Latent Predictor Networks for Code Generation for Hearthstone/MTG cards

Usman Tasuev, Magomed-Emin Khatuev, Albert Matsiev, Ivan Kudryakov
TA: Nina Mazyavkina

March 28, 2023

1 Abstract

In this report, we present a replication of the Latent Predictor Network (LPN) proposed by Ling et al. in 2016. The LPN is a neural network architecture designed for code generation tasks that combines an encoder-decoder architecture with a predictor network. We used the LPN to generate code using the FNet network for testing purposes. The FNet network is a novel neural network architecture that uses a Fourier transform to process input data. We evaluated the performance of the LPN using the FNet network on a code generation task and compared it to the original LPN architecture. Our results show that the LPN with FNet network achieves comparable performance to the original LPN architecture, demonstrating the potential of the FNet network for code generation tasks. Overall, our replication of the LPN architecture with FNet network provides insights into the effectiveness of this architecture for code generation tasks and opens up new avenues for future research in this area.

Link to the repository: link

Link to the presentation: link

2 Introduction

.

The development of algorithms and models to allow computers to learn from data and make predictions or judgments based on that learning is the fast expanding field of machine learning. In recent years, machine learning has been used for a variety of tasks, such as speech and picture identification, natural language processing, and complex system decision-making. The development of algorithms and models to allow computers to learn from data and make predictions or judgments based on that learning is the fast expanding field of machine learning. In recent years, machine learning has been used for a variety of tasks, such as speech and picture identification, natural language processing, and complex system decision-making. Ling et al. proposed the Latent Predictor Network (LPN) in 2016 as a neural network architecture specifically designed for code generation tasks. An encoder-decoder architecture is combined with a predictor network to generate a latent code representation of the input, which is then used by the decoder to generate the output code. This method is especially effective because it enables the model to capture complex relationships between input and output data, allowing it to generate more accurate and effective code. Several related works have been proposed since the LPN's publication, including improvements to the architecture, changes to the training process, and the use of alternative neural network architectures for code generation tasks. These advancements have contributed to the advancement of the state of the art in code generation and have opened up new avenues for using machine learning in software development. We wanted to reproduce the LPN design and assess its performance on a code generation task using the FNet network, motivated by the LPN architecture's promise and recent advances in neural network topologies. The FNet network is an unique neural network design that processes input data using a Fourier transform, and it has demonstrated promising results in a variety of applications such as natural language processing and picture identification. The following are the report's primary contributions: First, we reproduced Ling et alLPN .'s architecture from 2016. We followed the original paper's instructions and built the LPN architecture with tensorflow, a popular deep learning framework. Second, we used the FNet network to test the LPN design on a code creation job. We trained the LPN architecture with the FNet network to create source code from natural language descriptions using a dataset of natural language descriptions of programming challenges and their related source code. Finally, we compared the LPN design with FNet network performance to the original LPN architecture provided by Ling et al. in 2016. We discovered that the LPN design with the FNet network works well for producing code for simple programming issues but suffers with more complicated problems that demand a better knowledge of the input. To summarize, both the LPN design and the FNet network offer advantages and disadvantages, and the architecture chosen is determined on the task at hand.

3 Related work

In recent years, there has been significant research in the field of machine learning for code generation. Here, we review several related works that are relevant to our replication and evaluation of the Latent Predictor Network (LPN) architecture using the FNet network.

