 doi:10.20944/preprints202204.0074.v1

Introduction to the Artificial Intelligence that can be applied to the Network Automation Journey

Alexandre GONZALVEZ[1] *alexandre.gonzalvez@nxo.eu*
Gilbert MOISIO[2] *gilbert.moisio@nxo.eu*
Noam ZEITOUN[3] *noam.zeitoun@nxo.eu*

Keywords: NetDevOps; NetOps; Intent-Based Networking; artificial intelligence; neural network; Natural Language Processing; transformer

Abstract

The computer network world is changing and the NetDevOps approach has brought the dynamics of applications and systems into the field of communication infrastructure. Businesses are changing and businesses are faced with difficulties related to the diversity of hardware and software that make up those infrastructures. The "Intent-Based Networking - Concepts and Definitions" document describes the different parts of the ecosystem that could be involved in NetDevOps. The recognize, generate intent, translate and refine features need a new way to implement algorithms. This is where artificial intelligence comes in.

1 Introduction

For several years, much research has been carried out in order to understand and predict the future. Those researches are generally based on statistical techniques. But there is now a new challenger to traditional statistical models: neural network models. The idea behind it is to give a computer a lot of examples of inputs and outputs, and then hope that the computer can find a way to relate the two in a meaningful way and generalizes the pattern. The way it learns a meaningful relationship is done through a series of connections between neurons. Each of these connections has a weight which represents the importance of the connection and each neuron has a bias which is a number added to the neuron to give it a higher or lower activation. NetDevOps is in the process of having an IETF standard that describes the concept of Intent-Based Networking, in which the relationship between the User Space part and the Intent-Based System Space part is being explored. Artificial Intelligence can help to solve problems that would require very long algorithms weighed down by long test suites.

[1]Computer Engineering Student, INSA Toulouse.
[2]Network & Methodology, Senior Consultant.
[3]Project Expert, NetDevOps.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

2 The Perceptron and activation functions

The perceptron (or a neuron) is a fundamental particle of neural networks. It works on the principle of thresholding. Let $f(x)$ be a summation function with a threshold of 40.

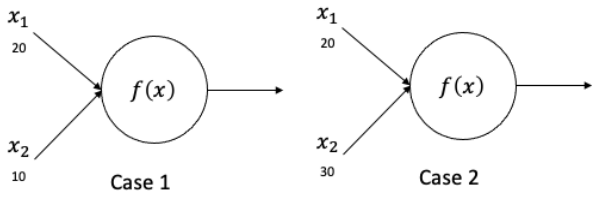

Figure 1: Firing up the neuron.

In both cases, the defined function returns the addition of two inputs, x_1 and x_2. In case 1, the function returns 30 which is less than the threshold value. In case 2, the function returns 50 which is above the threshold and the neuron will fire. Now this function becomes more complicated than that. A neuron in a typical neural network receives a sum of input values multiplied by its weights. Then we add the bias, and the function, also known as the activation function or step function, helps to make a decision.

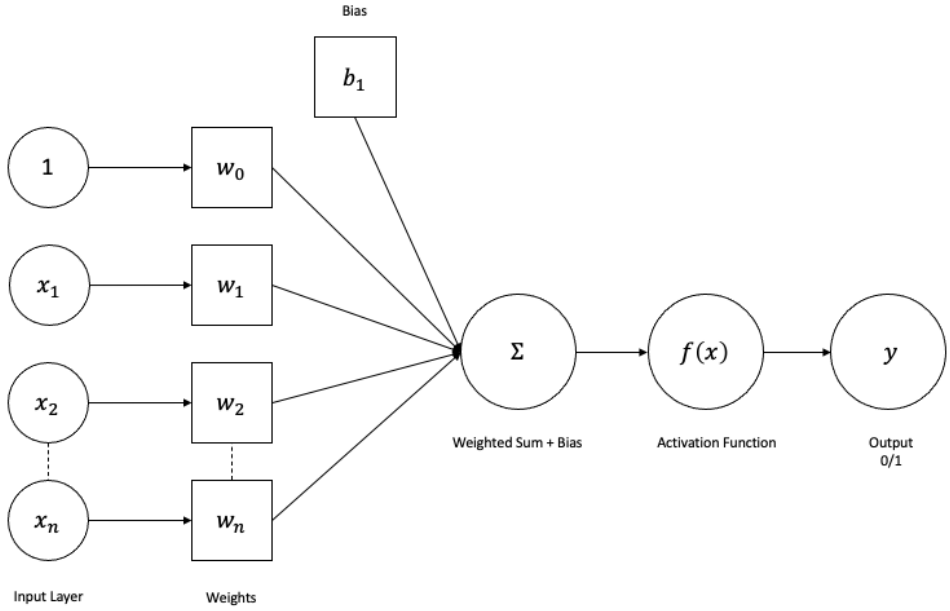

Figure 2: Perceptron.

Activation functions convert the node's output into a binary output; 1 if the weighted input exceeds the threshold, 0 otherwise (depends on the activation function). There are three best known activation functions:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

2.1 Sigmoid and Hyperbolic Tangent (Tanh)

Sigmoid is a widely used activation function that helps capture non-linear relationships.

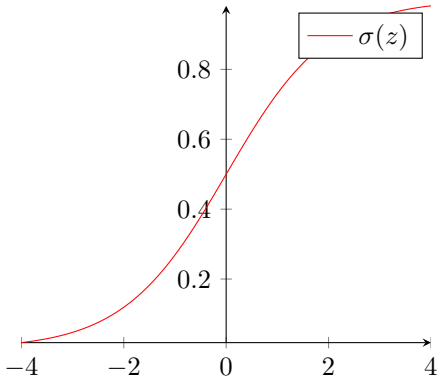

Figure 3: Sigmoid curve.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

For any value of z, if the input to the function is either a very large negative number or a very large positive number, the function $\sigma(z)$ will always return *0* or *1* as an output. For this reason, it is widely used in probability-based questions.

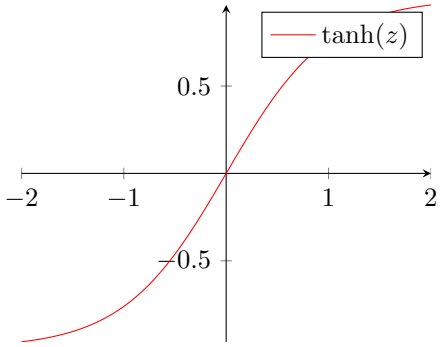

Figure 4: Hyperbolic tangent curve.

