8-11-2022

Enhancing the Cognition and Efficacy of Machine Learning Through Similarity

Vishnu Pendyala
San Jose State University, vishnu.pendyala@sjsu.edu

Rakesh Amireddy
Coherent Inc

Recommended Citation
Vishnu Pendyala and Rakesh Amireddy. "Enhancing the Cognition and Efficacy of Machine Learning Through Similarity" *SN Computer Science* (2022). https://doi.org/10.1007/s42979-022-01339-y

Enhancing the cognition and efficacy of machine learning through similarity

Vishnu Pendyala[1*] and Rakesh Amireddy[2]

[1*]Department of Applied Data Science, San Jose State University, One Washington Square, San Jose, 95192, CA, USA.
[2]Software Engineer, Skoruz Technologies, 5600 Mowry School Rd Ste 150, Newark, 94560, CA, USA.

*Corresponding author(s). E-mail(s): vishnu.pendyala@sjsu.edu; Contributing authors: amireddy.rakesh@gmail.com;

Abstract

Purpose: Similarity is a key element of machine learning and can make human learning much more effective as well. One of the goals of this paper is to expound on this aspect. We identify real-world concepts similar to hard-to-understand theories to enhance the learning experience and comprehension of a machine learning student. The second goal is to enhance the work in the current literature that uses similarity for transcoding. We uniquely try transcoding from Python to R and vice versa, something that was not attempted before, by identifying similarities in a latent embedding space. **Methods:** We list several real-world analogies to show similarities with and simplify the machine learning narrative. Next, we use Cross-Lingual Model Pretraining, Denoising Auto-encoding, and Back-translation to automatically identify similarities between the programming languages, Python and R and convert code in one to another. **Results:** In the course of teaching machine learning to undergraduate, graduate, and general pool of students, the first author found that relating the concepts to real-world examples listed in this paper greatly enhanced student comprehension and made the topics much more approachable despite the math and the methods involved. When it comes to transcoding, in spite of the fact that Python and R are substantially different, we obtained reasonable success measured using various evaluation metrics and methods as described in the paper. **Conclusion:** Machines and human beings predominantly learn by way of similarity, a finding that can be explored further in both the machine and human learning domains.

Keywords: Machine Learning, Similarity, Transcoding, Deep Learning

1 Introduction

Machine learning takes inspiration predominantly from human learning. Human learning can take inspiration from how machines learn as well. Recent research [1] proved that every machine learning algorithm that uses gradient descent and that includes deep learning, is essentially a kernel machine, where the kernel function measures the similarity between data points. In this paper, we extend [2] and explore similarity in both human learning of the subject of machine learning and in machine learning itself. The former is done through a series of analogies as in [2] and the latter, by making machines learn to translate from one programming language to another by discovering similarities between the languages through a shared embedding space. The translation follows the methodology described in the existing literature [3], but for a different pair of languages, Python and R.

Similarity has been substantially explored in machine learning algorithms such as in the K-nearest neighbors, Kernel methods, Support Vector Machines and can be used to simplify the machine learning narrative as well. Human beings and machines learn by analogy. Human beings relate new knowledge to what they already know to help in the assimilation of the new knowledge. A detailed survey on the role of analogies in the learning process [4] supports this assertion. It is not possible to easily comprehend an abstract concept, completely new from thin air if it does not resemble or relate to any known metaphor. This is similar to how machines learn to classify the test data, unseen till then, by relating it to the training data that is used to build a model. Analogies, therefore, play a critical role in comprehending complex topics. This paper relates some of the concepts, artifacts, and algorithms in machine learning such as overfitting, regularization, and Generative Adversarial Networks to the real world, all summarized in a table.

The concepts of similarity, data proximity, and nearest neighbors have found tremendous applications in machine learning. Given that the dot product of vectors inherently has the semantics of similarity and matrix multiplications are essentially a series of dot products, it can be concluded that a significant part of machine learning is inherently, learning by analogy. While classification algorithms such as K-Nearest Neighbors and Non-linear Support Vector Machines using Kernel Methods use the concept of data proximity directly, any algorithm that uses the dot product or matrix multiplication is directly or indirectly leveraging the notion of similarity.

The other way deep learning models can explore similarity in data is by using a base representation space as we see in the case of automatic transcoding described in detail later in this paper. The power of the transcoder is in making code translations between Python and R without requiring parallel corpora with equivalent code snippets in Python and R for training. We follow the approach in [3] and use a sequence-to-sequence model with an attention mechanism, composed of an encoder and decoder with a transformer architecture. We used a single shared model for all languages and trained it using three unsupervised machine translations, namely, Cross-Lingual Model Pretraining, Denoising Auto-encoding, and Back-translation to achieve higher accuracy. As can be seen from the following sections, the design and implementation is substantially complex, involving multiple techniques. The focus of this paper is to leverage similarity in both human and machine learning. The former is done by drawing parallels between real-world and machine learning concepts in section 3, while the latter is done in section 4 by demonstrating transcoding which also uses similarity.

2 Related Work

We survey the literature for related work in both the domains as described below

2.1 Real-world analogies of machine learning concepts

The literature survey to discover analogies related research was carried out by using search queries such as ‘teaching ”machine learning” ”real world”,’ ‘understanding ”machine learning”’ and ‘”machine learning” analogy ”real world”’. The search results include textbooks such as [5], which explain how machine learning can be applied to the real world, but not how the machine learning concepts are similar to the real-world notions, indicating that the work in this paper is unique and novel. In an extensive 124-page write-up, Mehta et al [6] draw some parallels between Physics and machine learning to explain various concepts. Using what they call a “physics-inspired pedagogical approach,” they point out that similar to Physics, machine learning emphasizes empirical results and intuition. They compare the cost function to “energy,” some of the steps in Stochastic Gradient Descent to the momentum-based methods, the pooling step in Convolutional Neural Networks to the decimation step of Renormalization Group (RG).

The human brain and machine learning algorithms both take high-dimensional data as input and perform classification tasks. In [7], the author touches upon multiple areas of intersection of neuroscience and machine learning when giving insights into his laboratory’s research program to decipher the algorithms in biological computations that go on in human and animal brains. Analogies accelerate the pace of innovation. Demonstrating this important hypothesis, the authors of [8] use recurrent neural networks to mine idea repositories, specifically, an online crowdsourced product innovation website, Quirky, to generate analogies. The inspiration from these analogies caused the participants in their experiments to generate better ideas. Drawing an analogy to human learning in a teacher-student setup, authors of [9] propose a machine learning framework for various DNN models that uses far lesser training data and executes faster, in fewer iterations, but still achieves similar performance.

Machine learning models, to a substantial extent, are opaque, lacking explainability. Addressing this issue, the author of [10] describes how this is a problem for socially significant applications of machine learning. The opacity not only makes it difficult to interpret the results but makes it harder for the students to get deeper insights. Relating the algorithms to real-world experiences, as we describe in this paper alleviates the comprehension difficulties. There is hardly any published research that delves into the problem of teaching machine learning effectively to any population [11]. The functioning of the human brain has inspired the development of neural networks and deep learning frameworks. The authors of [12] argue that relating infant and toddler psychology to algorithms used in computer vision such as Convolutional Neural Networks may result in newer principles of learning.

