MODULE-IV

RECURRENT NEURAL NETWORKS

A Recurrent Neural Network (RNN) is a type of artificial neural network specifically designed for processing sequential data. Unlike traditional feedforward neural networks, RNNs can handle inputs with temporal or sequential dependencies, making them well-suited for tasks such as time series prediction, language modeling, and sequence classification.

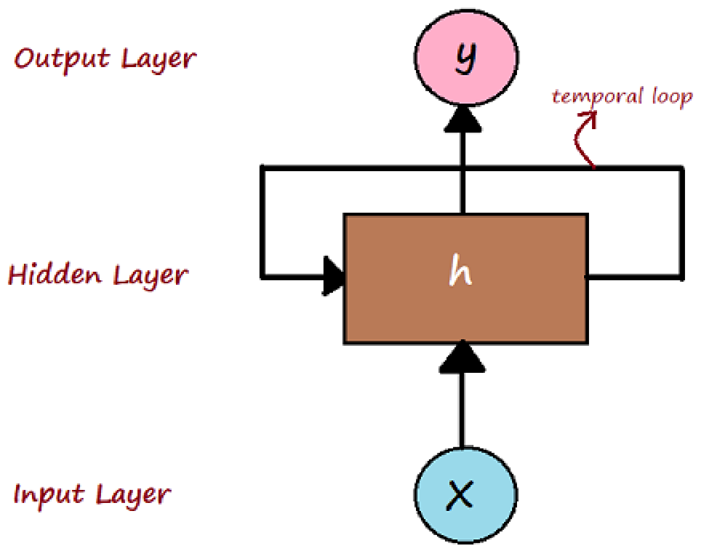
Key Concepts of RNNs

- Sequential Processing: RNNs are adept at handling sequences of data (e.g., sentences or time series) where the order of elements matters. They achieve this by maintaining a hidden state that captures information from previous inputs in the sequence.

- Recurrent Connections: RNNs feature recurrent connections, which allow them to process the current input while also considering the hidden state from the previous time step. This recurrent mechanism enables the network to retain and utilize memory of past inputs.

- Hidden State: The hidden state in an RNN is a vector that encodes information about the network's previous inputs. This state is updated at each time step based on both the current input and the previous hidden state.

- Weight Sharing: RNNs use the same weights (parameters) across all time steps. This weight sharing allows RNNs to efficiently process sequences of varying lengths.



Key Components of RNN

1. Input Layer (x):
    - The input layer is where the data enters the network. For sequences, this could be an individual word in a sentence, a time step in a time series, or another sequential data form.
2. Hidden Layer (h):
    - The hidden layer processes the input and previous hidden states. It maintains a "memory" of the sequence, with the hidden state carrying information from past time steps through recurrent connections.
3. Temporal Loop (Recurrent Connection):
    - The loop represents how the hidden state at time step ttt (denoted by hth_tht) is fed back into the hidden layer at the next time step t+1t+1t+1. This recurrent connection allows the RNN to retain and utilize information from previous inputs, enabling it to handle sequences and temporal data. This loop embodies the network’s short-term memory.
4. Output Layer (y):
    - The output layer provides the final prediction or output at each time step. For instance, in a language model, this could be the predicted next word, while in time series forecasting, it could be the predicted next value.

Types of RNN

1. Vanilla RNN
    - Description: The basic form of RNN, where neurons have self-connections to remember previous inputs. It learns sequential patterns by passing the hidden state from one time step to the next.
    - Limitations: Struggles with long-term dependencies due to vanishing/exploding gradients.
    - Applications: Simple time-series data, basic language modeling.

2. Long Short-Term Memory (LSTM)
    - Description: An advanced RNN that overcomes vanishing gradients with a more complex architecture featuring input, forget, and output gates. This structure allows LSTMs to remember information over long sequences.
    - Applications: Speech recognition, language translation, and time-series prediction where long-term dependencies are important.
3. Gated Recurrent Units (GRU)
    - Description: A simplified version of LSTMs with merged forget and input gates into a single "update gate." GRUs are less computationally intensive while still handling long-term dependencies effectively.
    - Applications: Text generation, machine translation, and tasks involving long sequences.
4. Bidirectional RNN (BRNN)
    - Description: Utilizes two RNNs: one processes the sequence from start to end (forward pass), and the other processes it from end to start (backward pass). This allows the network to consider context from both directions.
    - Applications: NLP tasks like sentiment analysis, machine translation, and speech recognition where context from both directions is useful.
5. Deep (Stacked) RNN
    - Description: Comprises multiple layers of RNNs stacked on top of each other to capture complex patterns through hierarchical feature learning.
    - Applications: Advanced NLP tasks, video processing, and other complex sequence tasks.
6. Recursive Neural Network
    - Description: Applied to hierarchical structures such as parse trees, processing inputs recursively. Suitable for analyzing structured data like sentence structures or trees.
    - Applications: Parsing tasks, sentiment analysis at the sentence or document level.
7. Attention-based RNNs

- Description: Incorporates an attention mechanism that allows the network to focus on specific parts of the input sequence, improving performance on long sequences.
- Applications: Machine translation, text summarization, image captioning, and speech synthesis.