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

The Hyperbolic Tangent looks more or less like the sigmoid function but tanh varies from -1 to 1, which makes it suitable for classification problems. It is non linear.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

2.2 Rectified Linear Unit (ReLU) and Gaussian Error Linear Unit (GELU)

It is the most used activation function in deep learning because it is less complex than other activation functions, nonetheless efficient. $ReLu(z)$ returns 0 or z .

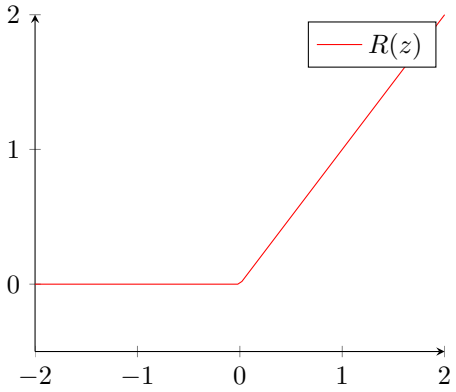

Figure 5: ReLu curve.

$$R(z) = \max(0, z) = \begin{cases} x_i, & \text{if } x < 0 \\ 0, & \text{if } x > 0 \end{cases}$$

This makes the calculation easier because the derivative of the function $R(z)$ returns 0 or 1. The GELU [1] is an activation function used in *Google's BERT* (described in 4.2.3) and *OpenAI's GPT-2*. It is only 6 years old (2016), but receives just recently any interest. This activation function can be written as an equation as follows:

$$GELU(z) = z \cdot P(X \leq z) = z \cdot \phi(z) = x \cdot \frac{1}{2}\left[1 + erf(x/\sqrt{2})\right]$$

Where $\phi(z)$ is the cumulative distribution function of the standard normal distribution.

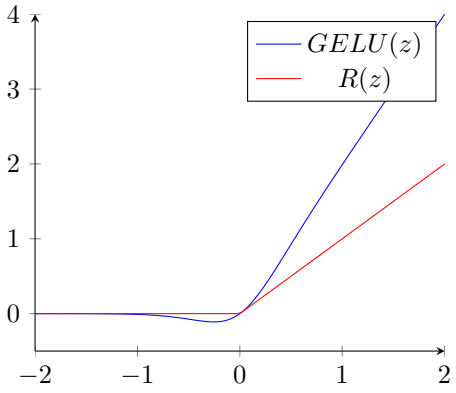

Figure 6: ReLu and GELU curves.

GELU is more interesting because it has a negative coefficient, which shifts to a *positive* coefficient. So when x is greater than *zero*, the output will be x, except when $x \in [0\ ;\ 1]$, where it slightly leans to

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

a smaller y-value. It seems to be state-of-the-art in *NLP*, specifically *Transformer models*, and avoids vanishing gradients problem.

3 Neural Network

To understand the black box of a neural network, let's consider a basic structure with 3 *layers*; an *input layer*, a *hidden layer*, connected on both sides of the neurons, and an *output layer*.

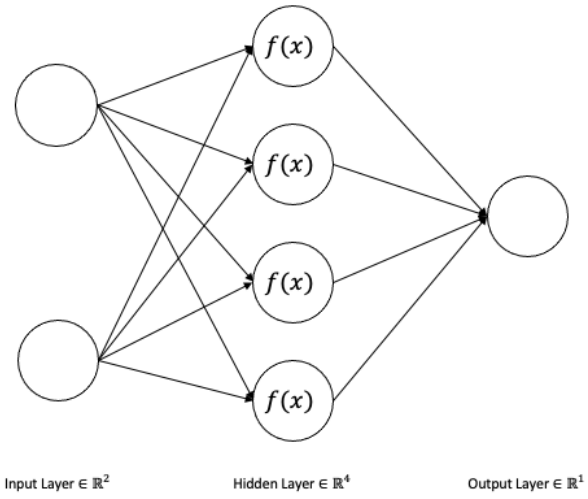

Figure 7: A simple neural network.

The weights and biases are randomly initialized. The accuracy of the output of a neural network consists in finding the optimal values for the weights and biases by continuously updating them. Consider an equation, $y = wx$ where w is the weight parameter and x is the input feature. In simple terms, the weight defines the weight given to the particular input attribute (feature). Now, the solution of the equation $y = wx$ will always pass through the origin. Therefore, an intersection is added to provide freedom to accommodate the perfect fit which is known as bias and the equation becomes $\widehat{y} = wx + b$.
Therefore, the bias allows the activation functions curve to fit up or down the axis. Now let's see how complicated a neural network can become. For our network, there are two neurons in the *input layer*, four in the *hidden layer* and one in the *output layer*. Each input value is associated with its weights and biases. The combination of input entities with weights and biases goes through the hidden layer. The network learns the entity using the activation function and it has its own weights and biases. Finally, it makes the prediction. This is the forward propagation. The number of parameters in total for our network is $((2 \times 4) + 4 + ((4 \times 1) + 1) = 17$.

For such a simple network, a total of 17 parameters are needed to optimize to get an optimal solution. As the number of hidden layers and the number of neurons in it increases, the network gains more power, but then we have an exponential number of parameters to optimize that could end up taking up a huge amount of computing resources. So there is a trade-off.

3.1 Cost Function

After a single iteration of direct propagation, the *error* is calculated by taking the squared difference between the actual output and the expected output. In a network, the inputs and activation functions are fixed. Therefore, we can modify the weights and biases to minimize the *error*. The *error* can be minimized by noticing two things: the change in *error* by changing the weights by small amounts and the direction of the change.

A simple neural network predicts a value based on the linear relationship, $\widehat{y} = wx + b$, where $\widehat{y}$ (predicted) is the approximation to y. Now, there may be several fitted linear lines $\widehat{y}$. To choose the best-fitting line, we define the cost function.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

Let $\widehat{y} = \theta_0 + x \cdot \theta_1$. We need to find values of θ_0 and θ_1 such that $\widehat{y}$ is as close to y . To do this, we need to find values of θ_0 and θ_1 such that the following defined error is minimal.

