ETSI GS ENI 030 V4.1.1 (2024-03)

Experiential Networked Intelligence (ENI); Transformer Architecture for Policy Translation

Disclaimer

The present document has been produced and approved by the Experiential Networked Intelligence (ENI) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/ENI-0030v411_Trans_Arch

Keywords
artificial intelligence, cognition, language,
policy management

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Experiential Networked Intelligence (ENI).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document specifies how a transformer architecture may be used to translate intent policies from an end-user, application, or other external source to ENI Policies for use in cognitive networking and decision making in modern system design. The primary use cases are twofold:

1) to translate a natural language intent policy into an ENI Policy that uses a Domain-Specific Language; and

2) to translate an ENI Domain Specific Language intent policy into an implementation that uses a programming language, such as Python™ or Java®.

1 Scope

The present document enhances the information described in ETSI GR ENI 018 [i.1] and specifies how a transformer architecture can be used to translate input policies from an end-user, application, or other external source to ENI Policies for use in cognitive networking and decision making in modern system design.

The present document specifies a transformer-based architecture that can be used to parse, understand, and translate text. This enables different types of input policies to be translated into an appropriate ENI Policy using a transformer architecture.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS ENI 005 (V3.1.1): "Experiential Networked Intelligence (ENI); System Architecture".

[2] ETSI GS ENI 019 (V3.1.1): "Experiential Networked Intelligence (ENI); Representing, Inferring, and Proving Knowledge in ENI".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GR ENI 018 (V2.1.1) (08-2021): "Experiential Networked Interlligence (ENI); Introduction to Artificial Intelligence Mechanisms for Modular Systems".

[i.2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin: "Attention Is All You Need", 31st Conference on Neural Information Processing Systems, 2017.

[i.3] J. Wei, et al.: "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", January 2023.

[i.4] Z. Dai, et al.: "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context", 57th Conference for the Association for Computational Linguistics, 2019.

[i.5] J. Rae, et al.: "Compressive Transformers for Long-Range Sequence Modelling", November 2019.

[i.6] J. Devlin, et al.: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", Google AI, March 2019.

[i.7] Y. Liu, et al.: "RoBERTa: A Robustly Optimized BERT Pretraining Approach", University of Washington and Facebook AI, 2019.

[i.8] Z. Lan, et al.: "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations", International Conference on Learning Representation, February 2020.

[i.9] A. Radford, et al.: "Improving language understanding by generative pre-training", Technical Report, OpenAI blog, 2018.

[i.10] A. Radford, et al.: "Language models are unsupervised multitask learners", OpenAI blog, 1(8):9, 2019.

[i.11] T. Brown, et al.: "Language models are few-shot learners", Advances in Neural Information Processing Systems, 2020.

[i.12] L. Ouyang, et al.: "Training language models to follow instructions with human feedback", Advances in Neural Information Processing Systems, 2022.

[i.13] N. Stiennon, et al.: "Learning to Summarize from Human Feedback", Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020.

[i.14] OpenAI: "GPT-4 Technical Report", March 2023.

[i.15] Big Science Workshop: "BLOOM: A 176B-Parameter Open-Access Multilingual Language Model", June 2023 (final version, v4).

[i.16] A. Chowdery, et al.: "PaLM: Scaling Language Modelling with Pathways", October 2022.

[i.17] P. Barham, et al.: "Pathways: Asynchronous Distributed Dataflow for ML", March 2022.

[i.18] D. Driess, et al.: "PaLM-E: An Embodied Multimodal Language Model", March 2023.

[i.19] Google®: "PaLM 2 Technical Report", May 2023.

[i.20] K. Singhal, et al.: "Large language models encode clinical knowledge", July 2023.

[i.21] H. Touvron, et al.: "LLaMA: Open and Efficient Foundation Language Models", February 2023.

[i.22] H. Touvron, et al.: "Llama 2: Open Foundation and Fine-Tuned Chat Models", July 2023.

[i.23] H. Shirzad, et al.: "Exphormer: Sparse Transformers for Graphs", July 2023.

[i.24] C. Raffel, et. al.: "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer", Journal of Machine Learning Research, June 2020.

[i.25] H. Chung, et al.: "Scaling Instruction-Finetuned Language Models", December 2022.