Challenges with RNNs

1. Vanishing and Exploding Gradients
   - Description: During training, gradients can become too small (vanish) or too large (explode) as they propagate through time steps, complicating learning of long-term dependencies.

Impact: Limits the ability to retain information from earlier in the sequence, affecting performance on tasks requiring long-term context.

2. Difficulty in Learning Long-Term Dependencies
   - Description: RNNs struggle with capturing dependencies across long sequences, failing to retain crucial information from the start of a sequence to later time steps.
   - Impact: Limits effectiveness in tasks requiring long-range context, such as complex NLP tasks.
3. Slow Training Process
   - Description: Sequential data processing leads to longer training times, as RNNs process one time step at a time and cannot easily leverage parallel processing.
   - Impact: Results in slower training, especially with large datasets or long sequences.
4. Difficulty in Handling Long Sequences
   - Description: Performance can deteriorate with longer sequences, despite the theoretical capability to handle sequences of arbitrary length.
   - Impact: Poor performance on tasks involving very long sequences, such as entire documents or extensive time-series data.
5. Lack of Parallelization

   - Description: RNNs process input sequentially, making it challenging to fully utilize parallel processing capabilities of modern hardware.
   - Impact: Slower and less efficient compared to architectures that can process input in parallel.

LSTM Networks

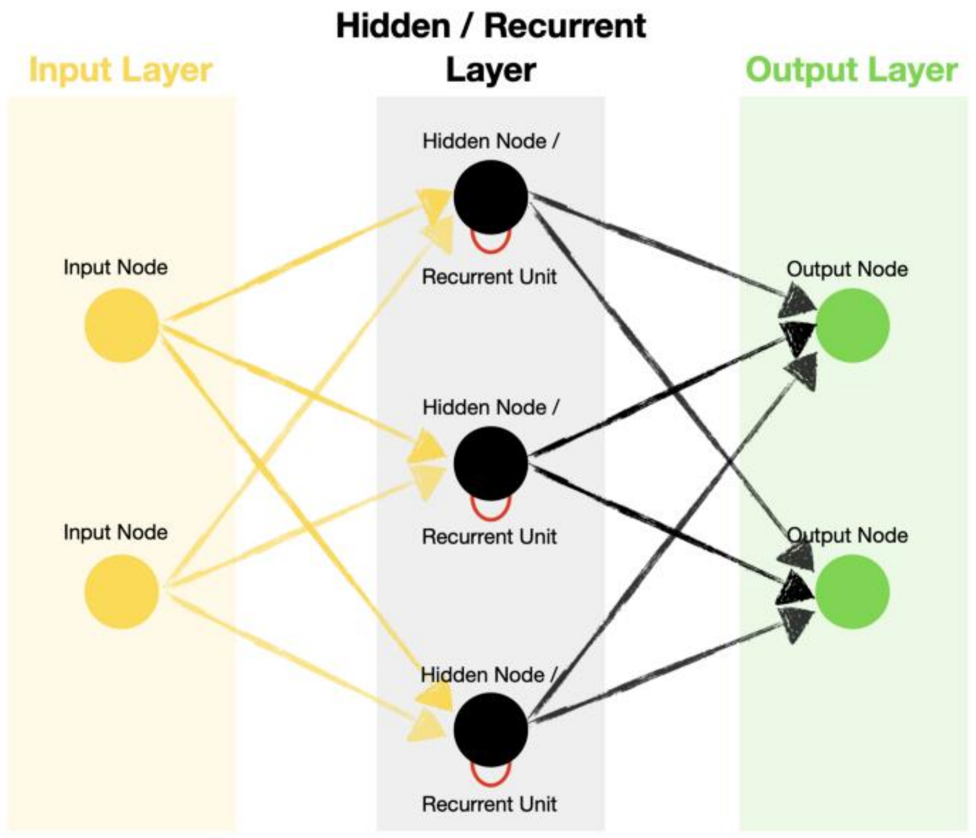
LSTM networks are designed to address the limitations of traditional RNNs, particularly with long-term dependencies and vanishing gradient problems. They are highly effective for sequential data tasks and are commonly used in areas like Natural Language Processing (NLP), time-series forecasting, and speech recognition.

