection, potentially leading to misuse by both state and non-state actors.
- Deepfakes, a byproduct of advanced computer vision techniques, pose another profound ethical challenge. They are hyper-realistic video forgeries where a person's likeness is replaced with someone else's, created using CNNs.

Ethical Issues in Computer Vision (Part 2)

- Deepfakes can be used to create convincing false narratives, manipulate public opinion, and discredit individuals, raising serious concerns about truth, consent, and the propagation of misinformation.
- Faceswaps, a less complex cousin of deepfakes, also have ethical implications. While often used for entertainment, swapping one person's face for another in a photo or video can lead to embarrassment, bullying, or worse when shared online.
- The ease with which face swaps can be created and disseminated makes it possible for individuals to be placed in compromising or harmful situations without their agreement, contributing to a culture where the authenticity of visual content is increasingly doubted.
- The ethical issues in computer vision are not confined to privacy and misinformation but extend to the potential biases encoded within CNNs.

Ethical Issues in Computer Vision (Part 3)

- These biases can manifest in the form of racial, gender, or socioeconomic discrimination, particularly if the training data for these networks is not diverse and representative.
- The decisions made by automated systems based on biased computer vision algorithms can lead to unfair treatment or marginalization of certain groups, exacerbating social inequalities.
- As the application of computer vision grows, ensuring that these systems are developed and employed ethically is a societal imperative that requires collaborative efforts from technologists, ethicists, and policymakers.

Ethical Issues in Large Language Models and Generative AI (Part 1)

- Large language models and generative AI raise ethical questions around the originality and attribution of the content they produce.
- These models, trained on vast swathes of data sourced from the internet, generate text that can closely mimic human writing.
- This capability brings up concerns about the erosion of original work and challenges in discerning whether content is created by a human or an AI.
- The difficulty in tracing the origins of AI-generated solutions can lead to issues of intellectual property rights and the potential devaluation of human creativity.
- Moreover, as these models remix existing content, it raises the question of whether the output can be truly considered 'original'.

Ethical Issues in Large Language Models and Generative AI (Part 2)

- The use of generative AI also introduces ethical considerations in terms of accountability and decision-making.
- When AI is used to draft legal documents, write news articles, or develop code, it's often unclear who is responsible for the final output.
- The impersonal nature of these models obscures human accountability, potentially allowing individuals and organizations to abdicate responsibility for AI-generated content.
- Furthermore, large language models and generative AI can amplify and perpetuate biases present in their training data.
- Addressing these biases requires careful curation of training datasets and ongoing monitoring of AI outputs.

Application Investigation: Loan Recommendations

- Fair Treatment by AI
  - Ensuring fair treatment across different social groups and individuals is critical when using AI for loan approval recommendations.
  - Fairness is often addressed by algorithms designed to evaluate applications based on merit and creditworthiness without regard to race, gender, or other protected characteristics.
  - Defining the boundaries of a group and identifying disadvantaged groups requires a nuanced understanding of historical and socio-economic factors.
  - AI can only make these distinctions if it's been appropriately trained on representative datasets, but this approach is fraught with challenges around privacy and the risk of reinforcing stereotypes.

Application Investigation: Loan Recommendations (cont'd)

- Ensuring Inclusivity and Respect for Diversity
  - Developers can ensure their AI systems are inclusive and respectful of diversity by engaging diverse and interdisciplinary teams.
  - Techniques such as fairness audits and rigorous testing against diverse datasets before deployment can help uncover biases.
  - Ongoing monitoring post-deployment is essential to ensure that the AI continues to act fairly as it encounters new data.
- Addressing Biases in AI Models
  - Curating training datasets meticulously and including various data sources can help eliminate biases.
  - Algorithmic fairness approaches can be implemented to adjust the model to account for known biases.
  - Transparency in AI processes, including explainability of neural networks, is vital for model accountability.

Explainability and Diversity in AI-based Loan Approvals

- The explainability issue in neural networks is particularly pertinent in loan recommendations.
  - Opacity of AI models undermines trust and makes it difficult to ensure fair decisions, especially regarding diversity and fairness.
  - An explainable AI model is needed to dissect and understand how and why certain decisions are made, ensuring transparency and justifiability.
  - Developers need to balance the complexity of neural networks with the need for explainable outputs, potentially by simplifying models or using explainability tools.

Introduction

- **Integration of AI Technologies in Business Operations and Decision Making: Example of Intelligent Document Processing (IDP)**
  - Intelligent Document Processing (IDP) is a sophisticated technology that automates the extraction of data from unstructured and semi-structured documents, converting it into a structured and usable form with the help of artificial intelligence (AI).
  - IDP solutions employ a variety of technologies, including Optical Character Recognition (OCR) to transform data from documents into machine-readable formats, Natural Language Processing (NLP) to understand the language and meaning of the data, and machine learning algorithms to classify and validate the data.