[i.26] S. Zuo, et al.: "MoEBERT: from BERT to Mixture-of-Experts via Importance-Guided Adaptation", April 2022.

[i.27] S. Yao, et al.: "Tree of Thoughts: Deliberate Problem Solving with Large Language Models", May 2023.

[i.28] J. Chen, et al.: "Skills-in-Context Prompting: Unlocking Compositionality in Large Language Models", August 2023.

[i.29] J. Johnson, M. Douze, H. Jégou: "Billion-scale similarity search with GPUs", February 2017.

[i.30] P. Christiano, et al.: "Deep reinforcement learning from human preferences", originally published June 2017, revised February 2023.

[i.31] Z. Feng, et al.: "A Pre-Trained Model for Programming and Natural Languages", September 2020.

[i.32] Y. Wang, et al.: "CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation", September 2021.

[i.33] Y. Wang, et al.: "CodeT5+: Open Code Large Language Models for Code Understanding and Generation", May 2023.

[i.34] S. Mahdavi: "Large Language Models Encode Clinical Knowledge", December 2022.

[i.35] P. Lewis, et al.: "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", April 2021.

[i.36] S. Pan, et al: "Unifying Large Language Models and Knowledge Graphs: A Roadmap", June 2023.

[i.37] Y. Wu, et al.: "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", 2016.

[i.38] Microsoft®: "Megatron-DeepSpeed".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

active learning: learning algorithm that can query a user interactively to label data with the desired outputs

NOTE: The algorithm proactively selects the subset of examples to be labelled next from the pool of unlabelled data. The idea is that an ML algorithm could potentially reach a higher level of accuracy while using a smaller number of training labels if it were allowed to choose the data it wants to learn from.

attention: part of a neural architecture that dynamically computes a weighted distribution on input text, assigning higher values to more relevant elements

NOTE: Attention mimics cognitive attention as performed in the human brain. It selectively enhances some parts of the input data while diminishing other parts. Instead of encoding the input sequence into a single fixed context vector, the attention model develops a context vector that is filtered specifically for each output time step. The model then predicts next word based on context vectors associated with these source positions and all the previous generated target words.

batch learning: type of offline learning algorithm that is updated (i.e. retrained) periodically

catastrophic forgetting: tendency of an artificial neural network to forget previously learned information when learning new information

chatbot: computer program that simulates human conversation through text or voice interactions

concept drift: underlying statistical properties of the data an algorithm is trained on change over time

NOTE: Concept drift means that the LLM or transformer is not taking changing data and its meanings into account during inference time.

domain specific language: small human-understandable language that uses a higher level of abstraction to communicate and configure software systems for a particular application domain

embedding: vector that represents the meaning of a word or phrase to the LLM

ensemble model: technique created by combining the predictions of multiple base models

Extended Backus-Naur Fom: notation used to define the syntax of a formal language

foundation model: type of LLM trained on a vast quantity of data at scale (often by self-supervised learning or semi-supervised learning) such that it can be adapted to a wide range of downstream tasks

generative artificial intelligence: type of artificial intelligence that can create new content (e.g. text, images or music) by learning the patterns and structures of existing data and then using those patterns to generate new data that is similar to the original data

graph transformer: type of transformer that generalizes the transformer architecture to graphs

instruction tuning: making a language model more generic by training the model on a large set of varied instructions

language model: use of probabilistic and/or statistical mechanisms to determine the probability of a given sequence of words occurring in a sentence

large language model: type of self-supervised language model whose number of parameters in the model can change autonomously as it learns

mixture of experts model: technique where multiple expert models are used to divide a problem space into homogeneous regions, where only one or a few expert models will be run

one-cold vector: 1 × N matrix (vector) used to distinguish each word in a vocabulary from every other word in the vocabulary, where the vector consists of 1s in all cells with the exception of a single 0 in a cell used uniquely to identify the word

one-hot vector: 1 × N matrix (vector) used to distinguish each word in a vocabulary from every other word in the vocabulary, where the vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify the word

pipeline: end-to-end construct that orchestrates a flow of events and data in response to a trigger

prompt: input that an LLM responds to

prompt drift: degradation of a model's performance due to changes in the prompts that it is given

prompt engineering: process of designing a set of prompts to generate a specific output

prompt injection: attack against applications that have been built on top of AI models

prompt tuning: efficient, low-cost method of adapting an AI foundation model to new downstream tasks without retraining the model and updating its weights

prompt pipeline: pipeline that takes a user request and translates into a prompt or set of prompts