Key Features of LSTM

1. Cell State (Memory):
   - LSTMs maintain a cell state that acts as long-term memory, preserving context and patterns across many time steps. This memory is regulated by gates to decide what information to keep or forget.
2. Gates:
   - LSTMs use a gating mechanism to control the flow of information:
     - Forget Gate: Decides what information to discard from the cell state.
     - Input Gate: Determines what new information to add to the cell state.
     - Output Gate: Controls the output based on the cell state and determines the next hidden state.
3. Cell State Update:
   - The cell state is updated by combining the old cell state (after the forget gate) and new information (from the input gate).
4. Hidden State:
   - The hidden state hth_tht is computed from the updated cell state and output gate, representing the short-term memory at each time step and being passed to the next time step or layer.

How LSTM Solves Key Problems

1. Vanishing Gradient Problem:
    - LSTMs mitigate the vanishing gradient problem by using the forget gate to retain important information and the cell state to carry forward relevant information without repeated multiplication.
2. Long-Term Dependencies:
    - The gating mechanism allows LSTMs to retain or discard information from earlier time steps, enabling them to capture long-term dependencies effectively.



Standard Recurrent Neural Network architecture.

LSTM Equations Summary:

- Forget Gate:

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

- Input Gate:

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

- Candidate Cell State:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- Cell State Update:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Output Gate:

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

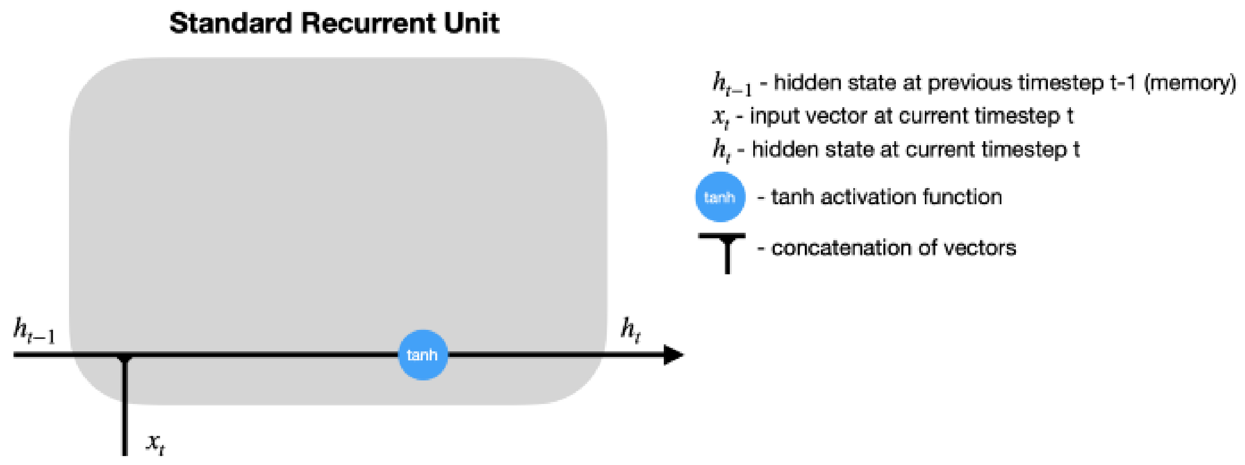
- Hidden State:

$$h_t = o_t * \tanh(C_t)$$

LSTMs (Long Short-Term Memory networks) are designed to overcome specific limitations of standard RNNs (Recurrent Neural Networks), particularly in handling long-term dependencies. Here's a comparison highlighting the main differences:

1. **Handling of Long-Term Dependencies:**
   - **Standard RNN:** RNNs often struggle with long-term dependencies due to the vanishing gradient problem. During backpropagation through time (BPTT), gradients can become very small, making it difficult for RNNs to learn from long sequences and retain information from earlier time steps.
   - **LSTM:** LSTMs are specifically engineered to address this issue. They use a memory cell and gating mechanisms to regulate the flow of information. This design helps LSTMs retain important information over long sequences and effectively manage the vanishing gradient problem.
2. **Architecture and Complexity:**

- **Standard RNN:** RNNs have a simpler architecture where each unit updates the current hidden state based on the current input and the previous hidden state. While this simplicity makes RNNs easy to implement, it limits their ability to remember information over long periods.
- **LSTM:** LSTMs feature a more complex architecture with memory cells and multiple gates (input, forget, and output gates). These components work together to control the retention and updating of information, allowing LSTMs to maintain context and capture long-term dependencies more effectively.


Standard Recurrent Unit
h_{t-1}
x_t
tanh
h_t
h_{t-1} - hidden state at previous timestep t-1 (memory)
x_t - input vector at current timestep t
h_t - hidden state at current timestep t
tanh - tanh activation function
- concatenation of vectors

Standard RNN recurrent unit.

- **LSTM**: LSTMs have a more complex architecture with **gates** (input gate, forget gate, output gate) and a **cell state** that acts as long-term memory. These gates control the flow of information and decide what to remember, forget, or output at each time step.
    - Forget Gate: Controls which information to discard.
    - Input Gate: Controls which new information to add.
    - Output Gate: Controls what part of the cell state is output as the hidden state.

