ASSIGNMENT - 4

1. Can you think of a few applications for a sequence-to-sequence RNN? What about a

sequence-to-vector RNN? And a vector-to-sequence RNN?

Ans: Applications of RNN Architectures:

Sequence-to-Sequence RNN (seq2seq):

* Machine Translation: Core application, translating text from one language to another (<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>)
  + Text Summarization: Condensing long texts into concise summaries.
  + Chatbots: Generating human-like conversation responses.
  + Music Generation: Creating musical scores or samples.
  + Video Captioning: Automatically generating captions for videos.
* Sequence-to-Vector RNN:
  + Sentiment Analysis: Classifying text sentiment (positive, negative, neutral).
  + Anomaly Detection: Identifying unusual patterns in time series data (e.g., sensor readings).
  + Speech Recognition: Converting spoken language into text.
  + Video Summarization: Capturing the gist of a video in a feature vector.
* Vector-to-Sequence RNN:
  + Text Generation: Generating different creative text formats (poems, code, scripts) from a starting concept vector.
  + Music Generation: Creating musical pieces based on a high-level musical style vector.
  + Machine Question Answering: Generating answers to factual questions from a knowledge base vector.
  + Image Captioning (indirectly): Using an image feature vector as input to generate a text description.

2. Why do people use encoder–decoder RNNs rather than plain sequence-to-sequence RNNs

for automatic translation?

Ans: Encoder-Decoder RNNs for Machine Translation:

* Standard seq2seq models might struggle with long sentences due to vanishing gradients.
* Encoder-decoder architecture addresses this by:
  + Encoder: Processes the input sequence (source language) and compresses it into a context vector capturing the meaning.
  + Decoder: Uses the context vector and generates the output sequence (target language) word by word, attending to relevant parts of the input at each step.
  + This separation allows the model to handle long-range dependencies more effectively.

3. How could you combine a convolutional neural network with an RNN to classify videos?

Ans: Combining CNNs and RNNs for Video Classification:

* Convolutional Neural Network (CNN):
  + Extract spatial features from each video frame (e.g., edges, shapes, objects).
* Recurrent Neural Network (RNN):
  + Process the extracted features across time (sequence of video frames).
  + Capture temporal relationships and dynamics in the video (e.g., object motion, scene changes).
* Classification Layer:
  + Use the combined features from CNN and RNN to classify the video content (e.g., action recognition, genre classification).

4. What are the advantages of building an RNN using dynamic\_rnn() rather than static\_rnn()?

Ans: Advantages of dynamic\_rnn() over static\_rnn():

* dynamic\_rnn() allows for sequences of variable lengths within a single batch.
* This is crucial for real-world applications where input sequences might differ in size.
* static\_rnn() assumes all sequences have the same fixed length, which can be inefficient and inflexible.

5. How can you deal with variable-length input sequences? What about variable-length output

Sequences?

Ans: Dealing with Variable-Length Sequences:

Input Sequences:

* Padding: Pad shorter sequences with special tokens to a fixed length (common, but wastes computation).
* Packed Sequences: Group sequences by length for more efficient processing.

Output Sequences:

* Beam Search: Decodes multiple possible outputs simultaneously, exploring promising candidates and filtering out unlikely ones.
* Greedy Search: Simple approach, predicting the most likely word at each step, but can get stuck in local optima.

6. What is a common way to distribute training and execution of a deep RNN across multiple

GPUs?

Ans: Distributing Deep RNN Training Across GPUs:

* Data Parallelism: Replicate the model across multiple GPUs and distribute training data batches across them.
* Model Parallelism: Split the model itself across GPUs, with each GPU handling a different part (complex and requires careful memory management).
* Gradient Accumulation: Train on smaller batches on each GPU and accumulate gradients across them before updating model weights, reducing memory requirements on individual GPUs.