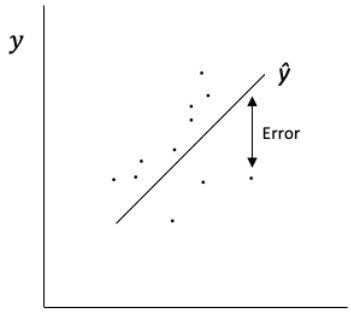

Figure 8: Best fit line.

The *error* is the squared difference between the actual value and the predicted value which is $E = (\widehat{y} - y)^2$. Therefore, we express the cost C with this equation

$$C = (1/2n)(\theta_0 + x \cdot \theta_1 - y)^2$$

where n is the total number of points to calculate the root mean square difference and it is divided by 2 to reduce the mathematical calculation. Therefore, we need to minimize this cost function.

3.2 Gradient Descent

This is the algorithm that helps find the best values for θ_0 and θ_1 by minimizing the cost function. For an analytical solution and starting from $C = (1/2n)(\theta_0 + x \cdot \theta_1 - y)^2$, we take a partial differentiation of C with respect to the variables θ_n, known as *Gradients*.

$$\frac{\partial C}{\partial \theta} = \frac{1}{n} \sum (\theta_0 + \theta_1 \cdot x - y),$$
$$\frac{\partial C}{\partial \theta} = \frac{1}{n} \sum (\theta_0 + \theta_1 \cdot x - y) \cdot x.$$

These gradients represent the slope. Now the original cost function is quadratic. Thus, the graph will look like this:

The equation to update θ is:

$$\theta^\beta = \theta^\alpha - \eta \cdot \frac{\partial C}{\partial \theta},$$

where θ^α is the old one and θ^β the new one.
If we are at point $\rho 1$, the slope is negative which makes the gradient negative and the whole equation positive. Therefore, the point goes down in a positive direction until it reaches the minima. Similarly, if we are at point $\rho 2$, the slope is positive which makes the gradient positive, and the whole equation negative moving $\rho 2$ in a negative direction until it reaches the minima. Here, η is the rate at which a point moves to minima known as the learning rate. All θ are updated simultaneously and the error is calculated. Following this, we may encounter two potential problems:

1. When updating the values of θ, you may get stuck at local minima. A possible solution is to use *Stochastic Gradient Descent* (SGD) with momentum that helps to cross local minima;

2. If η is too small, it will take too long to converge. Alternatively, if η is too large, or even moderately large, it will continue to oscillate around the minima and never converge. Therefore, we cannot use the same learning rate for all parameters. To handle this, we can program a routine that adjusts the value of η as the gradient moves toward the minima.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

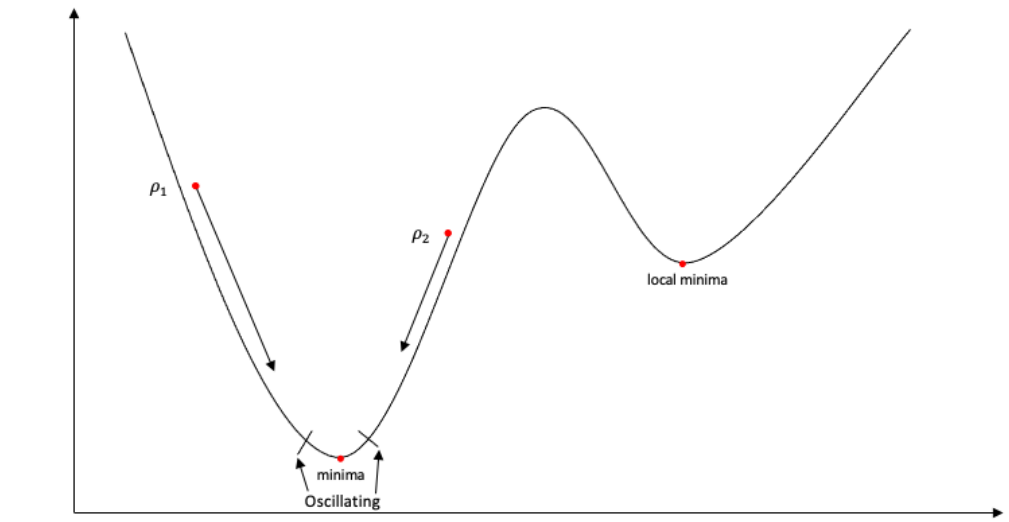

Figure 9: Gradient descent curve.

3.3 Backpropagation

The *backpropagation* is a series of operations that optimize and update the weights and biases in a neural network using the *Gradient Descent* algorithm. Consider a simple neural network (*Figure 2.*) with one input, a single hidden layer and one output.
Let, x be an input, h be a hidden layer, σ be a *Sigmoid* activation, w be weights, b be a bias, w_1 be input weights, w_0 be output weights, b_1 be an input bias, b_0 be an output bias, o be an output, E be an error, and μ be the linear transformation $(\sum w_1 x_1) + b$.
Now we create the computational graph of the *Figure 2* by stacking the series of operations needed to reach from the input to the output.

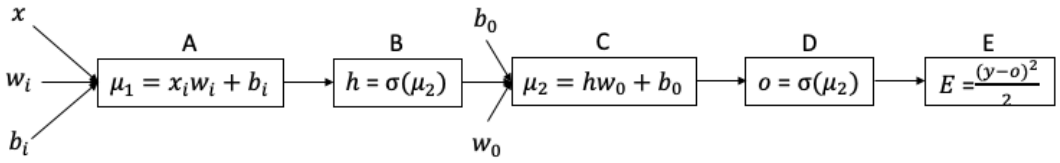

Figure 10: Computational graph.