3. Memory Mechanism:

- **Standard RNN**: In RNNs, the hidden state carries the memory, and it is passed from one time step to the next. The hidden state, however, is highly susceptible to the vanishing gradient problem, making it hard for RNNs to retain information over long periods.
- **LSTM**: LSTMs use a **separate cell state** to store long-term memory. The cell state allows LSTMs to maintain and carry forward relevant information over long time steps without constantly being overwritten, leading to better handling of long sequences.

4. Vanishing Gradient Problem:

- **Standard RNN**: Standard RNNs are highly prone to the vanishing (or exploding) gradient problem during backpropagation. As gradients are propagated through many time steps, they tend to diminish, making it difficult for the model to learn long-range dependencies.
- **LSTM**: LSTMs mitigate the vanishing gradient problem by using the **cell state** and **gates** to control the flow of gradients more effectively. The forget and input gates enable LSTMs to selectively update the cell state, preserving important information for longer periods.

5. Gating Mechanisms:

- **Standard RNN**: RNNs do not have gates to control the flow of information. The hidden state is updated directly by a non-linear activation function (like tanh), which makes it difficult to modulate the information flow across time steps.
- **LSTM**: LSTMs have three main gates—**forget gate**, **input gate**, and **output gate**—that regulate the flow of information. These gates allow LSTMs to decide at each time step which information to retain, forget, or output, making them more flexible and powerful in managing sequential data.

6. Capability for Longer Sequences:

- **Standard RNN**: Due to the limitations of vanishing gradients, RNNs perform poorly on tasks involving long sequences where relationships between distant time steps are important (e.g., long sentences or extended time-series data).
- **LSTM**: LSTMs are designed to handle such tasks. By retaining long-term memory through their cell state, they can effectively capture long-range dependencies and perform well even when sequences are long.

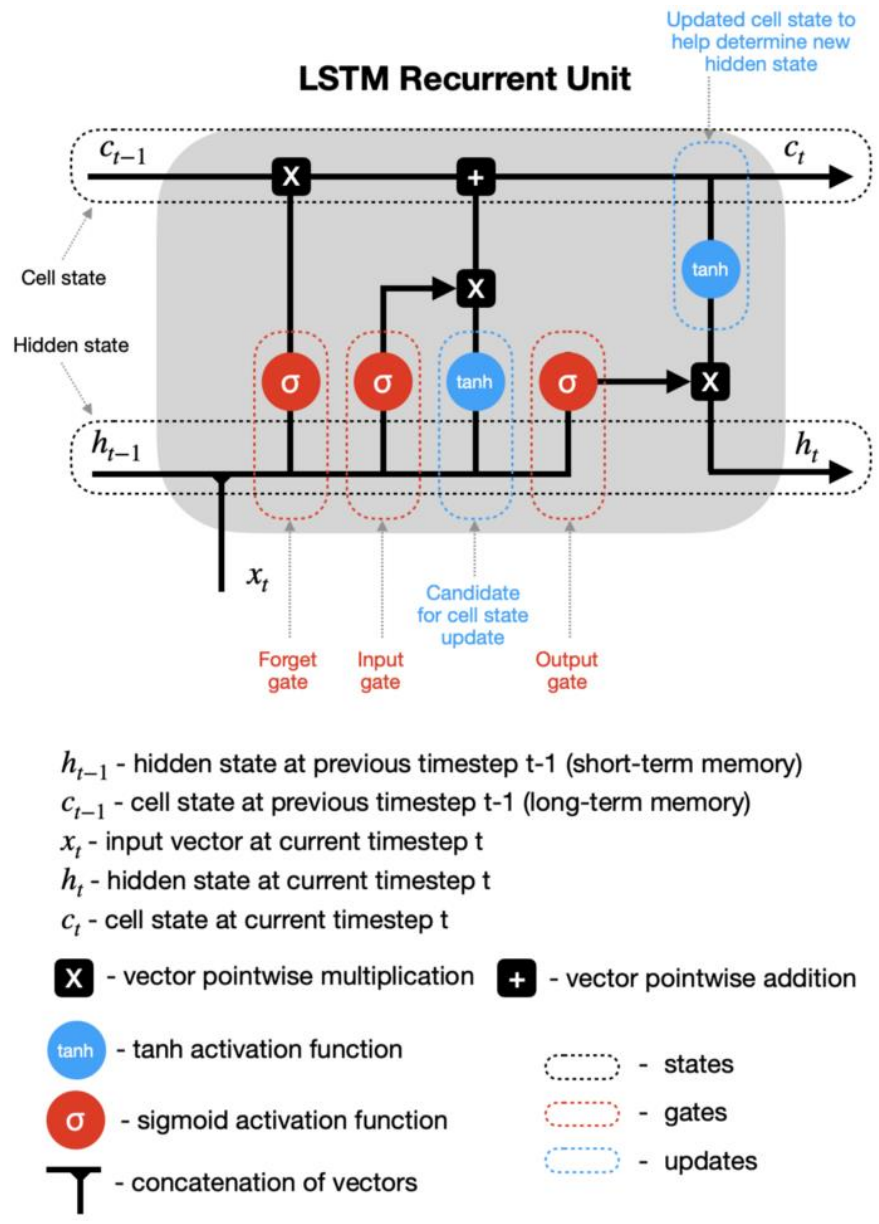
7. Training Efficiency:

- **Standard RNN**: While RNNs are simpler and faster to train for short sequences, they are inefficient for longer sequences because they fail to retain critical long-term information.
- **LSTM**: LSTMs are more computationally expensive due to their complex architecture with multiple gates. However, they perform significantly better on tasks requiring long-term memory, justifying the additional computational cost in many applications.

8. Applications:

- **Standard RNN**: RNNs can be effective in tasks with shorter sequences or tasks where short-term dependencies dominate. However, they tend to struggle with tasks that require long-range context, like text generation or speech recognition involving longer inputs.

- **LSTM**: LSTMs are better suited for tasks requiring long-term dependencies, such as:
  - Language modeling
  - Machine translation
  - Time-series prediction
  - Speech and video recognition
  - Text generation


LSTM Recurrent Unit
Updated cell state to help determine new hidden state
c_{t-1}
c_t
Cell state
Hidden state
tanh
σ
h_{t-1}
h_t
x_t
Candidate for cell state update
Forget gate
Input gate
Output gate
h_{t-1} - hidden state at previous timestep t-1 (short-term memory)
c_{t-1} - cell state at previous timestep t-1 (long-term memory)
x_t - input vector at current timestep t
h_t - hidden state at current timestep t
c_t - cell state at current timestep t
X - vector pointwise multiplication
+ - vector pointwise addition
tanh - tanh activation function
σ - sigmoid activation function
- concatenation of vectors
- states
- gates
- updates

Gated Recurrent Unit:

The Gated Recurrent Unit (GRU) is a streamlined variant of the Long Short-Term Memory (LSTM) network, designed to tackle similar issues with long-term dependencies in Recurrent Neural Networks (RNNs) while being more computationally efficient. GRUs have fewer parameters compared to LSTMs due to the merging of some gates, which makes them faster and simpler to train.

Key Features of GRU:

1. Simpler Architecture: GRUs use only two gates—reset and update—compared to the three gates in LSTMs (input, forget, and output).
2. No Separate Cell State: GRUs do not maintain a separate cell state. Instead, they rely on the hidden state to store and transfer information.
3. Fewer Parameters: By combining the input and forget gates into a single "update gate," GRUs reduce the number of parameters, speeding up training while achieving performance similar to LSTMs.
4. Efficient for Shorter Sequences: Due to their simpler structure, GRUs are often faster and perform well on shorter sequences, though they can also handle longer sequences effectively.

GRU Structure:

At each time step ttt, the GRU updates the current hidden state hth_tht based on the current input xtx_txt and the previous hidden state ht−1h_{t-1}ht−1. The GRU comprises two main components:

1. Update Gate: This gate determines how much of the previous hidden state ht−1h_{t-1}ht−1 should be retained and how much of the new input xtx_txt should be incorporated.
2. Reset Gate: This gate controls how much of the previous hidden state ht−1h_{t-1}ht−1 should be forgotten when calculating the new hidden state.
    - Update Gate: Combines the functions of the input and forget gates from LSTMs, deciding the extent to which the old hidden state is preserved and new input is integrated.
    - Reset Gate: Determines how much of the previous hidden state to ignore, influencing the calculation of the new candidate hidden state.

- Candidate Hidden State: h~t\tilde{h}_th~t is computed from the current input xtx_txt and the reset-modified previous hidden state ht−1h_{t-1}ht−1. This represents the new information to be added to the hidden state.

Comparison of GRU vs. LSTM:

Feature	GRU	LSTM
Gates	2 (Update, Reset)	3 (Input, Forget, Output)
Cell State	No separate cell state (only hidden state)	Has a separate cell state
Complexity	Simpler (fewer parameters)	More complex (more parameters)
Training Time	Faster due to fewer gates and parameters	Slower due to more gates and parameters
Performance on Short Sequences	Generally performs well	Similar performance
Performance on Long Sequences	May struggle with long-term dependencies	Better for long-term dependencies
Use Cases	Applications needing less memory or faster training	Applications needing better handling of long sequences

Applications of RNN Networks

Recurrent Neural Networks (RNNs) are effective for applications involving sequential data, where the order of inputs is crucial. Their variants, like LSTMs and GRUs, are used in various fields such as natural language processing, time-series analysis, and video processing. Here are some notable applications:

1. Natural Language Processing (NLP):
    - Machine Translation: RNNs translate text between languages by encoding and generating sentences. Modern systems often use RNN-based models like LSTMs and GRUs with attention mechanisms.
    - Text Generation: RNNs generate text by learning patterns from data, including language models and automatic code generation.