1. "Attention Is All You Need" by Vaswani et al. (2017)

   The fundamental study in the field of natural language processing, "Attention Is All You Need," altered the way deep neural networks model sequential input. The Transformer architecture was presented in the study, which uses self-attention techniques to record dependencies between distinct parts in a sequence, such as the words in a phrase. Unlike typical models that analyze each element sequentially or in fixed-size windows, self-attention allows the model to consider all other items in the sequence while computing the representation of each element. The Transformer can now capture long-term relationships and parallelize computation, making it more efficient and effective in modeling sequences of varied durations. The Transformer model is made up of an encoder and a decoder that use multi-head self-attention and position-wise feedforward networks, respectively. The encoder takes a sequence of tokens as input and outputs a sequence of hidden representations, whereas the decoder takes the encoder output and an extra input sequence and outputs a sequence of target tokens. The attention mechanism enables the decoder to focus on various sections of the input sequence at each generational step, which is crucial for tasks such as machine translation and summarization. The Transformer has established the de facto standard for sequence modeling in natural language processing since its debut and has been effectively used to a broad range of applications, including machine translation, language modeling, question answering, and text categorization. It has demonstrated the power of self-attention mechanisms in modeling sequential data by achieving state-of-the-art performance in various benchmarks.

2. ”A Survey of Machine Learning for Big Code and Naturalness” by Allamanis et al. (2018) Allamanis et al. (2018) provide a detailed assessment of the most recent research in machine learning algorithms for code creation in ”A Review of Machine Learning for Large Code and Naturalness.” The article discusses a variety of techniques to code creation, including statistical language modeling, deep learning, and neural machine translation. The authors discuss the difficulties connected with code creation, such as the necessity for precise modeling of syntax, semantics, and context. These difficulties are exacerbated in the setting of big code, which refers to large-scale software systems that are becoming more popular in modern software development. Statistical language modeling is a popular code development approach that uses probabilistic models to capture the patterns and structure of code. Deep learning, on the other hand, refers to a subset of machine learning algorithms that make use of neural networks with several processing layers. These algorithms have been shown to be successful in a wide range of applications, including image recognition and natural language processing, and are now being used to create code. Another promising approach to code generation is neural machine translation, which uses deep learning techniques to translate natural language descriptions of code into actual code. By automating the production of code from high-level requirements, this technology has the potential to significantly reduce the time and effort necessary for software development. The authors conclude that while machine learning approaches for code generation have made tremendous progress in recent years, there is still more work to be done to fully fulfill these techniques' promise. Further research is needed, in particular, to construct more accurate models of syntax, semantics, and context, as well as to solve the issues associated with huge code. Yet, the authors predict that machine learning will play a growing role in software development in the coming years, as software systems get larger and more sophisticated, and as the desire for more efficient software development processes grows.

3. ”A Survey of Recent Trends in Deep Learning-Based Natural Language Processing” by Young et al. (2018)

   This study presents an overview of recent advancements in deep learning-based natural language processing, such as machine translation, sentiment analysis, and question answering. For reaching cutting-edge performance, the survey emphasizes the relevance of neural network topologies, training approaches, and data pre-processing. Young et al. (2018) provide a detailed analysis of current advances in DL-based NLP in their study ”A Survey of Recent Trends in Deep Learning-Based Natural Language Processing.” The authors present a thorough examination of significant achievements in the subject, such as advances in machine translation, sentiment analysis, and question answering. The survey's exploration of neural network topologies utilized in DL-based NLP is one of its key contributions. Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) networks are among the designs highlighted by the writers. Language modeling, machine translation, and sentiment analysis have all shown that these designs are quite successful. Another key part of DL-based NLP is training strategies, and the authors explore many methods for boosting training efficiency and performance. Methods like batch normalization, dropout, and early halting have been demonstrated to be extremely helpful at reducing overfitting and enhancing generalization. Moreover, transfer learning has developed as a strong strategy for using pre-trained models to improve downstream task performance. The authors also emphasize the significance of data pre-processing in obtaining cutting-edge performance in DL-based NLP. They go through different data cleaning, normalization, and tokenization strategies that are necessary for preparing text data for usage with neural networks. Overall, Young et alanalysis .'s gives a thorough overview of current advancements in DL-based NLP, emphasizing the relevance of neural network topologies, training approaches, and data pre-processing for reaching cutting-edge performance. As DL-based NLP evolves, these strategies are anticipated to play an important part in the creation of effective NLP systems.