NOTE: A prompt pipeline typically uses one or more prompt templates and may also use external knowledge.

prompt template: pre-defined structure that can be used to generate text

prompting, chain-of-thought: concatenating a series of prompts that serve as intermediate steps to achieve desired behaviour

prompting, few-shot: prompting an LLM with a small number of examples of expected behaviour

prompting, skills-in-context: instructs an LLM how to compose basic skills to resolve more complex problems

prompting, tree-of-thought: prompting an LLM with a starting point and then asking it to explore different possible outcomes or conclusions that serve as intermediate steps toward problem solving

prompting, zero-shot: prompting an LLM without any examples of expected behaviour

retrieval augmented generation: framework for improving the performance of LLMs by providing them with access to external knowledge sources

reinforcement learning: use of software agents to take actions in an environment in order to maximize a cumulative reward

reinforcement learning with human feedback: trains the reward model in reinforcement learning from human feedback

transfer learning: mechanism where knowledge learned from one task is reused to improve performance on a related task

transformer: deep learning transduction model that utilizes attention, weighing the influence of different parts of the input data

transpiler: translates one language into an equivalent language

NOTE: A transpiler works at a high level of abstraction. More specifically, while it may ultimately produce source code, that code is human-readable and cannot be directly executed (it requires its own compiler).

vector database: type of database that stores data as vectors

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AI	Artificial Intelligence
AIIMS	All India Institutes of Medical Sciences
ANN	Artificial Neural Network
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BoW	Bag of Words
BSS	Business Support System
CNN	Convolutional Neural Network
CoT	Chain-of-Thought
DPR	Dense Passage Retrieval
DSL	Domain Specific Language
EBNF	Extended Backus-Naur Fom
EBNF	Extended Backus-Naur Form
FAISS	Facebook AI Similarity Search
FB	Functional Block
FLAN	Finetuned LAnguage Net
GoT	Graph-of-Thought
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LLM	Large Language Model
LM	Language Model
LSTM	Long Short-Term Memory
MedQA	Medical Question Answering
MoE	Mixture of Experts
NEET	National Eligibility cum Entrance Test
NLP	Natural Language Processing
OSS	Operational Support System
PHP	Hypertext Preprocessor
PPO	Proximal Policy Optimization
RAG	Retrieval Augmented Generation
ReLU	Rectified Linear Unit
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
SiC	Skills-in-Context
SOP	Sentence-Order Prediction
ToT	Tree-of-Thought
TPU	Tensor Processing Unit
TRPO	Trust Region Policy Optimization
USMLE	United States Medical Licensing Examination
VLM	Visual Language Model

4 Natural Language Processing with AI

4.1 Introduction (informative)

Natural Language Processing (NLP) is a branch of machine learning that enables machines to "understand" human language. A combination of linguistics, computer science, and machine learning, NLP works to transform regular spoken or written language into a form that can be processed by machines.

The purpose of this clause is to explain why a Transformer Architecture is preferred to other approaches.

4.2 Problems with Natural Language Processing (informative)

There are a number of fundamental problems in processing natural language. The most obvious first problem is that the amount of text to be processed is *variable*. A linear algebra model cannot deal with vectors with varying dimensions. This means that vector processing is recommended to be used. However, the size of the vectors grow as the size of the text grows (see clause 4.3.1 for a naïve solution and its attendant problems).

Word order is critical, as the ordering often provides important semantics. This causes the size of the vectors used to increase. Vector space model or term vector model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers (such as index terms). It is used in information filtering, information retrieval, indexing and relevancy rankings. Its first use was in the SMART Information Retrieval System.

Both of these problems dictate an increased computational cost as size increases.

4.3 Background and Previous Work (informative)

4.3.1 Bag of Words Model

The Bag of Words (BoW) model represents the input text as a vector. The size of the vector is the size of the number of terms in the text. Hence, most of the vector elements will be zeros. To minimize the size of the vector for computation, only the positions of the presented terms are stored.