- Sentiment Analysis: RNNs analyze sequential word data to understand the sentiment expressed in a text.
- Speech Recognition: RNNs convert spoken language into text by capturing dependencies in audio signals.
- Named Entity Recognition (NER): Identifies entities such as names and locations in text.

2. Time-Series Prediction:
    - Stock Market Prediction: Predicts stock prices or trends based on historical financial data.
    - Weather Forecasting: Forecasts weather conditions by learning from historical patterns.
    - Electricity Load Forecasting: Predicts electricity demand to manage energy efficiently.
    - Anomaly Detection: Identifies unusual events in time-series data, such as sensor data or financial transactions.
3. Speech Processing:
    - Speech-to-Text (Automatic Speech Recognition): Converts spoken words into text, used in virtual assistants like Siri and Google Assistant.
    - Speech Synthesis (Text-to-Speech): Generates human-like speech from written text, integral to voice assistants.
    - Speaker Recognition: Identifies or verifies speakers by analyzing voice patterns.
4. Video Analysis:
    - Action Recognition: Identifies human actions in videos by analyzing sequences of frames.
    - Video Captioning: Generates descriptive captions for videos by understanding visual sequences.
    - Gesture Recognition: Recognizes gestures from video sequences, useful in human-computer interaction.
5. Music Generation:
    - Automatic Music Composition: Creates new music by learning from existing compositions.

   - Melody Prediction: Predicts the next note in a sequence to generate harmonious music in real-time.

6. Handwriting Recognition:
   - Offline Handwriting Recognition: Recognizes handwritten text from scanned images by analyzing stroke sequences.
   - Real-Time Handwriting Recognition: Identifies handwritten text as it is written, such as on tablets or touchscreens.

Application to Automatic Image Captioning

Automatic Image Captioning involves generating textual descriptions for images, merging computer vision and natural language processing. RNNs, particularly LSTMs and GRUs, are crucial for generating coherent and contextually accurate captions.

How Image Captioning Works:

1. Image Feature Extraction (using CNN):
   - A Convolutional Neural Network (CNN) extracts features from the input image. The CNN acts as an encoder, summarizing the image into a feature vector (or set of features).
   - Extracted features represent the image's content, objects, and context, and are passed to the RNN.
2. Caption Generation (using RNN):
   - The extracted image features serve as input to the RNN. Typically, the CNN features initialize the RNN’s hidden state, providing context for generating the caption.
   - Word-by-Word Generation: The RNN generates the caption word by word. At each time step, it predicts the next word based on the image features and the previously generated words. This process continues until the model produces an end token, signifying the end of the sentence.