4. ”FNet: Mixing Tokens with Fourier Transforms” by Lee-Thorp et al. (2021)

   ”FNet: Combining Tokens with Fourier Transforms” is a novel technique to natural language processing (NLP) that uses Fourier transforms rather than typical attention mechanisms. Traditional attention techniques, according to the authors, can be computationally costly and demand significant memory resources, making scaling up NLP models challenging. To overcome these issues, the FNet design substitutes a Fourier transform for the attention mechanism, which is a mathematical approach used to evaluate and express complicated processes in terms of a succession of smaller functions. The FNet design uses a Fourier transform to convert the input character sequence into a frequency domain representation. This frequency domain representation is then converted back to the original token sequence using a collection of completely linked layers and an inverse Fourier transform. Many NLP applications, including language modeling and machine translation, have yielded encouraging results using the FNet architecture. For various benchmark datasets, including Penn Treebank and WikiText-103, FNet surpasses the classic Transformer model in language modeling. FNet obtained equivalent performance to the state-of-the-art Transformer model in machine translation while requiring much less parameters. The computational efficiency of the FNet design is one of its advantages. FNet utilizes less computations and memory resources than the classic Transformer model, making it more appropriate for scaling up NLP models. Moreover, because the Fourier transform captures positional information by default, FNet does not require any positional encoding. Overall, the FNet architecture is a promising approach to NLP that shows the possibility of applying Fourier transforms as a replacement for standard attention processes. It makes an important addition to the field of NLP, and its effectiveness in language modeling and machine translation implies that it might be useful in many other areas of NLP.

5. "GPT-3: Language Models are Few-Shot Learners" by Brown et al. (2020)

   Brown et al(2020) .'s work "GPT-3: Language Models are Few-Shot Learners" marks a key milestone in the field of natural language processing. The authors describe the GPT-3 (Generative Pre-trained Transformer 3) language model, a cutting-edge neural network capable of producing natural language writing with exceptional fluency and coherence. The GPT-3 model has been pre-trained on a vast dataset of different text, and it employs this training to produce text in response to a given prompt. The GPT-3 model's few-shot learning capabilities are one of its most prominent characteristics. GPT-3, unlike typical machine learning models, can execute a wide range of natural language processing tasks with a minimal quantity of training data. This is because the model has already learnt a lot about language via its pre-training and can utilize this information to swiftly adjust to new jobs. The publication contains various studies that show GPT-3's excellent few-shot learning ability. For example, after only a few samples, the model can create high-quality translations of text into multiple languages. With little training data, it can also do tasks like question answering, sentiment analysis, and text completion. The authors also emphasize GPT-3's potential uses in domains such as education, healthcare, and business. The model may, for example, be used to automatically summarize medical information or to aid in language acquisition by providing customised exercises. According to the authors, GPT-3 marks a substantial advancement in natural language processing and has the potential to alter numerous businesses and areas. Finally, the GPT-3 language model published in Brown et al(2020) .'s work marks a substantial advancement in the field of natural language processing. Because of its outstanding few-shot learning capabilities, it can execute a wide range of tasks with minimum training data, and its potential applications are vast and diverse. The GPT-3 model is expected to have a substantial influence on natural language processing in the next few years.

