(17)

Convolutional Neural Networks (CNN's)

(1) Input Image: 3d matrix of pixel values.

(2) Convolutional layer :-

- Applies filters (kernels) to input to extract feature maps.

$$I_{i,j} = \sum_{m,n} X_{i+m, j+n} \cdot W_{m,n} + b.$$

(3) ReLU activation

(4) Pooling layer (subsampling) :-

- Maxpooling.
- Average pooling

(5) Fully connected layer :-

- After convolution and pooling, high level feature maps are flattened into a vector and passed through fully connected

(dense) layers.

(5) Softmax Layer (for classification):-

In multiclass classification, the final layer is softmax function that outputs probabilities for each class.

Limitations:-

1. Computationally intensive
2. Requires large datasets.
3. Sensitive to rotation and occlusion:- traditional CNN's may struggle with rotated or partially hidden objects unless additional preprocessing steps are applied.

Popular CNN architectures:-

1) LeNet:- One of the earliest CNNs designed for handwritten digit recognition.

2. AlexNet:- Won the ImageNet competition in 2012.

3. VGGNet:- Uses many layers of small filters (3×3) to improve accuracy.

4. ResNet:- Introduces residual connections to address the vanishing gradient problem in deep neural networks.

5. Inception Network (GoogleNet):-
Uses multiple layers in parallel to learn features at different scales.

(18). Recurrent Neural Network:-

(1) Input sequence:-

RNN's process sequence of inputs, such as words in a sentence or time series data.

(2). Hidden State:-

At each time step t, the network maintains a hidden state h_t, that acts like memory, storing information from previous state.

(3) Recurrence Formula:-

At each time step t, the hidden state is updated.

$$h_t = f(Wx_t + Uh_{t-1} + b).$$

x_t: input at timestep t.

h_{t-1}: Hidden state from previous state.

f: activation function

4. Output:-

The output y_t at each time step is computed from current hidden state.

$$y_t = g(Vh_t + c).$$

g - activation function (like Softmax)

Challenges / Limitations:-

(1) vanishing gradient
(2) exploding gradient.
(3) Training time.

Types of RNN:-

(1) vanilla RNN
(2). Long short-term memory.
(3) Gated Recurrent Unit (GRU):-

A simpler version of LSTM with fewer gates, making it easier to train.

4. Bidirectional RNN:-

Proces the sequence in both forward and backward directions to capture the past and future information.

---

(19)

Long Short -Term Memory (LSTM):-

(1) Forget gate:- (f_t)

- decides how much previous cell state (c_{t-1} should be forgotten.
- uses a sigmoid activation to output a value b/w 0 and 1.

(2) Input gate:- (i_t)

decides how much new information from current input x_t should be added to cell state. (percentage)

3. Candidate cell state:- ($\tilde{C}_t$)

- Computes potential new information to be added to cell state using tanh activation function.

4. Update cell state:- (C_t) The new state is a combination of old cell state C_{t-1} (controlled by forget gate) and the new candidate value $\tilde{C}_t$ (controlled by input gate).

5. Output gate:- (O_t) :-

Decides how much of cell state should be exposed as hidden state for the current time step.

6. Compute new hidden state:-

The hidden state is filtered version of cell state passed through a tanh activation, and controlled by output gate.

Mathematical summary of LSTM :-

At each timestep t,

(1) Forget gate : $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

(2) Input gate :- $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

(3) Candidate cell state :- $\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

(4) Update cell state :- $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$

(5) Output gate :- $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

(6) Hidden state :- $h_t = o_t \cdot \tanh(C_t)$.

(20) Word embedding:-

- Word embedding is a technique used in natural language processing (NLP) to represent words or phrases as continuous valued vectors.
- The key idea is to map words into a high dimensional space where semantically similar words have similar vector representations.
- These vectors capture the context, meaning & relationship between words.

Why use word embedding?

- Traditional methods like one hot encoding, represent words as sparse, binary vectors.
- However, these approaches have limitations.
  (1) High dimensionality.
  (2) No semantic similarity.

- Wood embeddings solve these issues by
  - Compressing words onto dense vectors with fewer dimensions.
  - Encoding semantic meaning by placing related words closer in vector space

~~How word embeddings works~~

popular word-embedding techniques:-

(1) Word2Vec (Google):-

- Word2Vec learns embeddings using a neural network.
- Two models:-

(1) Continous Bag of words:-
Predicts a target word from the surrounding text.

② Skip-gram:- predicts surrounding text words given a target word.

② GloVe (Global vectors for word representation)

- GloVe is trained on word co-occurance matrices and tries to capture statistical information about word pairs in the corpus.

③ FastText (Facebook):-

- Extends word2vec by representing words as combinations of character-level n-grams, making it effective for morphologically rich languages and handling out-of-vocabulary words.

4. BERT (Bidirectional Encoder Representations from Transformers):-

- Contextual embeddings:- Unlike Word2vec,

BERT generates different embeddings for the same word based on sentence context.

---

(21).

Sequence-to-Sequence (Seq2Seq) Model :-

- A sequence-to-sequence is a type of neural network architecture used for mapping one sequence to another sequence, such as machine translation, text summarization, and chatbot responses.