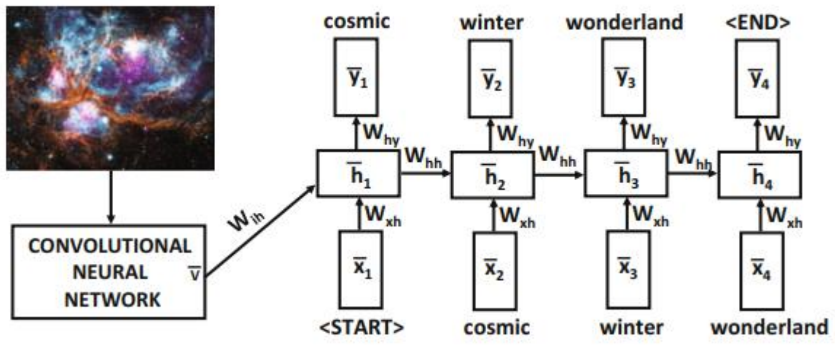


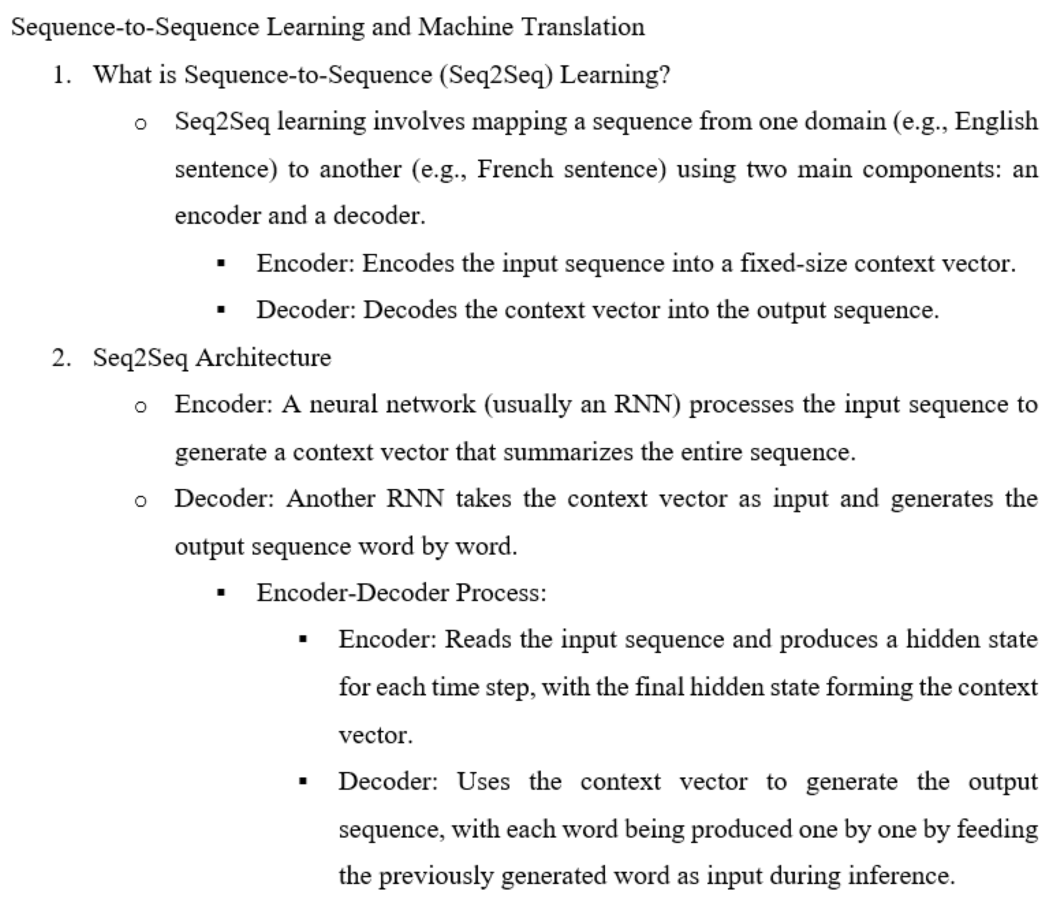
Figure 7.9: Example of image captioning with a recurrent neural network. An additional convolutional neural network is required for representational learning of the images. The image is represented by the vector $\overline{v}$, which is the output of the convolutional neural network. The inset image is by courtesy of the National Aeronautics and Space Administration (NASA).

Training with Paired Image-Text Data

1. Model Training
   - The model is trained on a dataset containing images paired with corresponding captions, such as the MS COCO dataset. During training, the model learns to align image features with the words in the captions by minimizing the discrepancy between predicted and actual words.
   - Teacher forcing is employed, where the ground truth word at each time step is used as input during training. During inference, the previously generated word is used as input for generating the next word.
2. Attention Mechanism (Optional Enhancement)
   - Modern image captioning models often incorporate an attention mechanism to enhance performance. This mechanism enables the model to focus on different parts of the image when generating each word in the caption, improving both accuracy and relevance.
   - Instead of feeding a fixed-size image feature vector into the RNN, attention mechanisms compute a weighted combination of feature vectors from various regions of the image. This allows the model to concentrate on pertinent parts of the image as it generates each word.
3. Example Architecture: Encoder-Decoder with Attention

- Encoder (CNN): A pre-trained CNN (e.g., ResNet) extracts features from the image, representing various regions or aspects.
- Attention Mechanism: At each time step, attention weights are calculated to determine which part of the image should be focused on for generating the next word.
- Decoder (RNN): The decoder, typically an LSTM or GRU, generates the caption word by word. The attention mechanism dynamically provides relevant image features for each word in the sequence.
- Loss Function: Cross-entropy loss is commonly used, comparing the predicted word at each time step with the actual word in the caption.
- Inference Strategy: Beam search is utilized during inference to explore multiple potential caption sequences and select the most suitable one, rather than generating captions greedily.

Sequence-to-Sequence Learning and Machine Translation



Sequence-to-Sequence Learning and Machine Translation

1. What is Sequence-to-Sequence (Seq2Seq) Learning?
   - Seq2Seq learning involves mapping a sequence from one domain (e.g., English sentence) to another (e.g., French sentence) using two main components: an encoder and a decoder.
     - Encoder: Encodes the input sequence into a fixed-size context vector.
     - Decoder: Decodes the context vector into the output sequence.
2. Seq2Seq Architecture
   - Encoder: A neural network (usually an RNN) processes the input sequence to generate a context vector that summarizes the entire sequence.
   - Decoder: Another RNN takes the context vector as input and generates the output sequence word by word.
     - Encoder-Decoder Process:
       - Encoder: Reads the input sequence and produces a hidden state for each time step, with the final hidden state forming the context vector.
       - Decoder: Uses the context vector to generate the output sequence, with each word being produced one by one by feeding the previously generated word as input during inference.