2.2 Deep learning for exploring similarity

In 2018, the research work by Lample et. all [3] from Facebook AI research developed a model that uses a shared latent space to map the monolingual corpora from two languages. Both target and source languages learn to reconstruct in both the languages by which the model can learn to translate without using any labeled data. This approach has achieved BLEU scores of 32.8 and 15.1 on English French datasets. There is ample literature on how deep learning frameworks use similarity to accomplish substantial tasks. In addition to the approach presented in this paper, the CycleGAN [13] framework has also been used to discover similarities in images [14] and even Indian Classical Music [15] and use them for equivalent conversions. For brevity, we suffice it by stating that there is no evidence of researchers attempting to translate software code from Python to R and vice versa using the approach presented in this paper.

2.3 Contribution

To the best of our knowledge, as shown in Figure 1 this is the first work in drawing a parallel between exploring similarity in human and machine learning. As illustrated in the figure, human learning involves converting concepts into latent cognitive artifacts such as impressions and thoughts to see similarities between known and new theories. In a similar fashion, deep machine learning can map programming language constructs into a shared space of latent embedding to learn similarities between them. This is the first journal paper to present real-world analogies to explain machine learning concepts. This is also the first work to use deep learning to convert programs in the Python programming language to R and vice versa. Section 3 expounds similarity in human learning and section 4 details the use of similarity in machine learning for transcoding between languages.

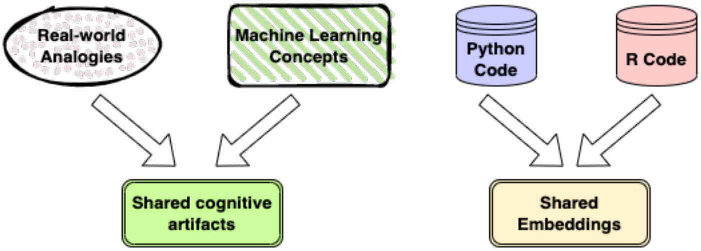


Fig. 1 How similarity plays a role in (a) human and (b) machine learning

3 Exploring Similarity in Human Learning: Machine Learning Parallels to Real World

Learning, whether in human beings or machines follows similar principles in certain respects. Machine learning algorithms are associated with considerable math that can intimidate new learners. Drawing parallels with the real world is important for improving comprehension in this area. Table 1 lists a few analogies that help simplify the intuition behind machine learning concepts, types, and algorithms. For details and explanation of the analogies, please refer to [2].

Machine Learning Concept	Real-World / Simpler Analogy	Similarity
Supervised Learning	Parents labeling what is good and bad for children	The world is already classified
Unsupervised Learning	Students forming groups without any supervision	Lack of labels
Matrix multiplication	Series of “doting” products	Similarity of two vectors/entities
Maximum Margin Classifier	Arbiter	Equidistant from either class
Artificial Neural Network	Self-starters, Achievers	Mastering the art of achieving targets
Hidden Layers in Artificial Neural Networks	Departments in an organization	Division of labor
GAN	Akinator game	Adversarial nature
Lazy Learning	Last-minute exam preparation	No preparation in advance
Boosting	Student improving exam over exam	Focus on past failures
Overfitting	Narrow-minded	Undue importance to limited artifacts
Regularization	Spirituality	Reduce weights attached to features
PCA	Caricature	Capturing the variance

Table 1 A few parallels between machine learning and the Real-World

4 Transcoding

In this section, we switch gears from the human learning domain into the machine learning domain and utilize similarity to train machines to convert code in one programming language to other.

4.1 Methodology

Deep learning algorithms are able to perform complex NLP tasks like language translation with the advent of Sequence-to- Sequence models. These were proposed in the research done by Google to solve sequence to sequence problems like translating from English to French [16]. They used Long Short-Term Memory (LSTM) neural network to map input sequences one word at time and one more LSTM to produce output sentences in target language from the input sequence based on encoder-decoder architecture. The results found that LSTM performed well on long sentences as opposed to regular Deep Neural Networks (DNNs), but it did not perform well if the input contains vocabulary words not used in the training data. Also, reversing the order of words in the input sequence improved the LSTM performance significantly. Later, Bahdanau et al. came up with Neural Machine Translation by Align and Translate [17] to improve the performance of the basic encoder-decoder architecture and to overcome problems faced especially with sentences longer than the sentences provided in the training data. This new model searches for only relevant information provided in the input sequence to predict the target word based on the context vector generated by input and the previously generated target words.

Later, a new architecture called the Transformer, was proposed by Vaswani et al. [18] based on the attention mechanism. With this new approach, the models need significantly less time in training and allows more parallel processing. Transformer uses a self-attention mechanism to generate a representation of the input sequence by relating different positions. It has been very successful in tasks like abstractive summarization, reading comprehension, etc. Self-attention layers over recurrent layers reduces the complexity of the computation per layer, improves parallel computation and finally reduces the path length between long-range dependencies. There are three principles of unsupervised machine translation, using which the pretraining is done: initialization, language modeling and back-translation.

Pretraining is considered the most important part of unsupervised machine translation according to Lample et al [19]. Encoders are neural networks which are generally used for feature extraction and feature selection. The encoders have nodes which are present in the hidden layer and at times outnumber the inputs being given to the network [20]. In such cases the learning rate of the encoder is low as it does not perceive its Identity Function, wherein the output is the same as the unmodified input [21].

A denoising encoder is the solution to the problem, whereby the input being passed to the nodes is corrupted by masking nonzero items randomly. The output obtained must be compared to the original input instead of the corrupted one to ensure that the features are extracted instead of perceiving an identity function from the data [22]. Over the past few years, Neural Machine Translation or NMT has gained its importance in translating one language to another language. According to a study in [23], the traditional NMT approach works in a semi-supervised or supervised setting where millions of data need to be labeled.

To avoid such a time and money consuming approach, the researchers He et. all [23] proposed an approach where machine translation is done in an unsupervised setting. The basis of their work is that any machine translation involves source-to-target and target-to-source translations both of which can form a close loop and produce feedback signals. These signals are used to train the translation models without requiring labelling of the data. They referred to this approach as dual-NMT. The experiments performed in this paper achieved the best performance in translating the English language into French.

Based on the above approach, the authors Artetxe et. all in the study [24] also experimented with the NMT system without the need for parallel data. Their work relies on the monolingual corpora and their model is based upon unsupervised embedding mappings slightly modified attentional encoder-decoder models. This model is trained on monolingual corpora and uses the combination of denoising and back-translation. Lample et. all [3] builds upon the foregoing research findings and we experimented with this approach to see how well it works for disparate programming languages like Python and R.

A Transformer Architecture with attention mechanism is the most advanced technique used in Natural Language Processing. Self-attention layers over recurrent layers reduce the complexity of the computation per layer, improves parallel computation and finally reduce the path length between long-range dependencies. Models using Transformer architecture need significantly less time in training and allow more parallel processing than LSTM models. Computational accuracy is used to evaluate the translation models. The Beam Search decoding algorithm and the Greedy Search algorithm are proposed to calculate the quality of the text when translating from one language to another. Beam Search is an optimization of best-first search to reduce memory requirements and make it faster for predicting next sequences. The various phases in the project are illustrated in Figure 2. The flow primarily comprises of Data collection, Preprocessing, Training, Testing and Evaluation. Each of these phases are detailed in the following paragraphs.