Introduction (contd.)

- As a software solution, IDP can capture, transform, and process data from various types of documents such as emails, text files, Word documents, PDFs, or scanned documents, making it a versatile tool for workflow automation.
  - It is capable of handling a wide array of document formats, including papers, PDFs, Word docs, spreadsheets, and more, by scanning, reading, extracting, categorizing, and organizing meaningful information into accessible formats from large data streams.
- The technology underpinning IDP incorporates AI, machine learning (ML), Deep Learning (DL), OCR, and NLP to read and comprehend the context of the information it captures from documents.
  - This facilitates the automation of complex document-processing tasks that would otherwise require manual effort, thereby increasing efficiency and accuracy in data handling.

Introduction

- This paper takes a practical look at Intelligent Document Processing (IDP), focusing on how businesses have historically used it. It fits IDP into the broader context of the current Industrial Revolution, known as Industry 4.0. The discussion centers on the practical difficulties that companies face when they sort through and extract important details from documents like forms and bills to use in their daily operations.
- The paper also looks at the tools available in the market for reading printed text, sorting documents, and pulling out data, along with their costs. This helps in understanding how IDP has been applied in the real world and bridges the gap between theoretical research and actual use.

Introduction (continued)

- To clarify the technical jargon for non-experts, it provides simple definitions of terms commonly found in academic and marketing literature. It highlights the ongoing difficulty of accurately extracting data from tables in invoices, clinical trial protocols, and other medical or insurance related documents—a problem yet to be fully resolved.
- A scenario involving a large industrial firm illustrates the need for flawless information extraction to feed into Enterprise Resource Planning systems, underscoring the high stakes of IDP in business contexts. The discourse extends to avant-garde research conducted by large corporations that are advancing deep learning with substantial computational and financial resources—an endeavor expected to eventually benefit practical applications.

Introduction (conclusion)

- The narrative concludes by questioning the congruence between the commercial sector's objectives and timelines and the advancements in computer science, inviting reflection on the alignment of these two parallel pursuits.

Introduction

- To design an intelligent document processing (IDP) system that is production-ready, we must consider a robust and scalable architecture that incorporates the latest in AI and machine learning technologies, ensuring that each component is designed for high availability and fault tolerance.
- At the core of this system lies the Optical Character Recognition (OCR) engine, a critical component tasked with the conversion of various document types into machine-readable text. This engine is strategically placed after the document ingestion point where raw data from scanned documents, PDFs, and images are received.

Introduction (continued)

- Once ingested, the OCR engine processes these documents, meticulously extracting text while preserving the structure and formatting indicative of the original document. The quality of OCR output is paramount, as it feeds into subsequent stages of the system; hence, it is vital to employ advanced OCR technologies that can handle complex layouts and various fonts and languages with high accuracy.
- Following the OCR process, the extracted text is then piped into a processing module where a large language model is utilized. This language model is designed to handle the intricacies of natural language, extracting meaningful information, entities, and context from the OCR output.

Introduction (continued)

- Given the variability and potential complexity of the text, the language model must be of a sophisticated nature, such as the GPT (Generative Pretrained Transformer) series, capable of understanding context, making inferences, and maintaining a high level of precision in information extraction.
- The system also incorporates a well-defined API layer, which serves as a bridge between the OCR output and the language model. This API layer is responsible for formatting the OCR data into a structure that is readily consumable by the language model, often involving a JSON format that encapsulates the text along with metadata such as document type and source.

Introduction (continued)

- The API then invokes the language model, passing the structured data for processing.
- In terms of the architecture, the system is built on a microservices design principle, ensuring each component, from document ingestion to OCR to the language model API, is decoupled and independently scalable. [See notebooks.]

Introduction (continued)

- This design allows for the seamless integration of additional services, such as data validation, compliance checks, or any other business-specific logic that may require post-information extraction.
- Now, to illustrate the functionality of the large language model within this system, a figure is created to depict the model receiving data from the OCR process via the API. [See notebooks.]

Introduction (continued)

- This figure highlights the flow of data, the transformation and structuring of OCR output, the API call to the language model, and the subsequent return of extracted information back to the system for further processing or storage.
- For the overall system diagram, a comprehensive figure is crafted to detail each step of the IDP system. Starting from document ingestion, it shows the flow through the OCR engine, the API layer, the large language model, and any subsequent processes.
- Each component is connected with arrows to illustrate the direction of data flow and the sequence of operations, ensuring clarity in the system's workings.
- This system, when implemented with high-quality components and rigorous testing, is poised to be production-ready, offering a reliable and efficient solution for intelligent document processing.