Each element denotes the normalized number of occurrence of a term in the present document. BoW uses *exact* term matching to count the number of occurrences of each term.

For example, given the two sentences:

John likes to read books. Kim also likes books.
Kim also likes to watch game shows.

The corresponding BoW models are:

BoW1 = {"John":1,"likes":2,"to":1,"read":1,"books":2,"Kim":1,"also":1};

BoW2 = {"Kim":1,"also":1,"likes":1,"to":1,"watch":1,"game":1,"shows":1}.

The Bag-of-words model is an *orderless* document representation. The only thing that matters is the counts of terms. This means that simple patterns present in the text cannot be found. For example, note that in all three sentences, the verb "likes" always follows a person's name in this text.

Another critical problem with the BoW model is that it ignores the ordering of the words. For example: "Work to live" is different from "Live to Work." To keep the data order, it is necessary to increase the dimension of the graph (n-gram) to add the order into our equation.

4.3.2 n-Gram Model

The n-gram model is an alternative to the BoW model. This model is a contiguous sequence of n items from a given sample of text. The items can be phonemes, syllables, letters, words or base pairs according to the application Applying to the same example above, a bigram model will parse the text from the first sentence into the following units and store the term frequency of each unit as before.

```
[
   "John likes",
   "likes to",
   "to read",
   "read books",
   "Kim also",
  "also likes",
   "likes books",
]
```

In n-gram models, the probability of a word depends on the (n-1) previous comments, which means that the model will not correlate with words earlier than (n-1). To overcome that, n is increased, which increases the computational complexity exponentially.

While variations of both the BoW and n-gram models exist, none provide enough improvements to make them competitive with other models below (especially transformers).

4.3.3 Recurrent Neural Networks Model

The Recurrent Neural Network (RNN) is the same as the n-gram model, except that the output of the current input will depend on the output of all of the previous computations. More specifically, an RNN has connections between nodes that form a directed or undirected graph along a temporal sequence (e.g. the ingesting of the characters of a word one character at a time), enabling it to exhibit temporal dynamic behaviour. This is shown in Figures 4.3.3-1a and 4.3.3-1b.

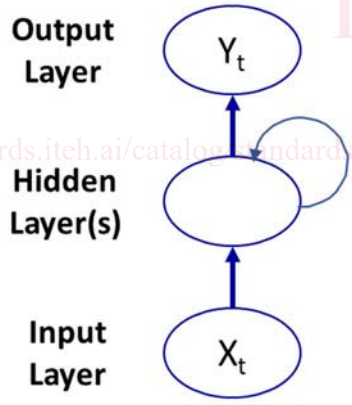

Figure 4.3.3-1a: Rolled RNN

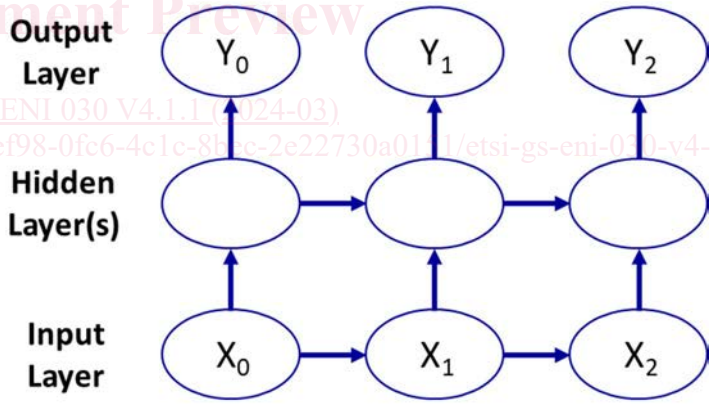

Figure 4.3.3-1b: Unrolled RNN

Figure 4.3.3-1a is a shorthand version of Figure 4.3.3-1b. The term "rolled" means that the structure is collapsed upon itself. Given a sentence, the output of the Rolled RNN is the predicted output, while the Unrolled RNN shows each individual step in determining the predicted output. The RNN handles a variable-length sequence by having a recurrent hidden state whose activation at each time is dependent on that of the previous time.