4.2 Dataset

Our Training data is a collection of JSON files that have R language programs that need to be converted into Python code and Python programs that are to be transcompiled to R. In the scope of this project, we only translate functions instead of complete programs from one language to another.

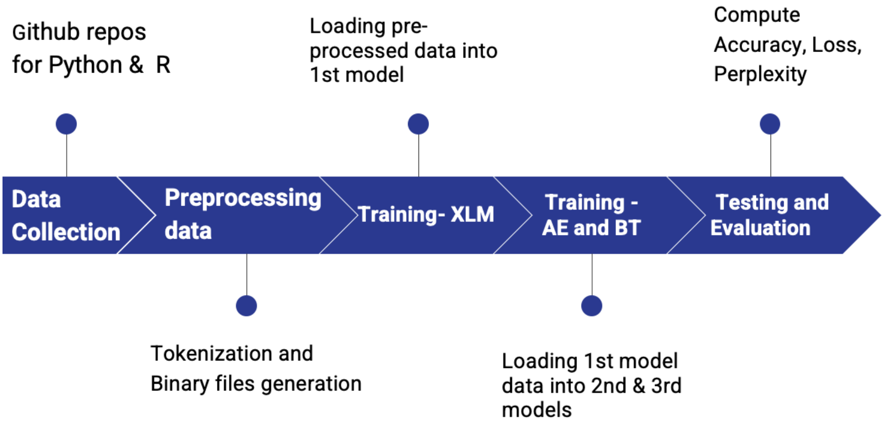


Fig. 2 Project flow

4.2.1 Data Collection

The compilation of dataset for this project is obtained from GitHub repositories through Google Big Query in Google Cloud Platform. The repositories consist of programs, functions in two programming languages, namely R and Python. The URL is given below.

https://console.cloud.google.com/marketplace/details/github/github-repos

4.2.2 Training Dataset

We prepared Python and R datasets as .json files from Google Big Query Github repositories (5000 repositories each) with repo name, path and content which contains actual source code of the programs in each line. However, due to resource constraint, we could use only 2922 program file samples for Python 788 program file samples for R from these repositories.

4.2.3 Test Dataset

Preprocessing step has the option to choose the size of the test/validation sets. Depending on the input, it divides the whole data set into training, testing and validation data sets.

For a monolingual dataset of 70 repositories, including 35 repositories for each Python and R programs, the test dataset consists of 5 repositories for each R and Python.

4.2.4 Validation Dataset

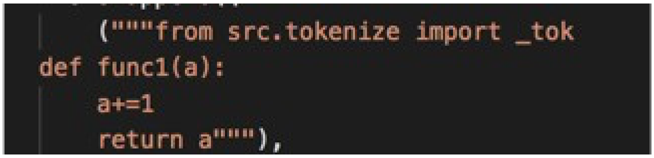
Prepared our own set of programs both in Python and R as a Validation Dataset for evaluation. This will be used to evaluate the correctness of the translations.

4.3 Preprocessing

The preprocessing pipeline extracts both R and Python source code from the training dataset JSON files and performs tokenization training and apply BPE (Byte Pair Encoding) codes to do sub word tokenization and compression, extract functions and binarizes the data to be used by XLM (Cross Lingual Models). These steps are described below.

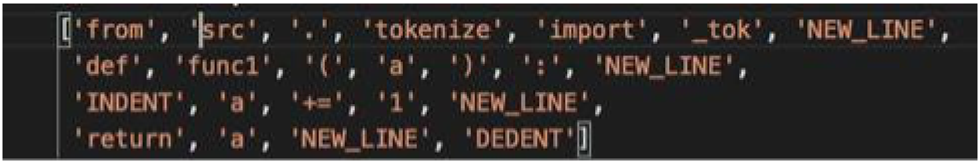
4.3.1 Tokenization

Tokenizer is used for respective programming languages to generate tokens for the input source code. The Python module, ‘tokenize’ is used to generate tokens for Python and ‘tokenizers’ library is used to generate tokens for R language and ‘PypeR’ Python library is used to access R Tokenization script from Python. In the next step, tokens are split into sub word units by learning BPE (byte pair encoding) codes. Byte Pair Encoding is a simple data compression technique that works by replacing common pairs of consecutive bytes with a byte that does not appear in that data. Figures 3 and 4 illustrate the process.



```
    ("""from src.tokenize import _tok
def func1(a):
    a+=1
    return a"""),
```

Fig. 3 A Python code snippet that needs to be translated to R



```
['from', 'src', '.', 'tokenize', 'import', '_tok', 'NEW_LINE',
 'def', 'func1', '(', 'a', ')', ':', 'NEW_LINE',
 'INDENT', 'a', '+=', '1', 'NEW_LINE',
 'return', 'a', 'NEW_LINE', 'DEDENT']
```

Fig. 4 Tokens generated from the Python code snippet

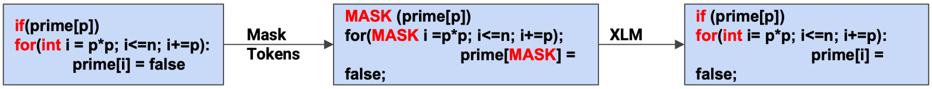
4.3.2 Function Extraction

Automatic Code Translator only works at the function level and it makes the process of the model evaluation simpler. Pretraining is done on all the source code available but denoising auto encoding and back translation models are trained only on functions. Extraction differentiates class functions and standalone functions as well. Based on the token (‘def’ in Python and function() in R) , the logic identifies and extracts functions from the source code.

4.3.3 Cross Programming Language Model Pretraining

Pretraining is a method of using uncategorized data to aid a Natural Language Processing (NLP) model performing a task. Recent studies show pretraining to be an effective and significant practice in cultivating results. Researchers of Multilingual or Multilingual language pretraining are pioneers who demonstrated how a new approach in Multilingual pretraining can benefit and produce better results as compared to generative pretraining. This research was a part of the Facebook Artificial Intelligence (A.I.) team and extended to multiple languages other than English. Their technique delivers a significant improvement over the previous work in both sub-categories of Machine Translation; supervised and unsupervised, as well as in multilingual text classification of languages that have sparse resources.