Here, E depends on o, o depends on μ_2, μ_2 depends on b_0, w_0 and h, h depends on μ_1 and μ_1 depends on x, w_1 and b_1. We need to calculate the intermediate changes with respect to weights and biases. Since there is only one hidden layer, there are input and output weights and biases. So we can divide it into two distinct cases.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

3.3.1 Case 1: Output weight and output bias.

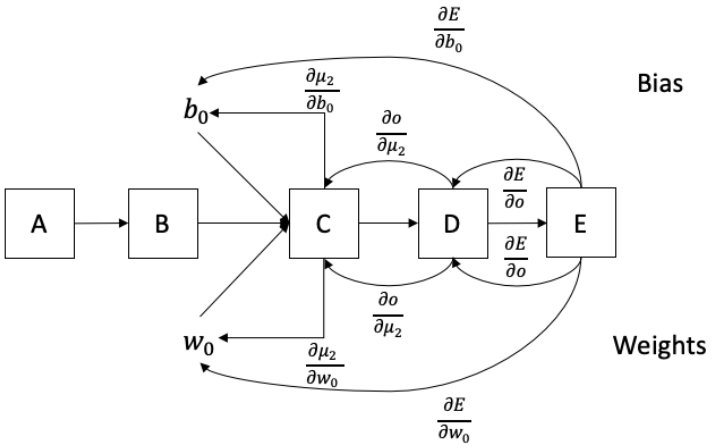

Figure 11: Computational graph for case 1.

By applying chain rule, following the weights path we have:

$$\frac{\partial E}{\partial w_0} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial \mu_2}\frac{\partial \mu_2}{\partial w_0}$$

Following the bias path we have:

$$\frac{\partial E}{\partial b_0} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial \mu_2}\frac{\partial \mu_2}{\partial b_0}$$

Therefore,

$$\begin{aligned}
\frac{\partial E}{\partial b_0} &= \frac{2}{2}\cdot(y-o) \\
&= (y-o), \\
\frac{\partial o}{\partial \mu_2} &= \sigma(\mu_2)\cdot(1-\sigma(\mu_2)) \\
&= o\cdot(1-o), \\
\frac{\partial \mu_2}{\partial w_0} &= h, \text{ As } b_0 \text{ is constant, its derivative is 0,} \\
\frac{\partial \mu_2}{\partial b_0} &= 1
\end{aligned}$$

Where $o(1-o)$ is the derivative of *Sigmoid*. Thus, by putting the values of the derivatives in the two change equations above by mistake, we obtain gradients as follows:

$$\begin{aligned}
\frac{\partial E}{\partial w_0} &= (y-o)\cdot o\cdot(1-o)\cdot h, \\
\frac{\partial E}{\partial b_0} &= (y-o)\cdot o\cdot(1-o)\cdot 1
\end{aligned}$$

And we can update the weights and biases by the following equation:

$$\begin{aligned}
\partial w_0 &= w_0 - \eta\cdot\frac{\partial E}{\partial w_0}, \\
\partial b_0 &= b_0 - \eta\cdot\frac{\partial E}{\partial b_0}.
\end{aligned}$$

This calculation concerns the hidden layer and the output. Similarly, for the input and hidden layer, it is as follows with the *Case 2*.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

3.3.2 Case 2: Input weight and input bias.

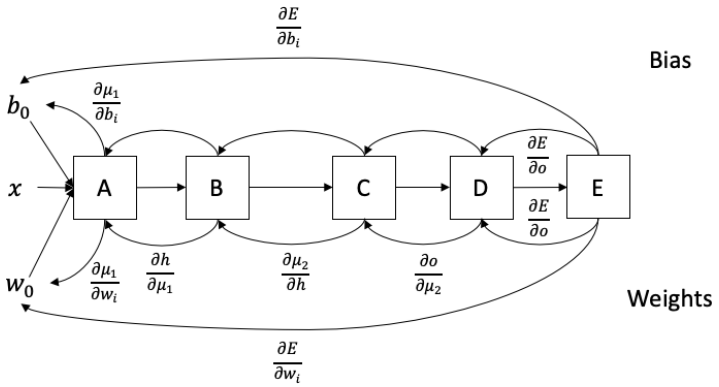

Figure 12: Computational graph for case 2.

By applying the chain rule, following respectively the path of the weights (1) and the bias (2) we have :

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial \mu_2}\frac{\partial \mu_2}{\partial h}\frac{\partial h}{\partial \mu_1}\frac{\partial \mu_1}{\partial w_i} \tag{1}$$

$$\frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial \mu_2}\frac{\partial \mu_2}{\partial h}\frac{\partial h}{\partial \mu_1}\frac{\partial \mu_1}{\partial b_i} \tag{2}$$

Now, we have:

$$\begin{aligned}
\frac{\partial \mu_2}{\partial h} &= w_0, \\
\frac{\partial h}{\partial \mu_1} &= h \cdot (1-h), \\
\frac{\partial \mu_1}{\partial w_i} &= x, \\
\frac{\partial w_i}{\partial b_i} &= 1.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= (y-o) \cdot o \cdot (1-o) \cdot w_0 \cdot h \cdot (1-h) \cdot x, \\
\frac{\partial E}{\partial b_i} &= (y-o) \cdot o \cdot (1-o) \cdot w_0 \cdot h \cdot (1-h) \cdot 1,
\end{aligned}$$

And again, we can update these gradients using:

$$\begin{aligned}
\partial w_i &= w_i - \eta \cdot \frac{\partial E}{\partial w_i}, \\
\partial b_i &= b_i - \eta \cdot \frac{\partial E}{\partial b_i}.
\end{aligned}$$

Both cases occur simultaneously and the error is calculated up to the number of repetitions called epochs. After running for a number of epochs, we have a set of optimized weights and biases for the selected features of a dataset. When new inputs to this optimized network are introduced, they are computed with the optimized values of weights and biases to obtain the maximum accuracy.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

4 Artifial Intelligence in the Intent-Based Networking

This section introduces an application of artificial intelligence in Intent-Based Networking. Concretely it will present the utilization of a transformer (deep-learning model) to translate user intent in a comprehensible format for computers.

The basics of the definition of Intent-Based Networking (IBN) were published by the research group NMRG (Network Management Research Group) of the IRTF (Internet Research Task Force). This definition [2] has evolved since 2019 and is currently at version 6, published on the 12th of December 2021.
The goal is to create a network accepting orders from users in the form of intent. This intent is a set of operational goals (that a network should meet) and outcomes (that a network is supposed to deliver), defined in a natural language without specifying how to achieve or implement them.

Intent goes through a life cycle described by Figure 13.