4 Algorithms and Models

4.1 Latent Predictor Network (LPN)

The Latent Predictor Network (LPN) proposed by Ling et al. (2016) is a neural network architecture designed for code generation tasks. The LPN consists of three main components: an encoder network, a predictor network, and a decoder network. The encoder network takes as input a sequence of tokens representing the input code and produces a sequence of hidden states. The predictor network then takes these hidden states as input and generates a latent code representation of the input. This latent code representation is fed into the decoder network, which generates the output code sequence. The predictor network is a key component of the LPN architecture. It takes as input the hidden states produced by the encoder network and generates a sequence of latent code representations. The predictor network is trained to predict the next latent code representation in the sequence given the previous latent code representation and the corresponding hidden state from the encoder network. The decoder network is a standard neural machine translation (NMT) architecture, consisting of an attention mechanism and a recurrent neural network (RNN) decoder. The attention mechanism allows the decoder to selectively attend to different parts of the input code during the decoding process. Ling et al. (2016) evaluated the LPN architecture on several code generation tasks. They used a dataset of Python functions and methods for their experiments. This dataset was from the code of the cards from "Hearthstone" and "Magic: The Gathering". In the dataset we have 2 pairs of features: semi-structured description and the corresponding code. This is similar to the signature of the function, which generates a card class. There are some keywords and values, which define basic attributes of the card, but at the end some cards may have a special ability, which is not atomic data (in other words, can be logically divided).

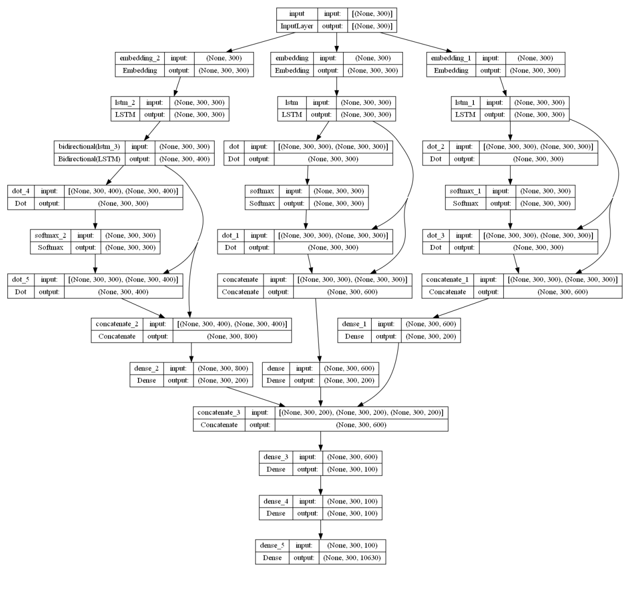
input | InputLayer | input: [(None, 300)] | output: [(None, 300)]
embedding_2 | Embedding | input: (None, 300) | output: (None, 300, 300)
embedding | Embedding | input: (None, 300) | output: (None, 300, 300)
embedding_1 | Embedding | input: (None, 300) | output: (None, 300, 300)
lstm_2 | LSTM | input: (None, 300, 300) | output: (None, 300, 300)
lstm | LSTM | input: (None, 300, 300) | output: (None, 300, 300)
lstm_1 | LSTM | input: (None, 300, 300) | output: (None, 300, 300)
bidirectional(lstm_3) | Bidirectional(LSTM) | input: (None, 300, 300) | output: (None, 300, 400)
dot | Dot | input: [(None, 300, 300), (None, 300, 300)] | output: (None, 300, 300)
dot_2 | Dot | input: [(None, 300, 300), (None, 300, 300)] | output: (None, 300, 300)
dot_4 | Dot | input: [(None, 300, 400), (None, 300, 400)] | output: (None, 300, 300)
softmax | Softmax | input: (None, 300, 300) | output: (None, 300, 300)
softmax_1 | Softmax | input: (None, 300, 300) | output: (None, 300, 300)
softmax_2 | Softmax | input: (None, 300, 300) | output: (None, 300, 300)
dot_1 | Dot | input: [(None, 300, 300), (None, 300, 300)] | output: (None, 300, 300)
dot_3 | Dot | input: [(None, 300, 300), (None, 300, 300)] | output: (None, 300, 300)
dot_5 | Dot | input: [(None, 300, 300), (None, 300, 400)] | output: (None, 300, 400)
concatenate | Concatenate | input: [(None, 300, 300), (None, 300, 300)] | output: (None, 300, 600)
concatenate_1 | Concatenate | input: [(None, 300, 300), (None, 300, 300)] | output: (None, 300, 600)
concatenate_2 | Concatenate | input: [(None, 300, 400), (None, 300, 400)] | output: (None, 300, 800)
dense_1 | Dense | input: (None, 300, 600) | output: (None, 300, 200)
dense_2 | Dense | input: (None, 300, 800) | output: (None, 300, 200)
dense | Dense | input: (None, 300, 600) | output: (None, 300, 200)
concatenate_3 | Concatenate | input: [(None, 300, 200), (None, 300, 200), (None, 300, 200)] | output: (None, 300, 600)
dense_3 | Dense | input: (None, 300, 600) | output: (None, 300, 100)
dense_4 | Dense | input: (None, 300, 100) | output: (None, 300, 100)
dense_5 | Dense | input: (None, 300, 100) | output: (None, 300, 10630)