We use cross-lingual masked language model for pretraining. The idea is similar to how students are trained to recognize patterns by assigning them fill-in-the-blank questions. The system learns linguistic patterns from the context of the tokens in the process of trying to fill-in the masked tokens. Masking happens in a probabilistic manner. Each token has a certain chance, typically 15%, of getting masked. As a result of this masked language modeling, fragments of code that express identical processing, although in different programming languages are mapped to the identical representation, regardless of the input programming language. This is a common technique used in NLP tasks. The pretraining process is illustrated in Figure 5. Research has shown that pretraining the complete model in a multilingual manner can result in improvements in unsupervised machine translation. For this project, we followed the strategy of Facebook Artificial Intelligence Research team where a model is trained with a masked language modeling objective on singular source code datasets.



```
if(prime[p])
for(int i = p*p; i<=n; i+=p):
        prime[i] = false
```

```
MASK (prime[p])
for(MASK i =p*p; i<=n; i+=p);
                  prime[MASK] =
false;
```

```
if (prime[p])
for(int i= p*p; i<=n; i+=p):
                  prime[i] =
false;
```

Fig. 5 Model trained with a Masked Language Modeling objective

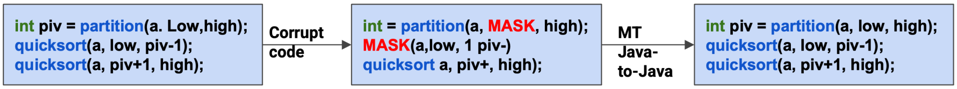
The cross-lingual nature of the resulting model comes from the significant number of common tokens (anchor points) that exist across languages. In the context of language-to-language translation, the anchor points consist essentially of digits and city and people names. In the case of programming languages like Python and R, these anchor points come from programming language keywords, for example, ‘for’,‘while’,’try’,’if’,’else’ and math operators like ‘+’,’- ’,’*’,’/’, also digits (0-9) which appear in the source dataset.

4.3.4 Denoising Auto Encoding

Encoders are artificial neural networks (ANN) which are generally used for feature extraction and feature selection. The encoders have nodes which are present in the hidden layer and at times outnumber the inputs being given to the network. In such cases the learning rate of the encoder is low as it does not perceive its Identity Function, wherein the output is the same as the unmodified input. A denoising encoder is the solution to this problem, whereby the input being passed to the nodes is corrupted by masking non-zero items randomly.

Generally, between 30 and 50% of the inputs are masked and fed into the layers and processed to find the loss function of the network. The output obtained must be compared to the original input instead of the corrupted one to ensure that the features are extracted instead of perceiving an identity function from the data. Once pretraining is done, the decoder is trained to generate valid code sequences even when fed with noisy data, increasing the encoder robustness to input noise. The process of denoising auto-encoding is illustrated in Figure 6.

As can be seen from the figure, the valid code in the leftmost box is corrupted by masking certain tokens and jumbling up certain other tokens. For instance, piv-1 is corrupted as 1 piv- and piv+1 is corrupted by dropping the 1. The rightmost box shows the recovered code that matches the correct code.



```
int piv = partition(a. Low,high);
quicksort(a, low, piv-1);
quicksort(a, piv+1, high);
```

```
int = partition(a, MASK, high);
MASK(a,low, 1 piv-)
quicksort a, piv+, high);
```

```
int piv = partition(a, low, high);
quicksort(a, low, piv-1);
quicksort(a, piv+1, high);
```

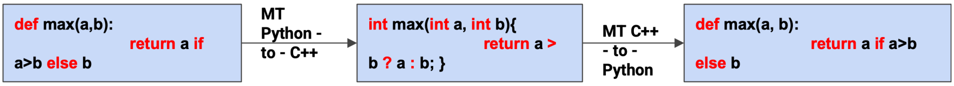
Fig. 6 The model is trained to predict the sequence of tokens given a corrupted sequence of tokens. Encoder encodes the sequence and produces noisy output. The Decoder is always trained to generate a valid function, even when the encoder output is noisy

During the computation of the Loss Function, it is of utmost importance to note that the output generated is tested with the initial input and not the noise filled input that has been fed to the model. By ensuring the above, the risk of absorbing the learning function instead of the model extracting attributes is minimized.

4.3.5 Back Translation

Back Translation is a procedure of re-translating the code to its original form. This is usually done in a completely unsupervised setting or semi-supervised setting. The Facebook research team mentioned in their paper that the translations generated from Cross-Lingual model pretraining and denoising auto-encoding could be of low quality as the model is not actually trained to make translations from one programming language to another. Back Translation addresses this issue by leveraging the monolingual in an unsupervised setting. It is illustrated in Figure 7

In a natural language translation using NMT (Neural Machine Translation), back translation is used under a semi- supervised setting where both monolingual and bilingual data is available. Similarly, in this paper, we are experimenting with this back-translation procedure with the python and R



```
def max(a,b):
                    return a if
a>b else b
```

```
int max(int a, int b){
                    return a >
b ? a : b; }
```

```
def max(a, b):
                    return a if a>b
else b
```

Fig. 7 Back Translation. The target-to-source model translates target sequences into the source language and the source-to-target model is trained to reconstruct the target sequences

datasets. After the translation, the code is generated by Cross-Lingual model pretraining and denoising auto-encoding, this source-to-target translation is coupled with the target-to-source translation model trained in parallel. The target sequences are translated to source sequences by using this target-to-source model. Finally, the target programming language code is translated back to original source code and its accuracy is verified against the programming code given by the user initially.

4.4 Architecture

The CodeTranslator model is essentially a sequence- to-sequence Transformer with an attention mechanism consisting of encoder and decoder components [18]. A Transformer consists of an encoder component and a decoder component.

4.4.1 Encoder

As described in [18], the encoder comprises a stack of N identical layers. Each layer has two sub-layers, namely, a multi-head self-attention mechanism, and a simple, position-wise fully connected feed-forward network. A residual connection [25] is employed around each of the two inner layers, followed by a layer of normalization [26]. That is, the output of each sub-layer is LayerNorm(x+Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself.

4.4.2 Decoder

Like the Encoder, the decoder too is composed of a stack of N. identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, a residual connection [25] is employed around each of the sub-layers, followed by layer normalization.

We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the output embedding are offset by one position, ensuring that the predictions for position ‘i’can depend only on the known outputs at positions less than ‘i’.

4.4.3 Attention

As described in [18], the attention function is a mapping. It maps a query and a set of key-value pairs K,V to an output. The keys, values, and the output are vector encoding from the same vector space. Each value is assigned a weight that is obtained using a function that represents the compatibility of the query with the corresponding key. The output is the weighted sum of the values. If the input comprises of queries and keys of dimension d_k, the values of dimension d_v, attention is computed using equation 1

$$attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

4.4.4 Positional encoding

In deep learning, recurrence and convolution are the two often used approaches to account for the position in the input stream. Since the Transformer architecture [18] neither uses convolution nor recurrence, a new technique called positional encoding is used to take into consideration, the position of the word in the input sequence. Positional encoding can be considered as extra information about the position that is added to the input embedding at the bottom of the encoder and decoder stacks. We now have a stacked self-attention mechanism built into the encoder and the decoder with fully connected layers.

4.5 Implementation

The implementation we attempted is quite straight-forward and follows from the above discussion as described below. Our end to end project design has a front end component with a MERN stack, which is not discussed in this paper for reasons of brevity, a CUDA environment for Training and a neural transcompiler, called a transformer. The transformer has been pretrained on C++, Java and Python programs and requires less time to tune and train when compared to its counterparts like RNN and LSTM. Transfer Learning is applied when it comes to training the model on R and Python alternatively between batches of back translation and the denoising auto encoder approaches. The model uses Adam Optimizer, paired with an attention mechanism to achieve excellent results. Figure 8 illustrates some of the tools and components used in the project implementation.