Technical Characteristics of the Proposed System

- To design an intelligent document processing (IDP) system that is production-ready, we must consider a robust and scalable architecture that incorporates the latest in AI and machine learning technologies, ensuring that each component is designed for high availability and fault tolerance.
- At the core of this system lies the Optical Character Recognition (OCR) engine, a critical component tasked with the conversion of various document types into machine-readable text.
- This engine is strategically placed after the document ingestion point where raw data from scanned documents, PDFs, and images are received.
- Once ingested, the OCR engine processes these documents, meticulously extracting text while preserving the structure and formatting indicative of the original document.
- The quality of OCR output is paramount, as it feeds into subsequent stages of the system; hence, it is vital to employ advanced OCR technologies that can handle complex layouts and various fonts and languages with high accuracy.
- Following the OCR process, the extracted text is then piped into a processing module where a large language model is utilized.

Technical Characteristics of the Proposed System (Cont'd)

- This language model is designed to handle the intricacies of natural language, extracting meaningful information, entities, and context from the OCR output.
- Given the variability and potential complexity of the text, the language model must be of a sophisticated nature, such as the GPT (Generative Pretrained Transformer) series, capable of understanding context, making inferences, and maintaining a high level of precision in information extraction.
- The system also incorporates a well-defined API layer, which serves as a bridge between the OCR output and the language model.
- This API layer is responsible for formatting the OCR data into a structure that is readily consumable by the language model, often involving a JSON format that encapsulates the text along with metadata such as document type and source.
- The API then invokes the language model, passing the structured data for processing.
- In terms of the architecture, the system is built on a microservices design principle, ensuring each component, from document ingestion to OCR to the language model API, is decoupled and independently scalable.

Illustration of Large Language Model Functionality

- To illustrate the functionality of the large language model within this system, a figure is created to depict the model receiving data from the OCR process via the API.
- This figure highlights the flow of data, the transformation and structuring of OCR output, the API call to the language model, and the subsequent return of extracted information back to the system for further processing or storage.

Overall System Diagram

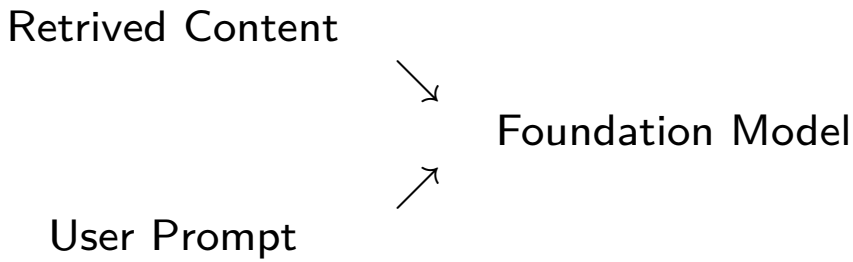
- For the overall system diagram, a comprehensive figure is crafted to detail each step of the IDP system.
- Starting from document ingestion, it shows the flow through the OCR engine, the API layer, the large language model, and any subsequent processes.
- Each component is connected with arrows to illustrate the direction of data flow and the sequence of operations, ensuring clarity in the system's workings.
- This system, when implemented with high-quality components and rigorous testing, is poised to be production-ready, offering a reliable and efficient solution for intelligent document processing.

Standard Architecture of RAG in LLM World

- Retrieval Augmentation Generation (RAG) is a framework that combines retrieval-based methods, augmentation techniques, and generative capabilities within Large Language Models (LLMs) to enhance natural language understanding and generation tasks.
- The standard architecture of RAG typically consists of three main components:
  1. **Retrieval Module**: This module retrieves relevant documents or passages from a large corpus of text based on the input query or context. It leverages efficient indexing and search algorithms to quickly identify potentially useful information.
  2. **Augmentation Module**: The retrieved documents or passages are then passed through an augmentation module. This module enriches the input data by adding contextual information, paraphrasing, or expanding upon the existing content to provide a more comprehensive understanding.
  3. **Generation Module**: Finally, the augmented input is fed into the generation module, which utilizes the capabilities of LLMs to produce coherent and contextually relevant responses or outputs. This module can generate summaries, answers to questions, or even create entirely new text based on the augmented input.
- By integrating these three components, RAG models can effectively leverage both the knowledge encoded in large text corpora and the generative power of LLMs to perform a wide range of natural language understanding and generation tasks with improved accuracy and fluency.