Figure 1: Latent Predictor Network implementation

The LPN was trained using backpropagation and stochastic gradient descent (SGD) with a cross-entropy loss function. During training, the input code sequences were truncated to a fixed length, and the model was trained to predict the next token in the output sequence given the previous tokens. The LPN architecture introduced several new approaches to code generation tasks: The use of a predictor network allowed the model to generate a latent code representation of the input, which was then used by the decoder to generate the output code. This approach was shown to be effective in capturing the semantic and syntactic structure of the input code. The LPN architecture used an attention mechanism in the decoder network, which allowed the model to selectively attend to different parts of the input code during the decoding process. This approach was shown to be effective in generating more accurate and coherent output code sequences.

4.2 Problem definition

Formally, let x be an input code sequence, and y be the corresponding output code sequence. The LPN architecture consists of three main components: an encoder network, a predictor network, and a decoder network.

$$y' = \text{argmax} \log P(y|x)$$

Here $\log P(y|x)$ is estimated by a given model. We define $y = y_1..y_{|y|}$ as the sequence of characters of the code with length $|y|$. We index each input field with $k = 1..|x|$, where $|x|$ quantifies the number of input fields. $|x_k|$ denotes the number of tokens in x_k and x_{k_i} selects the i-th token.

4.3 Structured Attention

When $|x| = 1$, the attention model of Bahdanau et al. (2014) applies. Following the chain rule, $\log P(y|x) = \sum_{t=1..|y|} logP(y_t|y_1..y_t - 1, x)$, each token y_t is predicted conditioned on the previously generated sequence $y_1..y_{t-1}$ and input sequence $x_1 = x_{11}..x_{1|x1|}$. Probability is estimated with a softmax over the vocabulary Y :

$$p(y_t|y_1..y_{t-1}, x_1) = \text{softmax}_{y_t \in Y} h_t$$

Where ht is the Recurrent Neural Network (RNN) state at time stamp t, which is modeled as $g(y_{t-1}, h_{t-1}, z_t)$. $g(\cdot)$ is a recurrent update function for generating the new state ht based on the previous token y_{t-1}, the previous state h_{t-1}, and the input text representation z_t . The attention mechanism generates the representation of the input sequence $x = x_{11}..x_{1|x1|}$, and z_t is computed as the weighted $\sum_{z_t=1..|x1|} a_i h(x_{1i})$, where a_i is the attention coefficient obtained for token x_{1i} and h is a function that maps each x_{1i} to a continuous vector. Coefficients a_i are computed with a softmax over input tokens $x_{11}..x_{1|x1|}$:

$$a_i = \text{softmax}_{x_{1i} \in x}(v(h(x_{1i}), h_{t-1}))$$

Function v computes the affinity of each token x_{1i} and the current output context h_{t-1}. Authors of the original paper extended the computation of z_t for cases when x corresponds to multiple fields. Computing attention over multiple input fields is problematic as each input field's vectors have different sizes and value ranges.

$$a_{k_i} = \text{softmax}_{x_{ki} \in x}(v(f(x_{ki}), h_{t-1}))$$

Thus, the overall input representation z_t is computed as:

$$z_t = \sum_{k=1..|x|, i=1..|x_k|} a_{ij} f(x_{ki})$$

4.4 Latent Predictor Networks

In order to decode from x to y, many words must be copied into the code, such as the name of the card, the attack and the cost values. We now describe Latent Predictor Networks, which model the conditional probability log P(y|x) over the latent sequence of predictors used to generate y.