The key element of the RNN is the hidden layer, which is used to remember some information about a sequence. This enables the RNN to predict the next word of a sentence by examining the previous words that it found. This also makes the RNN fundamentally different from other artificial neural networks (which work in a linear fashion in both the feedforward and back-propagation processes), since the RNN follows a recurrent relation and uses back-propagation through time to learn (i.e. the errors at each time step are calculated to update the weights).

A neural network is a series of nodes, or neurons. The weight is the parameter within a neural network that transforms input data within the network's hidden layers. A weight can be thought of as the strength of the connection, and affects how much influence a change in the input has on the output. There is also a bias, which represents how far off the predictions are from their intended value. Within each node is a set of inputs, weight, and a bias value. As an input enters the node, it gets multiplied by a weight value and the resulting output is either observed, or passed to the next layer in the neural network. Often the weights of a neural network are contained within the hidden layers of the network. The weights and biases are learnable parameters, and are typically randomized before training begins. RNNs apply weights to the current and to the previous input. Furthermore, an RNN also adjusts the weights through gradient descent and backpropagation through time. Back-propagation fine tunes the weights of a neural net based on the error rate (i.e. loss) obtained in the previous iteration. Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

Learnable parameters in the RNN are shared in each layer in order to create a common function that can be applied at all time steps. Parameters are used to train the model. At each time step, the loss is computing and is backpropagated through the gradient descent algorithm.

The RNN consists of multiple fixed activation function units, one for each time step. Each unit has an internal hidden state, which holds the past knowledge of the network currently at a given time step. The hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation at each time step:

$$h(t) = f(x(t) + h(t-1) + b_h)$$

where h(t) is the new hidden state, h(t-1) is the old hidden state, x(t) is the current input, b_h is the bias parameter, and f() is a fixed function with trainable weights.

The activation function (i.e. a function that determines whether a neuron will be activated) uses the traditional tanh function applied to the sum of the weight times the recurrent neuron plus the weight times the input neuron. The output function is the weight of the output layer times the current state. Figure 4.3.3-2 can be simplified to emphasize their repeating structure using a single activation function, as it is done below.

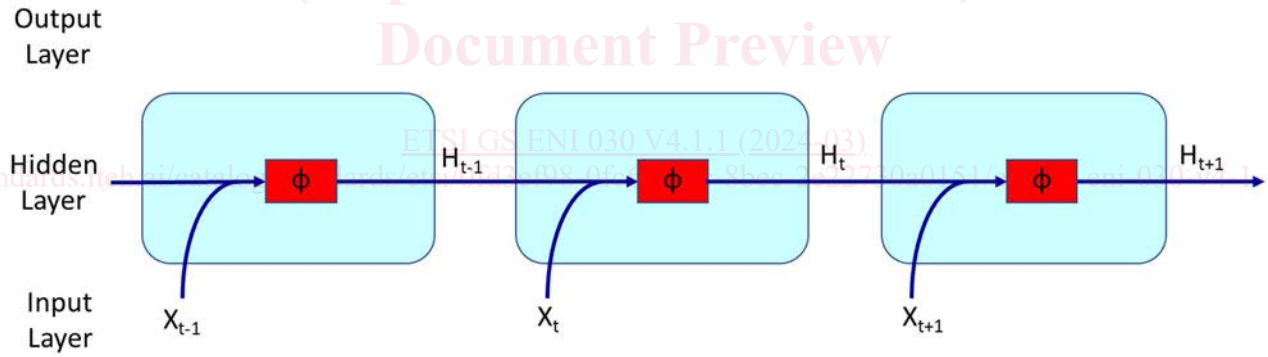

Figure 4.3.3-2: Repeating Cell in a Standard RNN

Models that are trained with gradient descent and backpropagation are subject to two problems:

1) Vanishing gradients occurs when the gradient becomes too small. When the gradient continues to become smaller, the earlier layers in the network will learn more slowly than later layers. This causes the weight parameters to continue to update until they become insignificant, which results in an algorithm that is no longer learning.

2) Exploding gradients occur when the gradient becomes too large. In this case, the model weights will grow too large, and they will eventually be represented as undefined.