4.5.1 Model Training

The model implements a transformer with dimensionality- 6x8x1024, which represents the number of layers, attention heads, and model size, respectively. A single encoder and decoder are used for all the programming languages supported by the CodeTranslator model. The model has been pretrained on Python and R programs which have been fed alternately as 32 sequences of tokens, split across batches of 512 tokens. Transfer learning is applied when it comes to training the model on R tokens, alternating on training it uses both, the back translation and the denoising auto encoder approach, respectively. The model utilizes an Adam Optimizer, with a learning rate like what was

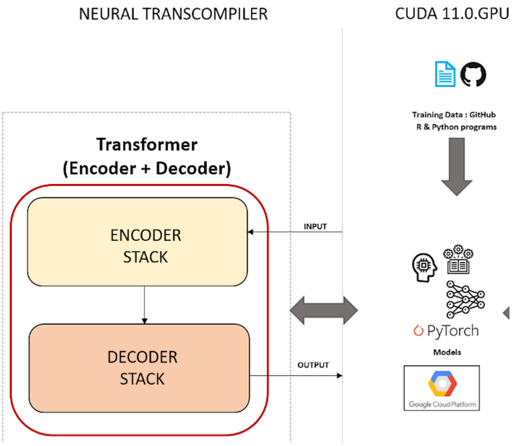


Fig. 8 Implementation

used in prior approaches. The model is trained on GPU-enabled systems to speed up training time.

The hyper-parameters we used for this project are listed below. Beam size is the width of the beam. Without beam search for finding the missing or masked token, we just go with the token that has the highest probability value as output by a softmax function. With beam search, instead of just one token, we start by considering multiple candidate tokens and eventually choose the best one after evaluating the probabilities of subsequent tokens. That is, we take a more holistic approach to filling in the blanks. For instance, if the beam size is 2, we initially consider two tokens instead of just one.

Dropout regularizes the model by randomly multiplying a few activations by zero. GELU combines Dropout with RELU. Adam Optimizer is Ensemble optimizer for faster convergence and less computation.

- epoch_size - Number of training iterations
- batch_size - Number of sentences per batch
- beam_size - beam search
- fp16 - 16-point floating point arithmetic for Faster Training and lower memory requirements
- activation - RELU (Rectified Linear Unit) or GELU (Gaussian Error Linear Unit activation) function for faster and better convergence of neural networks and to regularize the model
- optimizer - Adam optimizer with learning rate 0.0001
- stopping_criteria - model stops training after reaching lowest score

We used Google Cloud AI Platform Deep learning instance with NVIDIA Tesla K80 GPU, 4 CPU and 26 GB memory and Pytorch/CUDA11.0.GPU for our implementation. Some of the Python Libraries we used are listed below.

- NumPy : Libraries to support working with arrays and mathematical functions

- PyTorch: Machine learning library for natural language processing
- fastBPE: Neural machine translation (NMT) models with Python API
- Apex: NVIDIA utilities for distributed training in Pytorch
- submitit: Packages to run jobs for concurrent processing
- six: Utility to support python code in different versions

Table 2 lists the challenges we faced during implementation and the resolutions we adopted to get over them. Each experiment took a lot of time. Multiple GPUs may have helped in speeding up the training with large amounts of data

Table 2 Challenges faced at the time of implementation and their resolutions

Challenge	Resolution
Multiple GPUs constraint to train huge data sets	Combined multiple training files into a single file
Pytorch CUDA memory error with batch_size 32	Decreased batch_size to 16 and cleared CUDA memory cache
Disk space issue due to periodic model saving after each epoch	Modified code to store only final model and increased disk space

4.5.2 Testing

The solutions were a different set of programs from GitHub online platform that are extracted for Python, and R programming languages and used as validation and test datasets to test our model. The evaluation of the model was performed using Accuracy, Loss and Perplexity. The plots that were generated are discussed in the following paragraphs. One problem with this approach of using accuracy, loss, and perplexity is that if there are small syntax mismatches it may generate a high score even the output produced is completely different from that of the source code. To fix this issue, computational accuracy was used where the outputs of the source and target programming languages were compared and given a high score if there was a match with the same input values. Table 3 summarizes our test strategy. Figure 9 and 10 shows the success we had with the experiments at translating from Python to R and vice versa. The command line used is

```
$python3  <path>translate.py
    --src_lang <source language>
    --tgt_lang <target language>
    --BPE_path <path to BPE codes>
    --model_path <model path>
    checkpoint.pth < <source file path>
```

Table 3 Test Plan for the Machine Models

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Results
TC_01	Preprocess Code - with comments	Pass 'Yes' to keep comments parameter	Source code from Github repo	High accuracy values
TC_02	Preprocess Code - with no comments	Pass 'No' to keep comments parameter	Source code from Github repo	High accuracy values and low perplexity values
TC_03	Test with different number of Transformer layers	Pass different numbers to Transformer layers parameter	Source code from Github repo	High accuracy values and low perplexity values
TC_04	Test with different number of Transformer heads	Pass different numbers to Transformer heads parameter	Source code from Github repo	High accuracy values and low perplexity values
TC_05	Test with different activation functions	Pass different activation functions	Source code from Github repo	High accuracy values, low perplexity score and low values of loss.
TC_06	Preprocess for Python language	Pass language parameter 'Python'	Source code from Github repo	High accuracy values and low perplexity values
TC_07	Preprocess for R language	Pass language parameter 'R'	Source code from Github repo	High accuracy values and low perplexity values