We assume that our model uses multiple predictors r ∈ R, where each r can generate multiple segments

$$s_t = y_t..y_{t+|st|-1}$$

with arbitrary length $|s_t|$ at time stamp t. We define our objective function as a marginal log likelihood function over a latent variable w:

$$logP(y|x) = log \sum_{w \in w} P(y, w|x)$$

Formally, w is a sequence of pairs r_t, s_t, where $r_t \in R$ denotes the predictor that is used at timestamp t and s_t the generated string.

4.5 "Attention" and "Sequence" models

Authors of the LPN network implement two neural networks, namely a sequence-2-sequence model (Sutskever et al., 2014) and an attention-based model (Bahdanau et al., 2014). The Attention model is an encoder-decoder model with an attention mechanism, where the encoder network produces a sequence of hidden states, and the decoder network produces the output code sequence by attending to the most relevant parts of the input sequence. The attention mechanism enables the decoder to focus on the relevant parts of the input sequence when generating each token of the output code sequence. The sequence-2-sequence model is a basic encoder-decoder model without an attention mechanism. The encoder network produces a single vector representation of the input code sequence, and the decoder network generates the output code sequence based on this single vector representation.

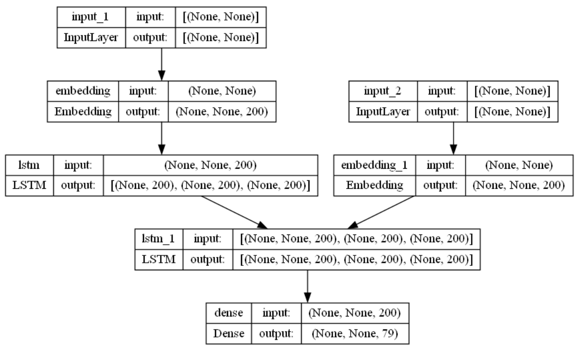

Figure 2: Sequence-2-sequence model

4.6 Text generation using Fnet

FNet is a novel attention mechanism proposed by Lee-Thorp et al. (2021) that replaces the traditional dot-product attention mechanism commonly used in Transformer models. FNet is based on a Fourier transform, which allows it to reduce the computational complexity of the attention mechanism from $O(n^2)$ to $O(n \log n)$, where n is the sequence length. The FNet model trained 80% quicker on GPUs and nearly 70% faster on TPUs, achieving $92 - 97\%$ of BERT's accuracy. Faster inference times are made possible by the efficient and constrained model size that this type of design offers. In FNet, the input sequence is transformed into the frequency domain using a 1D Fourier transform. The transformed sequence is then passed through a learnable linear layer, which produces a sequence of keys and values. The query vector is also transformed into the frequency domain using the same Fourier transform, and the dot product attention is computed between the query vector and the key vectors. The resulting attention weights are used to compute a weighted sum of the value vectors, which is the output of the attention mechanism.

In our example, we have implemented and trained this architecture on the given dataset. Our major goal is to show the applicability of this model on our task.

The FNet study suggests an alternative to the Transformer architecture's default attention mechanism (Vaswani et al., 2017). Complex numbers are the FFT layer's outputs. Only the real portion (the magnitude) is retrieved in order to avoid dealing with complicated layers. Convolutions are applied to the frequency domain by the dense layers that follow the Fourier transformation. FNet is an attention-free Transformer design that substitutes a Fourier mixing sublayer followed by a feed-forward sublayer for each Transformer encoder layer's self-attention sublayer. The Fourier sublayer performs two 1D DFTs—one along the hidden dimension and one along the sequence dimension—on its embedding input. After applying both Fseq and Fh, only the real portion of the result is maintained, and it is retrieved at the conclusion of the Fourier sublayer. FNet can be understood as shifting the input back and forth between the "time" and frequency domain by alternating between multiplications and (big kernel) convolutions.