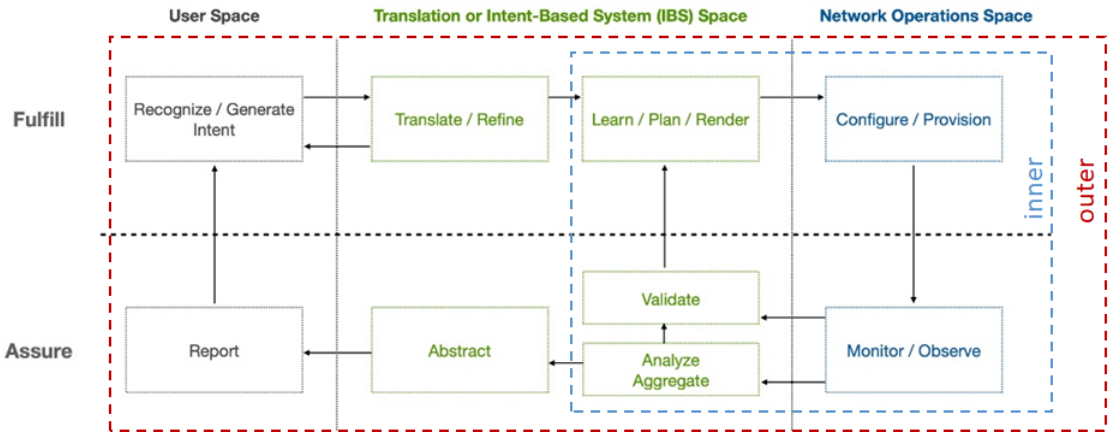

Figure 13: IBN lifecycle

This lifecycle has 2 loops:

- The inner intention control loop between Intent-Based System (IBS) and Network Operations is a completely autonomous space that does not involve any human intervention. It's a Closed-loop Automation which involves automatic analysis and validation of intent-based on observations from the network operating space.
- The outer intent control loop extends into the user space. This includes user action and adjusting their intent based on IBS observations and feedback.

One of the most challenging tasks is to make the system understand the user intent in natural language (Recognize/Generate Intent box). Understanding natural language is a complex problem that includes the meaning of the words, sentence structure, meaning of sentences, and context. This problem involves NLP (Natural Language Processing) a subfield of computer science, linguistics, and artificial intelligence.

According to the current state, a common method to solve this problem is to divide it into two parts:

1. Information extraction: Extract and label entities from the users input.
2. Intent assembly: Use extracted information to recreate intent in comprehensive form for the system.

In this paper, we will focus on information extraction. This requires part-of-speech tagging and named-entity recognition. These two NLP components can benefit from artificial intelligence progress to outperform the classical approach (ruled-based).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

4.1 Introduction to NLP

This section presents some helpful NLP concepts to extract information from user's requests.

4.1.1 Structure of NLP document

Commonly in NLP, a document is converted into an array of sentences. Each sentence contains an array of tokens. A token is a sequence of characters that are grouped as a useful semantic unit for processing (ex: word, number, dates, acronym, punctuation). Spans are similar to tokens, in that they are a piece of a Doc container. Spans have one distinguishing feature: they can cross successive tokens. Spans can also be classified into SpanGroups. Figure 14 shows this hierarchy.

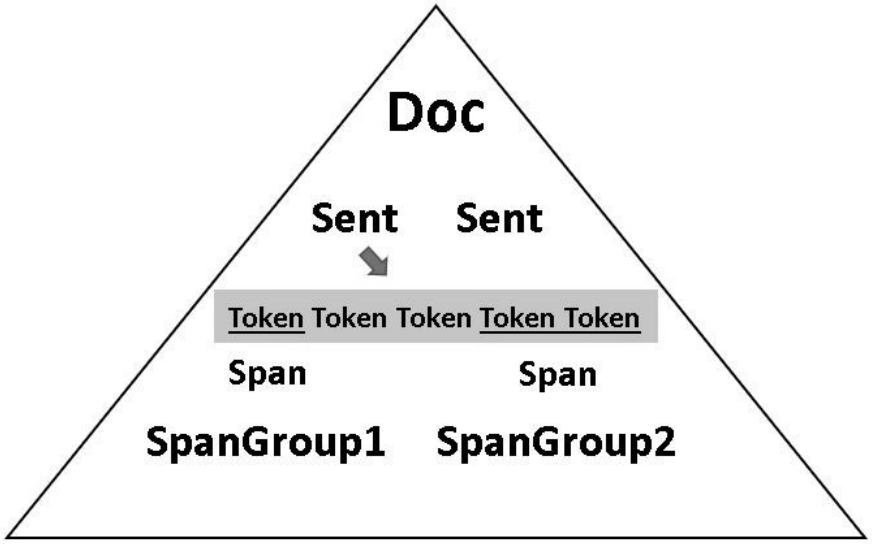

Figure 14: The classic architecture of an NLP document

4.1.2 Part-of-speech tagging and named-entity recognition

Analyzing human speech data deeply relies on part-of-speech tagging and named-entity recognition. Those processes link tokens and spans to established categories allowing general treatment (i.e. Intent Assembly algorithms).

Part-of-speech (POS) tagging POS tagging link each token of a text to is part of speech as shown in Table1 1.

Sentence	How	many	switches	are	up	for	more	than	2	hours	?
POS Tags	SCONJ	ADJ	NOUN	AUX	ADV	ADP	ADJ	ADP	NUM	NOUN	PUNCT

Table 1: Example of POS tagging result

There are mainly 2 ways of performing POS tagging:

- Rules-based: Create a preset list of rules for the algorithm to follow. It's almost impossible to create enough rules to match each word of the English language to its part of speech (especially taking in consideration the position in the text).
- Statistical model: A statistical approach of learning to tag based on a labeled dataset. This approach can be handled by hidden Markov model, conditional random field, (deep) neural network models, or a combination of these.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

Named-Entity Recognition (NER) NER link span to a spangroup. For example instead of identifying "Barack" and "Obama' as separated entities, NER can understand that "Barack Obama" is a single entity belonging to the spangroup *person*.

The 2 most known NER methods are the following:

- Ontology-based: An ontology is a collection of data sets containing words, terms, and their interrelation. NER can rely on this knowledge base. This technique excels at recognizing known terms and concepts but the ontology needs to be extremely detailed and require updates.
- Deep Learning: Recent token embedding techniques (using attention mechanisms to represent for each word their context) associated with deep learning techniques allow NER to recognize terms and concepts not present in the knowledge base.