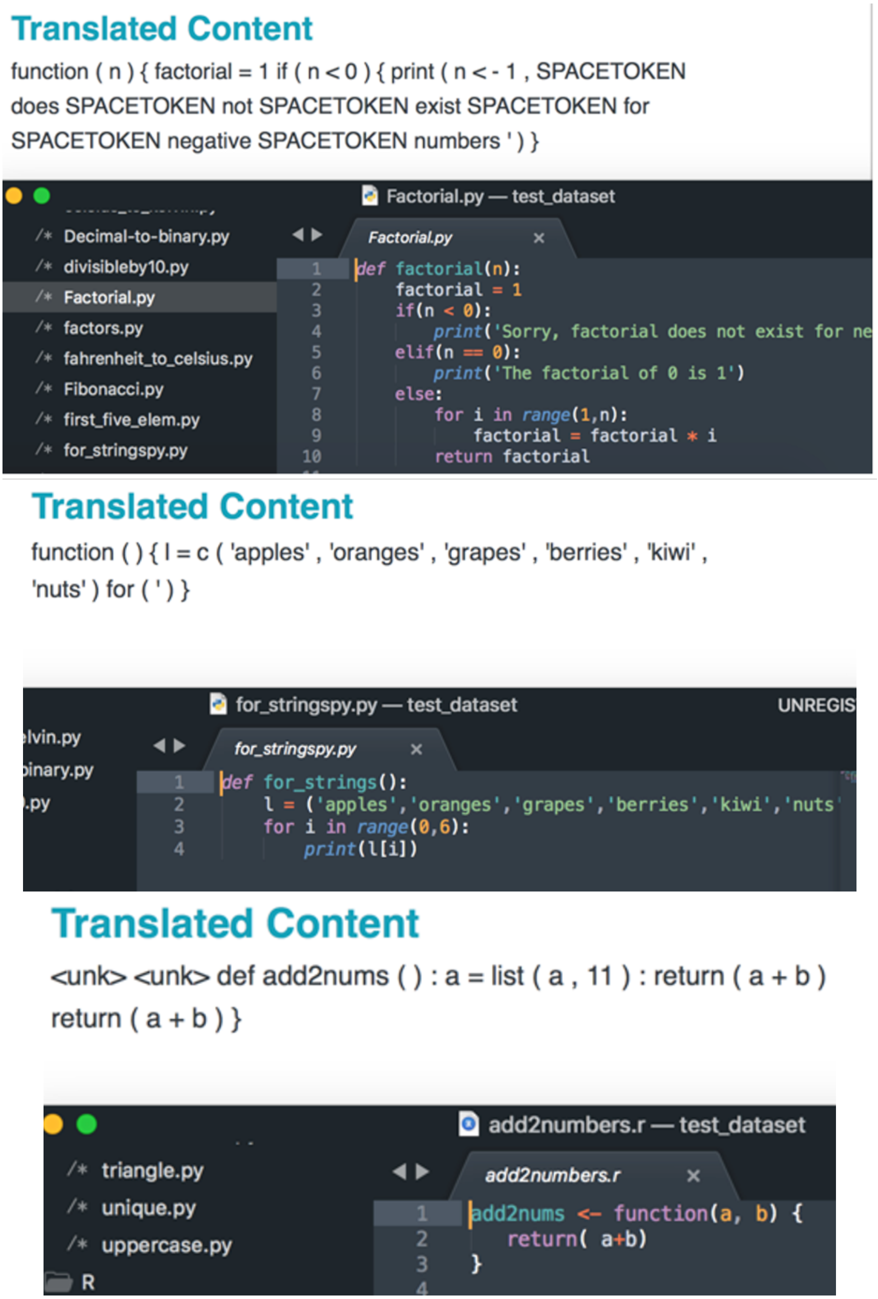
4.6 Results

The model was evaluated using computational accuracy, wherein the source keywords and tokens are mapped to the translated tokens to identify the number of matching elements. Perplexity score was used to measure the translation capability of the model, wherein a lower score indicates better performance. Model Loss was also used as a metric, to evaluate the models. The plots of these metrics were visualized in Tableau software and plotted as accuracy versus epochs, Batch size, and Beam size.

4.6.1 Model accuracy

Current research has established that for the deep learning models, using more resources such as by increasing model size, size of the dataset, or training steps, leads to higher model accuracy. With the computational resources increasing, there is a critical constraint on improving model accuracy.

Amplifying compute efficiency required reconsidering common assumptions about model training. The authors of the paper [27] proved the assumption that models must be trained until convergence, which made larger models appear less viable for limited compute budgets. In [18], the authors proved that the fastest way to train the Transformer models was to substantially increase model size but stop training very early.



Translated Content

function (n) { factorial = 1 if (n < 0) { print (n < - 1 , SPACETOKEN does SPACETOKEN not SPACETOKEN exist SPACETOKEN for SPACETOKEN negative SPACETOKEN numbers ') }

```
def factorial(n):
    factorial = 1
    if(n < 0):
        print('Sorry, factorial does not exist for ne
    elif(n == 0):
        print('The factorial of 0 is 1')
    else:
        for i in range(1,n):
            factorial = factorial * i
        return factorial
```

Translated Content

function ( ) { l = c ('apples' , 'oranges' , 'grapes' , 'berries' , 'kiwi' , 'nuts') for (') }

```
def for_strings():
    l = ('apples','oranges','grapes','berries','kiwi','nuts'
    for i in range(0,6):
        print(l[i])
```

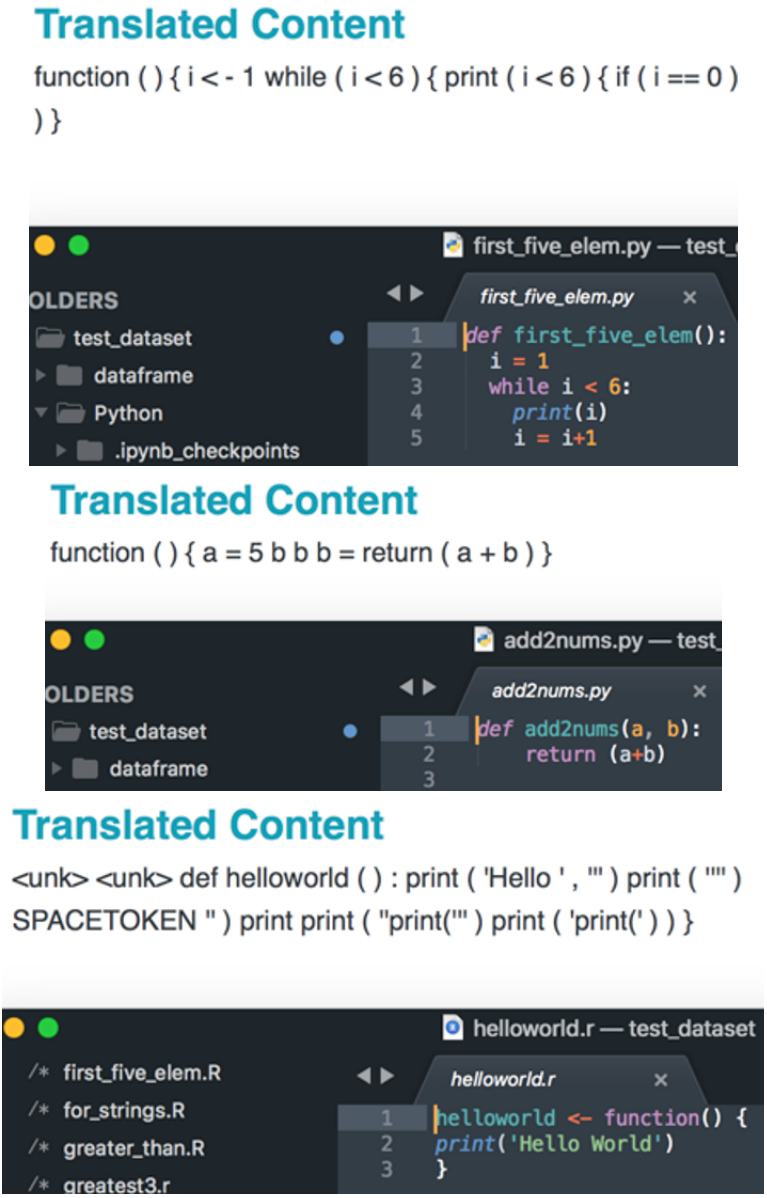
Translated Content

<unk> <unk> def add2nums ( ) : a = list (a , 11) : return (a + b) return (a + b) }

```
add2nums <- function(a, b) {
    return( a+b)
}
```

Fig. 9 Screenshots of the transcoding from Python to R and vice versa

In this paper, there is a stopping criterion to stop the training after 20 epochs. An increase in accuracy and better translation in the programming languages was observed. The results of accuracy versus epochs were recorded



Translated Content

function ( ) { i < - 1 while (i < 6) { print (i < 6) { if (i == 0)) }

```
def first_five_elem():
    i = 1
    while i < 6:
        print(i)
        i = i+1
```