- The key idea is to convert an input sequence into an intermediate representation using an encoder, then generate the target sequence using a decoder.

Architecture of sequence-to-sequence models:-

(1) Encoder: Preprocess the input sequence and creates a contextual representation (also called a thought vector).

(2) Decoder:- Uses the encoders output to generate the target sequence, step by step.

How Seq2Seq Works?-

(1) Encoder:-

- Takes the input sequence (eg. a sentence in English) and processes it into a fixed size vector.
- Each word is transformed into a word embedding (vector) and passed through RNN's, LSTM's, GRU's.

- The last hidden state of the encoder acts as summary (context vector) of the input sequence.

(2) Decoder :-

- The decoder starts with context vector from the encoder and generates the output sequence (eg. translated sentence in French).

- The decoder is also RNN-based (or uses LSTM or GRU) and generates each word in the target sequence one step at a time.

- The decoder uses the previous word's output as input for next time step, along with current hidden state.

(3) Training with Teacher forcing :-

During Training, the actual target word from the dataset is used to input the decoder instead of word predicted by the model.

Attention mechanism in seq2seq :-

- Instead of using just the last hidden state of encoder as the context vector, attention allows the decoder to look at all encoder hidden states at every step. This makes the model more effective for longer sequences.

$$\alpha_{t,i} = \frac{\exp(h_t^{dec} \cdot h_i^{enc})}{\sum_j \exp(h_t^{dec} \cdot h_j^{enc})}$$

$\alpha_{t,i}$: attention b/w decoder step t and encoder step i.

the context vector is computed using

$$c_t = \sum_i \alpha_{t,i} \cdot h_i^{enc}.$$

The decoder use this dynamic context vector c_t at each time step to generate the output.

---

(22).

Attention Mechanism

- The attention mechanism is a technique that enables models, especially in seq2seq tasks, to focus on specific parts of input when generating each word in the output.

- It allows the model to dynamically decide which input tokens are most relevant at each step of the output generation.

- Attention solves the information bottleneck problem in traditional seq2seq models.
- By giving the decoder direct access to all encoder outputs, attention enables better performance, especially on long sequences.

How Attention Works:-

At each decoding step:-

1. Calculates a score for how relevant each input token is to the output token.
2. Generates attention weights by applying a softmax function to these scores.
3. Computes a weighted sum of the encoder outputs using the attention weights, producing a context vector.
4. Uses the context vector along with the decoder's hidden state to generate the next word.

~~text word~~).

Mathematical formulation:-

1. Score calculation.
   - ① dot product.
   - ② Additive (Bahdanau Attention)

   } different types of attention score function

2. Softmax to generate weights.
3. Context vector calculation.
4. Generate output token.

(23.) Transformer Neural Networks:-

- A transformer consists of 2 main components:-

(1) Encoder :- Process the input sequence.

(2). Decoder :- Generates the output sequence, step-by-step.

Each encoder and decoder is composed of layers with multi-head self-attention and feedforward neural networks.

Architecture Components of Transformers :-

(1) Input Embedding :-

- Converts each word or token into a dense vector representation.
- Positional encoding is added to the embeddings to capture the order of words in the sequence since transformers do not inherently handle order.

2. Self-attention mechanism :-

- At each layer, the model computes attention scores to decide which parts of input are most relevant.
- Query, key, and value vectors are derived from the input sequence to compute these scores.

3. Multi-head Attention :-

- Instead of using a single attention mecha-nism, transformers apply multiple attention heads in parallel.
- This helps the model focus on different parts of input simultaneously.

4. Feed forward layers :-

- After the attention mechanism, the output passes through a fully connected feed forward

network to introduce non-linearity.

5. Residual Connections and layer Normalization:-

- Each layer has a residual connections and layer normalization to stabilize and improve training.

Workflow of transformers:-

1. Tokenization and input preparation.
2. Positional encoding
3. Encoder: process input sequence
4. Decoder: generate output sequence.
5. Multihead self-attention mechanism.
6. Feedforward layer and Residual connections.
7. Inference: output generation.

Decoder-Only Transformers: (like gpt)

(24).

Normal transformers vs decoder only transformer :-

A normal Transformer :-

- A normal transformer uses one unit to encode the input, called the encoder, and a seperate unit to generate the output, called the decoder.
- A normal transformer uses two types of Attention during inferance : self Attention and Encoder-Decoder Attention.
- During Training, a normal transformer uses masked self attention but only on the output.

decoder-only Transformer

~~- A decoder only transformer uses a single type of attention.~~

- A decoder only transformer has a single unit for both encoding the input and generating the output.

- A decoder only transformer uses a single type of attention, masked self-attention.

- A decoder-Only Transformer uses masked self-Attention all the time on everything the input and output.

(25) Hugging Face?-

- Machine learning platform that provides easy-to-use tools, libraries and pre trained models for Natural language processing and other AI tasks.
- It has become the go-to framework for building state of art models in areas such as text classification, translation, question answering, chatbots and transformer based models.

Hugging face offers?

(1) Transformers library
(2) Datasets library.
(3) Hugging face hub.