Actual RNNs are not constrained to have the same number of inputs and outputs. For example, RNNs can map many inputs to one or many outputs. Bidirectional recurrent neural networks are another a variant of RNNs. Unidirectional RNNs can only use previous inputs to make predictions about the current state, but bidirectional RNNs can also use future data to improve their accuracy.

4.3.4 Convolutional Neural Networks Model

A Convolutional Neural Network (CNN) is a Deep Learning algorithm that takes an input, assign importance (learnable weights and biases) to various aspects/objects in the input, and be able to differentiate one from the other. It specializes in processing data that has a grid-like topology, such as an image. The pre-processing required in a CNN is much lower compared to other classification algorithms. A CNN can determine spatial and temporal dependencies in the input through the application of relevant filters. A convolution is the application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.

CNNs have the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modelling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images. The architecture of a CNN is analogous to that of the connectivity pattern of neurons in the human brain. Individual neurons respond to stimuli only in a restricted region of the visual field; a collection of such fields overlap to cover the entire visual area. A typical CNN has three main layers: convolutional, pooling, and fully connected. The convolutional layer uses filters that perform convolution operations as it scans the input with a filter, building up a feature map. The pooling layer is typically applied after the convolution layer, and applies a particular function (e.g. minimum, average or maximum) of its current view. The fully connected layer connects each input to all of its neurons, and is used to optimize objectives.

CNNs are not applicable to word translation due to high computational cost, but some architectures do use a convolutional function as part of their architecture.

4.3.5 Long Short Term Memory Model

Long Short-Term Memory (LSTM) was created to solve the vanishing and exploding gradient problems that RNN approaches had. LSTM focuses on modelling chronological sequences and their long-range dependencies more precisely than conventional RNNs. The main difference between an RNN and an LSTM architecture is that the LSTM's hidden layer is a gated unit. The problematic issues of vanishing gradients is solved through LSTM because it keeps the gradients steep enough, which keeps the training relatively short and the accuracy high.

A simplified block diagram of an LSTM cell is shown in Figure 4.3.5-1.

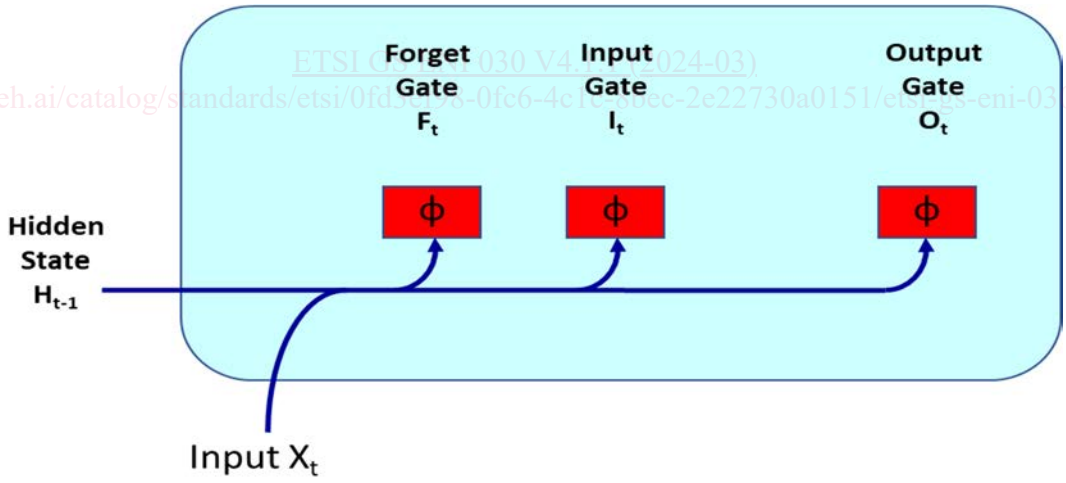

Figure 4.3.5-1: Basic LSTM Cell Architecture

The three parts of an LSTM cell shown in Figure 4.3.5-1 are known as gates. The first part is called the Forget gate, and chooses whether the information coming from the previous timestamp is to be remembered or can be forgotten. The second part is called the Input gate, and learns new information from the input to this cell. The last part is called the Output gate, and sends the updated information from the current timestamp to the next timestamp. The gate calculations are computed as follows:

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$F_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$