Translated Content

function ( ) { a = 5 b b b = return (a + b) }

```
def add2nums(a, b):
    return (a+b)
```

Translated Content

<unk> <unk> def helloworld ( ) : print ('Hello ' , "') print ("'") SPACETOKEN ") print print ("print('") print ('print(')) }

```
helloworld <- function() {
print('Hello World')
}
```

Fig. 10 Screenshots of the transcoding from Python to R and vice versa

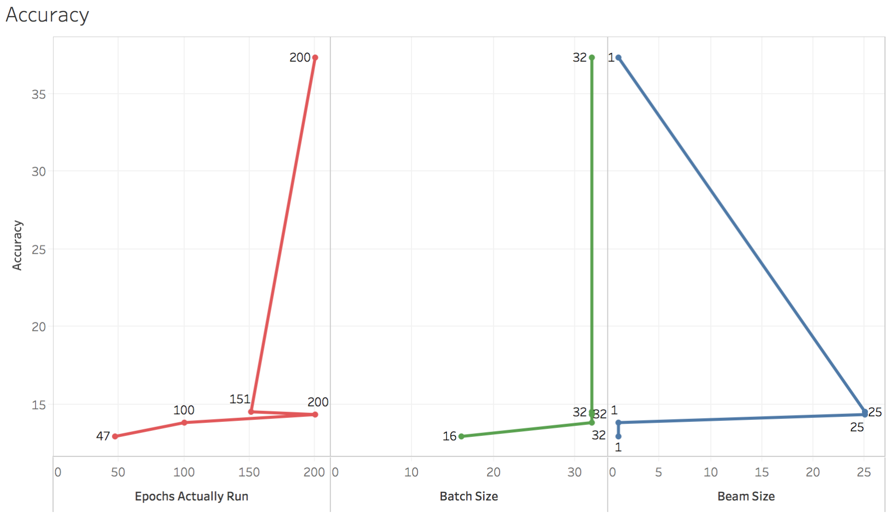
and as can be seen from Table 4 the model received 40% accuracy with 500 epochs.

As can be seen from Figure 11 the hyper-parameter sets (epochs, batch size, beam size) tried are (47, 16, 1), (100, 32, 1), (200, 32, 25), (151, 32, 25), and (200, 32, 1). The accuracy reached nearly 40% for 200 epochs, which

Table 4 Computational accuracy vs existing baselines

Baseline 1	Baseline 2	Our Experiments
Python to C++ 32.2	Python to Java 24.7	Python to R 37.341772

resulted in a better model. It was observed to be wavering around 13% to 14% for other permutations of the hyper-parameters.



The trends of Epochs Actually Run as an attribute, Batch Size as an attribute and Beam Size as an attribute for Valid Mlm Acc.

Fig. 11 Model accuracy vs epochs, batch, beam_size

4.6.2 Model perplexity

The existing research performed on Transformer models evaluated their perplexity on test data and information-theoretic assessment of its predictive power.

Perplexity $PP(W)$ is a metric used to evaluate the translation capability of a language model. It is defined as the inverse probability of the test set, normalised by the number of words as given in equation 2

$$PP(W) = \frac{1}{P(w_1, w_2, ..., w_n)^{\frac{1}{N}}} \tag{2}$$

where W represents the entire masked code made up of fragments of codes in a test set $(w_1, w_2, w_3...w_N)$.

A lower perplexity score indicates a better model. The plot of perplexity and epochs was plotted as in Figure 12 and observed to be decreasing for a larger number of epochs while training. The lower value of perplexity resulted in a better model for translation.

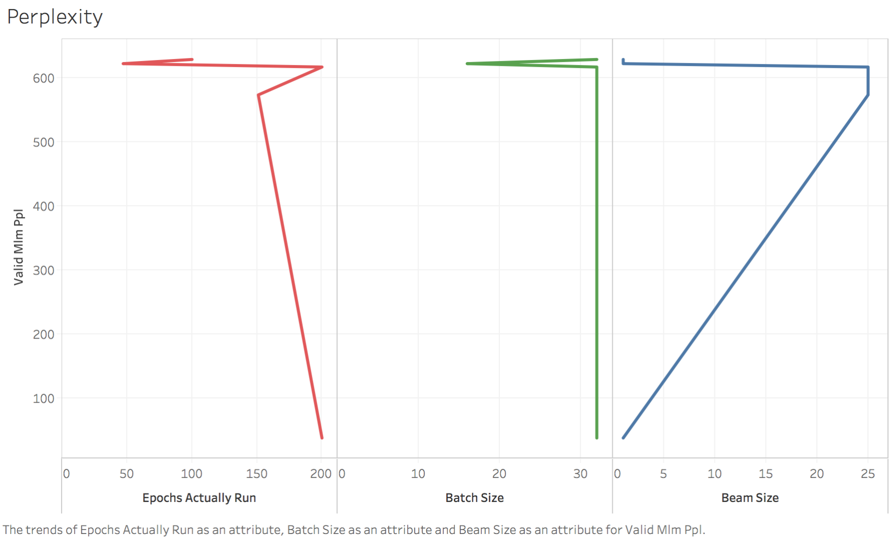


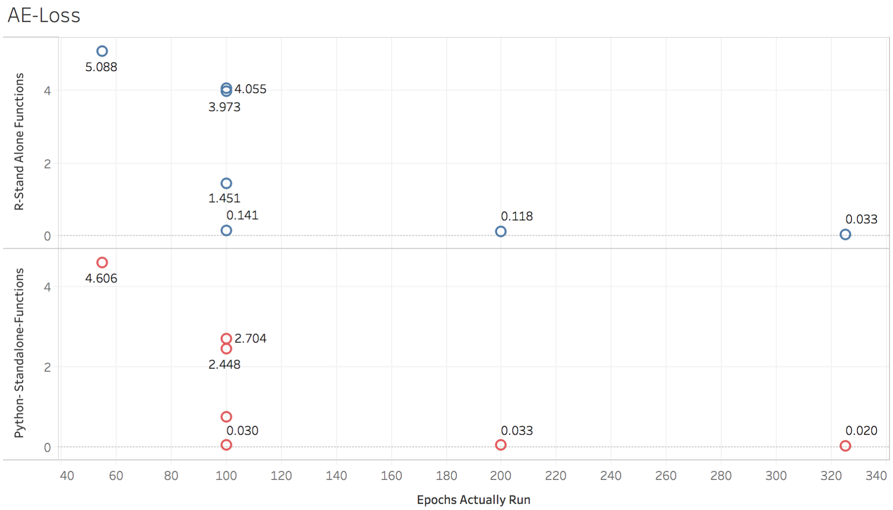
Fig. 12 Model perplexity vs epochs, batch, beam_size

4.6.3 Model Loss

The loss of the model is the mean squared error computation. The loss is observed by comparing the generated output with the initial input and not the noise filled input that has been fed to the model to ensure better training. The AE Loss vs Epochs plot was visualized using Tableau and plotted as in Figure 13 for Loss versus epochs. On observation, it was noted that loss decreased as the epochs increased and a model with lower loss resulted in better translation.