4.1.3 Token embedding

To use machine learning techniques on text data each word needs to be vectorized (as Figure 15). Let ϕ be a word embedding mapping function:

$$\phi : \mathbb{W} \rightarrow \mathbb{R}^n$$

Where $\mathbb{W}$ is a set representing the word space and $\mathbb{R}^n$ is an n-dimensional vector space.

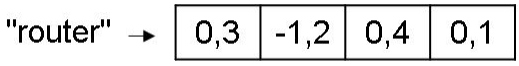

Figure 15: Example of token embedding

The first breakthrough was with the *word2vec* model [3] which represent each word by a specific vector taking into account word distance and word meaning (e.g. *cat* and *car* have close embedding considering their distance whereas *spine* and *switch* have a similar embedding because of their meaning). From there, other models surfaced with a vector contextualized meaning as with attention-based deep learning techniques.

4.2 State of this art of attention-based techniques in NLP

The context of words is essential to understand their meanin g. For example in the sentences *"Switch VLAN's configuration of each device."* and *"Show me the switch named spine1"* the word *"switch"* have a different meaning.
At first bidirectional[1] RNN (Recurrent Neural Network [4]) with attention mechanism was used to transfer all sentence information, including the relation between words[2]. However, creating a tool to transform each word into a vector with meaningful information about its context does not require RNN but only the attention mechanism.

4.2.1 The Attention Mechanism

In the paper "Attention Is All You Need" [6], they explain the Scaled Dot-Product Attention. The principle is from a matrix x (of "basic" representation of each word) to produce a new matrix A where each vector represents a word and its context.
This is done using 3 matrices of weights W^q, W^k, and W^v. These parameters are improved by backpropagation during the learning phase.
The structure of the scaled dot-product is shown in Figure 16.

[1]The context of a word is given by previous and next words, thats why a bidirectional algorithm is needed
[2]Bidirectional LSTM[5] were deeply used and are still convenient for whole text classification

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

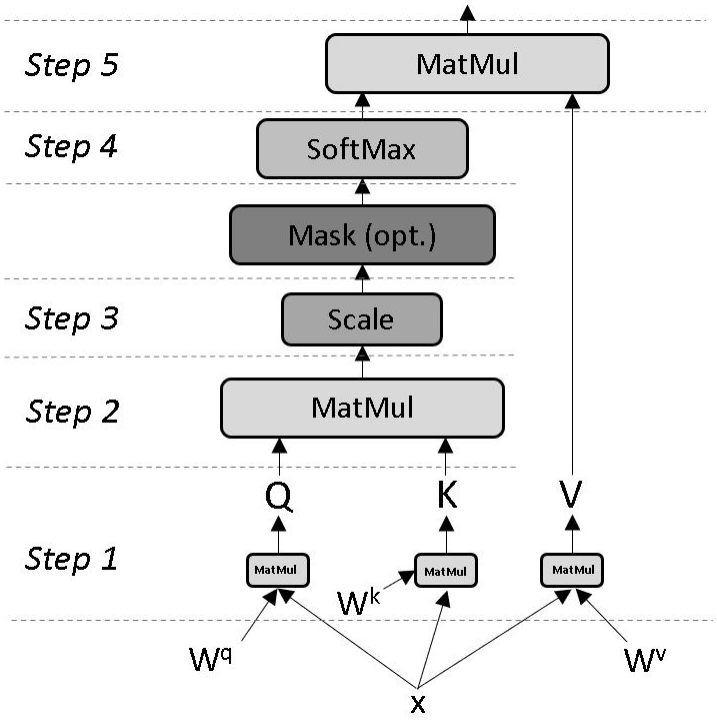

Figure 16: Scaled Dot-Product Attention

Step 1: Calculation of Q, K and V

The first step is to construct queries(Q), keys(K), and values(V). This is done by matrix multiplication between x and the weighted matrices as shown in Figure 17.

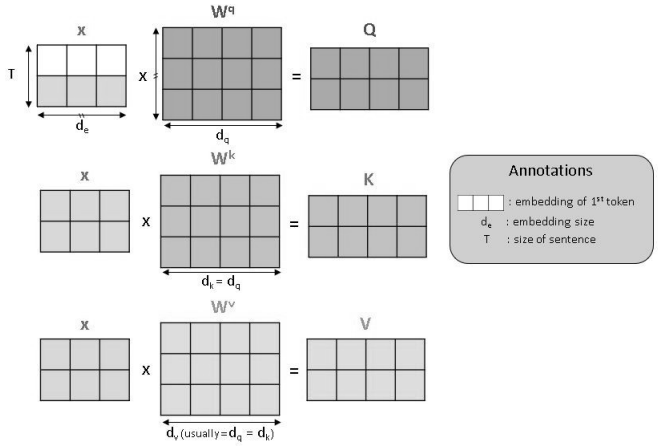

Figure 17: Calculation of Q, K and V

Step 2: MatMul of Q and K^T

Then we construct a matrix that represent the relation between words by computing Q and K^T:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

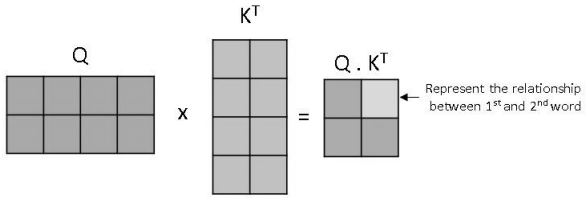

Figure 18: Matrix multiplication of Q and K^T

Step 3: Scaling

To prevent softmax from being too sharp[3], we scale the product between Q and K^T by dividing it by $\sqrt{d_k}$.

Step 4: SoftMax

The softmax is here to normalize $\mathbb{R}$ values between 0 and 1.

Step 5: MatMul of $softmax(QK^T/\sqrt{d_k})$ and V

The ending step is to multiply this relationship matrix with V to get a matrix A where each vector represents a new token embedding:

$$Attention(Q, K, V) = softmax(\frac{Q.K^T}{\sqrt{d_k}}).V$$

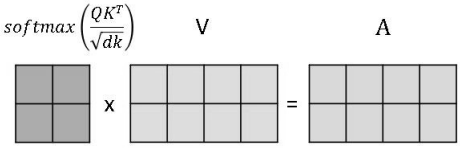

Figure 19: Matrix multiplication of $softmax(QK^T/\sqrt{d_k})$ and V

This new matrix A have T lines (1 for each token) and d_v columns.