Standard Architecture of RAG in LLM World

- Retrieval Augmentation Generation (RAG) typically takes the following architecture:



- The Retrieved Content are usually a tabular form of references according to certain similarity criteria.
- The User Prompt is the input question that the user entered when interacting with the app.

About the Author

- I have been in the AI/ML space since 2015, leading all forms of AI-backed solutions including but not limited to Computer Vision, Natural Language Models (NLP), and most recently Large Language Models (LLMs) and Generative AI. I am currently a Lead Developer at Vertex Inc, a global leading provider of tax technologies. Previously, I was a Senior ML Engineer at an S&P 500 company, LabCorp, developing AI-driven solutions in drug diagnostics, drug development, operations management, and financial decisions for our global leaders in life sciences (see Labcorp SEC filings here). I have also held positions such as enterprise-level Data Scientist at Bayer (a EURO STOXX 50 company), Quantitative Researcher (apprenticeship) at AQR (a global hedge fund pioneering in alternative quantitative strategies to portfolio management and factor-based trading), and Equity Trader at T3 Trading on Wall Street (where I was briefly licensed Series 56 by FINRA). I supervise a small fund specializing in algorithmic trading (since 2011, performance is here) in equity market, cryptocurrencies, and real estate investment. I also run my own monetized YouTube Channel. Feel free to add me on LinkedIn.

About the Book

- This book provides a comprehensive yet accessible overview of foundational neural network architectures driving recent advancements in artificial intelligence. Beginning with an introduction to deep neural networks, the author outlines key principles and design components underpinning complex intelligent systems. Convolutional and recurrent neural networks are also explored in detail, elucidating their specialized capabilities in processing image, text, speech, and time-series data. Expanding beyond supervised learning, the book dives into reinforcement learning for developing intelligent agents that can operate independently in dynamic environments. Groundbreaking algorithms behind large language models like ChatGPT and GPT are analyzed as well, revealing inner workings of systems capable of sophisticated text generation. While avoiding advanced mathematics, the book focuses on high-level conceptual understanding and real-world applications throughout. With informative diagrams and clear prose, readers gain insight into the fundamentals behind artificial intelligence innovations transforming society today. Ideal for beginners and moderately experienced practitioners alike.


Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.
Neural machine translation by jointly learning to align and translate.
arXiv preprint arXiv:1409.0473, 2014.


Xi Chen, Nedialko B Dimitrov, and Lauren Ancel Meyers.
Uncertainty analysis of species distribution models.
PloS one, 14(5):e0214190, 2019.


Noam Chomsky.
Systems of syntactic analysis.
The Journal of Symbolic Logic, 18(3):242–256, 1953.


Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
Bert: Pre-training of deep bidirectional transformers for language understanding.
arXiv preprint arXiv:1810.04805, 2018.


Dagmar Divjak, Serge Sharoff, and Tomaž Erjavec.
Slavic corpus and computational linguistics.
Journal of Slavic linguistics, 25(2):171–200, 2017.


Howard Gardner.
Beyond the iq: Education and human development.
Harvard Educational Review, 57(2):187–196, 1987.


Frederick Jelinek.
Statistical methods for speech recognition.
MIT press, 1998.


Sotiris B Kotsiantis.
Decision trees: a recent overview.
Artificial Intelligence Review, 39:261–283, 2013.


Márk Lajkó, Viktor Csuvik, and László Vidács.
Towards javascript program repair with generative pre-trained transformer (gpt-2).
In *Proceedings of the Third International Workshop on Automated Program Repair*, pages 61–68, 2022.


Yann LeCun, Yoshua Bengio, et al.
Convolutional networks for images, speech, and time series.
The handbook of brain theory and neural networks, 3361(10):1995, 1995.


Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324, 1998.


Jürgen Schmidhuber.
Scientific integrity and the history of deep learning: The 2021 turing lecture, and the 2018 turing award.
Technical report, Technical Report IDSIA-77-21 (v3), IDSIA, Lugano, Switzerland, 2021–2022, 2022.


Kaj Sotala.
How feasible is the rapid development of artificial superintelligence?
Physica Scripta, 92(11):113001, 2017.


ChatGPT Generative Pre-trained Transformer and Alex Zhavoronkov.
Rapamycin in the context of pascal's wager: generative pre-trained transformer perspective.
Oncoscience, 9:82, 2022.


Amit Kumar Tyagi and Ajith Abraham.
Recurrent neural networks: Concepts and applications.
CRC Press, 2022.


Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
Advances in neural information processing systems, 30, 2017.