5 Conclusion

Time and again, software concepts continue to draw inspiration from the real world. Deliberately or unconsciously, many computer science artifacts bear a striking resemblance to the happenings in the real world. Similarity is probably the single most important underlying principle of machine learning. From a linear predictor to advanced deep learning frameworks, all use dot products. Dot product that is ubiquitously used in machine learning is a measure for similarity. It can therefore be concluded that machines predominantly learn by way of similarity. However, this fundamental way of learning remains unexplored to a significant extent in human learning of difficult topics like machine learning itself. Part of this paper and [2] attempted to address that gap. When the subject is challenging as is the case with machine learning, it helps to draw



Epochs Actually Run vs. AE-r sa and AE-python sa.

Fig. 13 Model loss vs epochs executed

parallels to the concepts that the students are already familiar with to help explain the underlying philosophy.

Accordingly, a few real-world analogies for machine learning concepts have been discussed in this paper in a tabular form. All the analogies are based on human intuition and ingenuity. A future direction for this work is to evaluate the feasibility of automatic generation of analogies, not just for machine learning topics, but for any advanced subject with hard-to-understand concepts. The similarity is a fundamental notion in machine learning. Using the right type of topic, language modeling, and NLP techniques, it may be possible to discover similarities automatically between topics using deep learning, particularly given the universal approximation theorem.

We also used similarity to improve the efficacy of machine learning by developing a software application called Transcoder for programming language translations between Python to R. The dataset for both programming languages was generated from Big Query with a collection of R and Python programs. As part of the preprocessing, tokenization and function extraction procedures were applied to both programs. As the model was able to identify the tokens and the functions XLM pretraining and denoising autoencoding models were applied to generate the translations. To improve the quality of translations, back translation was applied to this monolingual data generated from source-to-target and target-to-source models. The target sequences generated in this process were then translated to source code which was manually verified against the original target program given by the user. This verification helps us to corroborate the accuracy of the translation.

The performance of the model was evaluated using accuracy, perplexity, and loss metrics. On observation, it was noticed that in some of the translations, the name of the function and return keywords were missing when compared to the original code in the target programming language. Overall, the translations do not seem to be production ready yet as certain syntax and logical changes are not recognized for specific languages. Nevertheless, it is a good start at exploring similarity using machine learning and complements the work done in the existing literature, as described in the preceding sections.

The software application built for the purpose of unsupervised translation between Python and R can be extended to other programming languages as well. Currently, the translations are at function level only but can be extended to programs as well. We leave this study for further research. The models used in this paper, do not recognize Python or R libraries such as Pandas, NumPy, tidyr, ggplot2, etc. In the future, modifications can be made to recognize them in their equivalent target languages. In this paper, we analyzed function level translation for standalone functions, which can be extended to class functions as well. It is sincerely hoped that this research will open up a fruitful discussion on exploring similarity in the context of machine learning.

6 Acknowledgment

The authors wish to thank Harika Andugula, Gayathri Ganesh, and Nihanjali Mallavarapu for their contributions to the transcoding part of this research.

7 Conflict of Interest Statement

On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

[1] Domingos, P.: Every model learned by gradient descent is approximately a kernel machine. arXiv preprint arXiv:2012.00152 (2020)

[2] Pendyala, V.S.: Relating machine learning to the real-world: Analogies to enhance learning comprehension. In: International Conference on Soft Computing and Its Engineering Applications, pp. 127–139 (2022). Springer

[3] Lample, G., Conneau, A., Denoyer, L., Ranzato, M.: Unsupervised machine translation using monolingual corpora only. In: International Conference on Learning Representations (2018)

[4] Duit, R.: On the role of analogies and metaphors in learning science. Science education **75**(6), 649–672 (1991)

[5] Brink, H., Richards, J., Fetherolf, M.: Real-world Machine Learning. Simon and Schuster, New York (2016)

[6] Mehta, P., Bukov, M., Wang, C.-H., Day, A.G., Richardson, C., Fisher, C.K., Schwab, D.J.: A high-bias, low-variance introduction to machine learning for physicists. Physics reports **810**, 1–124 (2019)

[7] Helmstaedter, M.: The mutual inspirations of machine learning and neuroscience. Neuron **86**(1), 25–28 (2015)

[8] Hope, T., Chan, J., Kittur, A., Shahaf, D.: Accelerating innovation through analogy mining. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 235–243 (2017)

[9] Fan, Y., Tian, F., Qin, T., Li, X.-Y., Liu, T.-Y.: Learning to teach. In: International Conference on Learning Representations (2018)

[10] Burrell, J.: How the machine 'thinks': Understanding opacity in machine learning algorithms. Big Data & Society **3**(1), 2053951715622512 (2016)

[11] Fiebrink, R.: Machine learning education for artists, musicians, and other creative practitioners. ACM Transactions on Computing Education (TOCE) **19**(4), 1–32 (2019)

[12] Smith, L.B., Slone, L.K.: A developmental approach to machine learning? Frontiers in psychology **8**, 2124 (2017)

[13] Zhu, J.-Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2223–2232 (2017)

[14] Welander, P., Karlsson, S., Eklund, A.: Generative adversarial networks for image-to-image translation on multi-contrast mr images-a comparison of cyclegan and unit. arXiv preprint arXiv:1806.07777 (2018)

[15] Surana, R., Varshney, A., Pendyala, V.: Deep learning for conversions between melodic frameworks of indian classical music. In: Reddy, A.B., Kiranmayee, B.V., Mukkamala, R.R., Srujan Raju, K. (eds.) Proceedings of Second International Conference on Advances in Computer Engineering and Communication Systems, pp. 1–12. Springer, Singapore (2022)

[16] Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. Advances in neural information processing systems **27** (2014)

[17] Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)

[18] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)

[19] Lample, G., Ott, M., Conneau, A., Denoyer, L., Ranzato, M.: Phrase-based & neural unsupervised machine translation. arXiv preprint arXiv:1804.07755 (2018)

[20] Aziguli, W., Zhang, Y., Xie, Y., Zhang, D., Luo, X., Li, C., Zhang, Y.: A robust text classifier based on denoising deep neural network in the analysis of big data. Scientific Programming **2017** (2017)

[21] Kano, T., Sakti, S., Nakamura, S.: End-to-end speech translation with transcoding by multi-task learning for distant language pairs. IEEE/ACM Transactions on Audio, Speech, and Language Processing **28**, 1342–1355 (2020)

[22] Weiss, R.J., Chorowski, J., Jaitly, N., Wu, Y., Chen, Z.: Sequence-to-sequence models can directly translate foreign speech. Proc. Interspeech 2017, 2625–2629 (2017)

[23] He, D., Xia, Y., Qin, T., Wang, L., Yu, N., Liu, T.-Y., Ma, W.-Y.: Dual learning for machine translation. Advances in neural information processing systems **29** (2016)

[24] Artetxe, M., Labaka, G., Agirre, E., Cho, K.: Unsupervised neural machine translation. In: International Conference on Learning Representations (2018)

[25] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

[26] Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)

[27] Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., Gonzalez, J.: Train big, then compress: Rethinking model size for efficient training and inference of transformers. In: International Conference on Machine Learning, pp. 5958–5968 (2020). PMLR