Multi-Head Attention

The previous steps are representing 1-attention head. But to intent different parts in the sequence differently, we can use several[4] heads (similar to multiple kernels channels in CNN). Each head comports its own weighted matrices $W^q_{(i)}$, $W^k_{(i)}$, and $W^v_{(i)}$. They can be computed simultaneously. Then they are concatenated. Commonly d_v, the embedding size of 1 head, is equal to $\frac{d_e}{h}$ with h the number of heads, and d_e the original embedding size. Therefore when concatenated the new embedding size is equal to the starting one. Finally, there is a linear layer to add more parameters to this model. The whole architecture of Multi-Head attention is represented in Figure 20.

[3]If values given by the softmax function are to close 0 and/or 1, the gradient descent algorithm will take more steps to converge[7]

[4]In the original paper, the model comport 8 heads

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

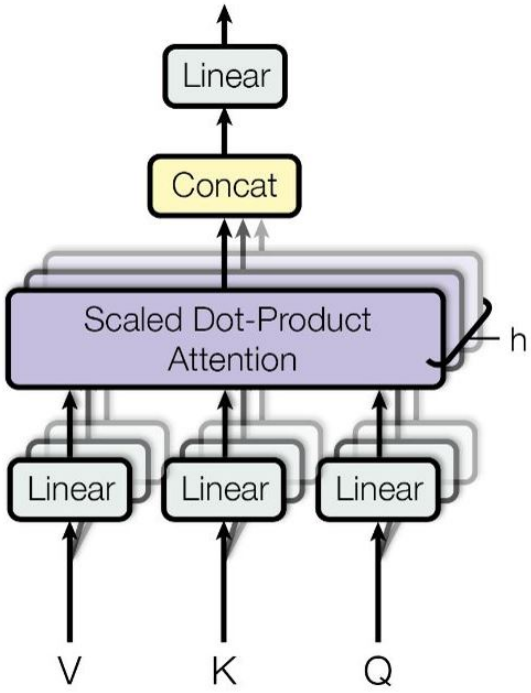

Figure 20: Multi-Head Attention - model architecture. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need.

4.2.2 The Transformer architecture

In the same paper [6], they present transformer entire architecture (Figure 21). A transformer is a self-supervised model, its structure comports 2 parts: on the one hand, the encoder takes the text input and returns a representation of that input, on the other hand, the decoder part takes the expected output value (whole text) masking some of the values (including the one our model is supposed to predict, one word at a time) and returning the output probabilities of this word. The 2 last layers (Linear and Softmax) are computationally expensive because their size corresponds to the size of our dictionary(i.e. return a probability for each word of our dictionary).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

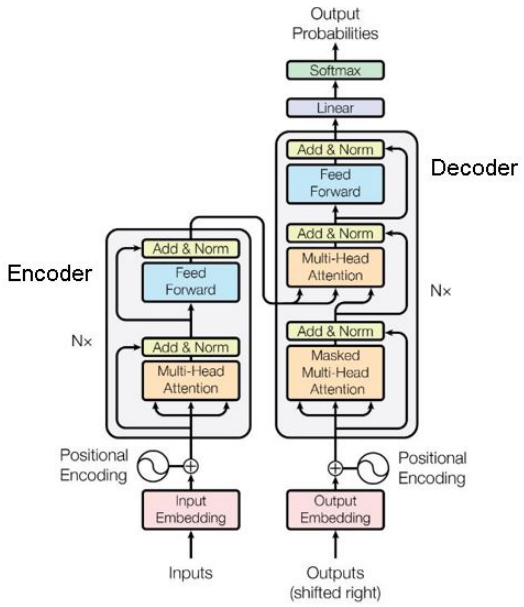

Figure 21: The Transformer - model architecture. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need.

Encoder

To begin with, some information is added through the position of each word. As our model is not an RNN, the purpose is to make the model behave slightly differently considering the position of words in the text. It won't add exact position information, but encode relative position through 2 sinusoidal functions and add that information to the input vector.
Another important behavior of this model is the skip connection part. After each layer(Multi-Head Attention layer and Feed Forward layer) there is an Add & Norm component. Which makes the sum of the output and the input: layer(x) + x with x the layer input. This passing residual information about the data before passing through the layer. Allowing the model to let information through if the layer does not learn useful things. The normalization part is a technique to normalize the distribution of intermediate layers that enables smoother gradients, faster training, and better generalization accuracy. Finally, the Feed Forward component is a multilayer perceptrons. It is a classic neural network where each neuron of layer n is connected to each neuron of the following layer n+1.

Decoder

On the decoder part, we have mostly the same components, the main difference is that we have Masked Multi-head attention. It is exactly the same principle, with the difference that it masks some words of the sentence as said previously.

Both encoder and decoder blocks are repeating N times. In the original paper, they repeat it 6 times, creating the structure represented in Figure 22.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

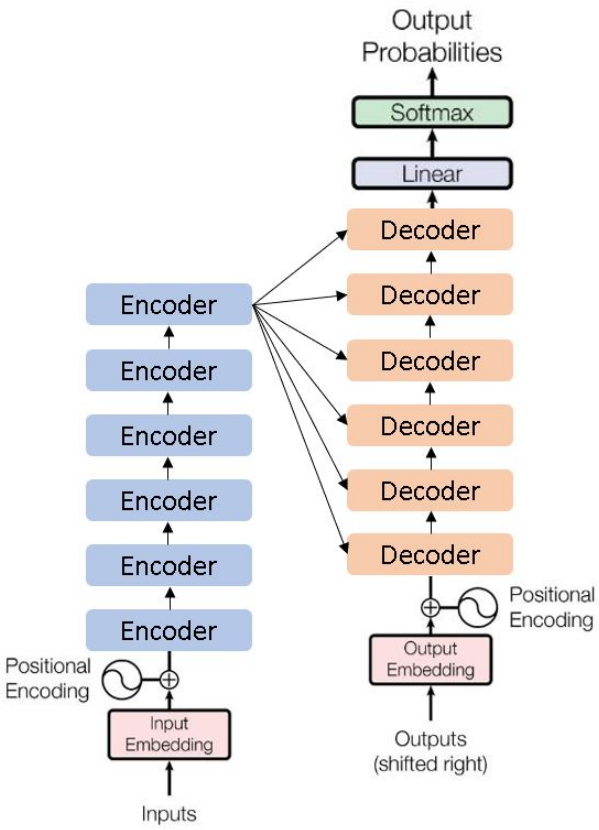

Figure 22: Computational flows with encoder and decoder repeated 6 times

4.2.3 BERT

There are several implementations of the transformer model, the most popular being BERT, GPT, and BART. Their popularity is mainly due to their performance in encoding long-range dependencies through self-attention and their self-supervision techniques for leveraging unlabeled datasets. In this section, we will focus on BERT, which stands for Bidirectional Encoder Representation from Transformer [8].