Applications of Seq2Seq

- Machine Translation: Translating sentences between languages.
- Speech Recognition: Converting spoken language to text.
- Text Summarization: Condensing long documents into shorter summaries.
- Dialogue Systems: Generating responses in chatbots or conversational agents.

Application to Sentence-Level Classification

Understanding Sentence-Level Classification

- Sentence-level classification involves assigning a label to an entire sentence. Common tasks include:
  - Sentiment Analysis: Classifying sentiment as positive, negative, or neutral.
  - Spam Detection: Identifying spam or non-spam content.
  - Topic Classification: Categorizing sentences based on topics like sports, politics, or technology.

2. Seq2Seq Architecture for Classification
   - Encoder-Decoder Model for Classification:
     - Encoder: Processes the input sentence to generate a context vector that represents the entire sentence.
     - Decoder: Modified to output a classification label instead of generating a sequence. This is achieved using a classification layer (e.g., a fully connected layer with a softmax activation).
   - Example Workflow:
     - Input Sentence: "I loved the movie."
     - Encoder: Processes the sentence into a context vector.
     - Classifier Layer: Predicts the class label (e.g., "positive" for sentiment analysis).
3. Using Seq2Seq Models with Attention
   - Attention mechanisms can be employed to enhance sentence-level classification by allowing the model to focus on relevant parts of the sentence.
     - Attention Mechanism: Weighs different parts of the sentence, improving focus on the most pertinent words or phrases.

        - Context Vector: Generated by the attention mechanism and used by the classification layer to determine the class label.

4. Example Applications of Seq2Seq for Sentence-Level Classification
    - Sentiment Analysis:
        - Input: "The product is fantastic."
        - Output: "Positive"
    - Spam Detection:
        - Input: "Congratulations! You have won a $1000 gift card."
        - Output: "Spam"
    - Topic Classification:
        - Input: "The new iPhone has an improved camera system."
        - Output: "Technology"
5. Training Seq2Seq Models for Classification
    - Dataset Preparation: Develop a dataset with sentences and their corresponding labels.
    - Loss Function: Use a classification loss function, such as categorical cross-entropy, to train the model.
    - Evaluation: Assess model performance using metrics like accuracy, precision, recall, and F1-score.

End-to-End Speech Recognition

End-to-End Speech Recognition directly converts speech into text using a unified model, bypassing traditional intermediate steps like acoustic modeling and language modeling.

1. Unified Model Architecture
    - A single neural network architecture handles the entire speech recognition process, simplifying the pipeline compared to traditional multi-stage systems.
2. Types of End-to-End Models
    - Listen, Attend and Spell (LAS)
        - Architecture: Uses an attention mechanism and Seq2Seq model, combining CNNs or RNNs for feature extraction and an attention-based decoder.
        - Components:

        - Encoder: Extracts features from audio using CNNs or RNNs.
        - Attention Mechanism: Focuses on different audio parts for generating text.
        - Decoder: Produces the text sequence from the attended features.
    - Advantages: Handles variable-length input sequences and captures long-term dependencies.
  - Connectionist Temporal Classification (CTC)
    - Architecture: Uses RNNs to produce frame-level predictions and aligns them with text sequences using CTC loss.
    - Components:
      - Neural Network: Typically uses RNNs (e.g., LSTMs, GRUs).
      - CTC Loss Function: Aligns predictions with text sequences, simplifying training and inference.
    - Advantages: Simplifies training by removing explicit alignment and segmentation.
  - Transformer-Based Models
    - Architecture: Adapts transformers for speech recognition, utilizing self-attention mechanisms.
    - Components:
      - Encoder: Processes audio features with self-attention.
      - Decoder: Generates text from encoded features.
    - Examples: Models like DeepSpeech and wav2vec.
    - Advantages: Handles long-range dependencies and parallelizes computation, enhancing training and inference speed.

3. Benefits of End-to-End Speech Recognition
   - Simplified Pipeline: Reduces complexity by eliminating multiple stages.
   - Reduced Feature Engineering: Maps raw audio directly to text, minimizing manual feature extraction.
   - Improved Performance: End-to-end training optimizes the entire recognition process.
   - Adaptability: Handles diverse speech patterns and accents effectively.

4. Challenges and Considerations
    - Data Requirements: Needs extensive labeled speech data for training.
    - Computational Resources: Requires significant hardware for training and inference, especially with transformer models.
    - Interpretability: More challenging to interpret compared to traditional multi-stage systems.
5. Practical Applications
    - Voice Assistants: Used in virtual assistants like Siri, Alexa, and Google Assistant.
    - Transcription Services: Converts speech from meetings, interviews, or lectures into text.
    - Accessibility Tools: Creates subtitles and closed captions for videos.