There are three parts to this model: the embedding, the encoder, and the pooler layer.

Embedding layer

The BERT model's input is an array of tokens that represent a text. The token embedding is the WordPiece embeddings [9]. This is an example of this token embedding:

"Show me Cisco routers up since a year"

$$\rightarrow E_{[CLS]} + E_{show} + E_{me} + E_{cisco} + E_{routers} + E_{up} + E_{since} + E_{a} + E_{year} + E_{[SEP]}$$

This embedding allows reducing vocabulary size with generalization as *verb* + *ing* to "only" 30.000 different tokens. But it was created for Asian languages such as Korean and Japanese. There are a huge number of characters in these languages, as well as homonyms and no or few spaces between words. The text had to be segmented because there were no or few spaces. Without segmentation, would result in a large number of out of vocabulary terms (OOV) in the model.
This embedding produces a vector of size 768.

Encoder layer

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2022 doi:10.20944/preprints202204.0074.v1

The encoder part is similar to the classic transformer model except for few specificity as:

- The BERT model uses 12 head, so each head return vector of size 64 (768/12).
- The Feed Forward layers use Gelu activation function (2.2).

Pooler layer

The pooler uses the output representation to uses it for downstream tasks(the task you want to solve with this model). This pooler contain a linear layer and a tanh activation function (2.1).

Self-supervision techniques

This pre-trained model has learn on English Wikipedia (about 2.5 billion words) and a book corpus (about 800 million words) with 2 tasks:

- Masked Language Modeling: About 15% of words are masked and the task is to retrieve them. It's a classification of N classes with N the size of the vocabulary (BERT vocabulary corresponds to WordPiece embedding (4.2.3), for comparison an adult English native speaker's vocabulary is around 20 000 - 35 000 words).
- Next Sentence Prediction: Given 2 sentences the task is to predict if the second follows the first. It's a binary classification.

This pre-training allows the parameters of this model to be already efficient. These parameters are weights of the multi-head attention (each Q_i, K_i, V_i), weights, and bias of the Feed-Forward neural network.

4.3 Using BERT to perform NER

To personalize a pre-trained BERT model on a specific task there are 2 commons methods: both involve adding extra layers to perform our task (classification, translation, question/answer, etc.). The first called fine-tuning is to retrain the whole model, all parameters of the BERT model, and extra layers. Second, the feature-based approach is to only update the extra layers. The principle is that the last layers of the BERT model gave significant information about each word and can be used as input for our neural network. The first method gives slightly better results but it's much more computationally expansive.

4.3.1 Application on NER

Applying the BERT model to specific named-entity recognition requires a few preparation steps: defining the desired span groups, creating a dataset with labeled sentences. A labeled sentence means an array containing tuples with a span group and the position of the given span (position can be indicated by the indices of the first and last token of the span or indices considering sentence character length). Then we choose to re-train the BERT model considering both methods: fine-tuning or feature-based approach.

4.3.2 Continuous learning techniques based on user refinement

Once trained our model can still miss some spans because they are too different by their initial embedding or by their context. In this case, by asking the user what the model miss understood, the system can correct the NER and add this new labeled sentence to the training dataset. Therefore the model will improve itself during its utilization.

 doi:10.20944/preprints202204.0074.v1

5 Conclusion

We worked on the chapters "Intent Ingestion and Interaction with Users" and "Intent Translation" of the "Intent-Based Networking - Concepts and Definitions" document.

The first chapter of the document specifies that "The goal is ultimately to make IBSs as easy and natural to use and interact with as possible, in particular allowing human users to interact with the IBS in ways that do not involve a steep learning curve that forces the user to learn the "language" of the system. Ideally, it will be the Intent-Based Systems that is increasingly be able to learn how to understand the user as opposed to the other way round. Of course, further research will be required to make this a reality."

The second chapter indicates that "Beyond merely breaking down a higher layer of abstraction (intent) into a lower layer of abstraction (policies, device configuration), Intent Translation functions can be complemented with functions and algorithms that perform optimizations and that are able to learn and improve over time in order to result in the best outcomes, specifically in cases where multiple ways of achieving those outcomes are conceivable."

Looking at the way to answer to those two chapters, we study the way to apply Artificial Intelligence to Intent-Based Networking. We will continue to develop the concepts described is this paper in Python to finalize our Proof of Concept. All the IBN inner loop of the Intent Lifecycle, that we already developed, is currently efficient and we will work on the top part of the outer loop in order to be able to take an intent in natural language and translate it in a way understandable by the main manufacturers specialized in network automation. AI will help to create agnostic approach that a lot of people are waiting for.

 doi:10.20944/preprints202204.0074.v1

References

[1] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)." [Online]. Available: http://arxiv.org/abs/1606.08415

[2] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-based networking - concepts and definitions." [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-nmrg-ibn-concepts-definitions

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space." [Online]. Available: http://arxiv.org/abs/1301.3781

[4] R. M. Schmidt, "Recurrent neural networks (RNNs): A gentle introduction and overview." [Online]. Available: http://arxiv.org/abs/1912.05911

[5] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pp. 207–212. [Online]. Available: http://aclweb.org/anthology/P16-2034

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need." [Online]. Available: http://arxiv.org/abs/1706.03762

[7] X. Wan, "Influence of feature scaling on convergence of gradient iterative algorithm," vol. 1213, no. 3, p. 032021. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1213/3/032021

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding." [Online]. Available: http://arxiv.org/abs/1810.04805

[9] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, . Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation." [Online]. Available: http://arxiv.org/abs/1609.08144