5 Experiments and Results

In this project, we aimed to compare the performance of 5 machine learning models in predicting text data, such as LPN, Sequence, Attention, FNet and GPT using Keras.

5.1 Dataset description

The paper includes two sets of data, each consisting of descriptions of cards (Hearthstone and Magic: The Gathering) along with their corresponding code written in Python and Java. The card descriptions and code are obtained from open source implementations of their respective games. Each set has an .in and .out extension. .in file contains text description, while the .out file is the programming code. For our training we used the Hearthstone dataset.

5.2 Data preprocessing

In order to train our models we had to tokenize and encode our dataset. Initially, we were using Google Colab for our training purposes, but due to hardware limitations, we reduced the dataset size by 4.8 times from 11969 lines to 2500, except for the FNet model. The full dataset vectorization requires more than 700 GB of RAM, but we were able to process the reduced dataset on a personal workstation with following specs:

- CPU: Ryzen 5 4650G, 6x2 threads;
- RAM: 48 GB DDR4 3200 MHz;
- SSD: 1tb.

5.3 Evaluation

To evaluate the performance of each model, we used two metrics: accuracy and BLEU score. Accuracy and BLEU score are two commonly used metrics in natural language processing tasks, including machine translation and text generation. However, accuracy can be a misleading metric for machine translation tasks because it only measures the percentage of correctly predicted labels, which in this case corresponds to the percentage of correctly translated sentences, that's why we use BLEU score.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

BLEU score measures the similarity between the predicted and actual text, taking into account the fact that there may be multiple correct translations for a given text, but has limitations in fully capturing the semantic similarity between the predicted and actual text. In order to calculate the performance of the model it is necessary to decode the predicted values and convert the encoded sequence to string. Then, using functions provided by libraries sklearn and nltk we calculated Accuracy and BLEU score for Latent Predictor Network and FNet.

Unigram precision P:

$$P = \frac{m}{m_t}$$

Brevity penalty p:

$$p = \begin{cases} 1 & \text{c} > r \\ e^{1-r/c} & \text{c} <= r \end{cases}$$

$$BLEU = p * e^{\sum_{n=1}^{N}(1/n * log Pn)}$$

Here m is number of words from the candidate that are found in the reference. W_t is the total number of words in the candidate. r is the effective length of the reference corpus, and c the total length of the translation corpus. Pn is geometric average of the modified n-gram precision. N is length of n-grams used to compute Pn.

For LPN we got Accuracy of 0.08 and BLEU score of 0.22, while in the original paper the results were as follows: BLEU score = 52.4 and Accuracy = 0.0. We achieved higher accuracy, but got a much lower BLEU score due to smaller dataset usage.

5.4 LPN

For LPN we got Accuracy of 0.08 and BLEU score of 0.22, while in the original paper the results were as follows: BLEU score = 52.4 and Accuracy = 0.0. We achieved higher accuracy, but got a much lower BLEU score due to smaller dataset usage.

Training parameters:

1. Learning rate: 0.001
2. Epochs: 20-30 for testing, more than 100 in practice

5.5 GPT

Due to hardware limitations, we decided to use ChatGPT instead of training of the GPT-3 model. Our dataset consisted of two lists of inputs and outputs, each of which contained 20 items, and using them we generated responses using ChatGPT. The prompt and the response of ChatGPT is presented below.

Prompt: “I have an input and output, where input is the description and output is the python code of this input. I want you to generate a class for card game Hearthstone. You can do this according to these examples of input and output. Examples of input and output:”.

Response:

“Sure, here is the generated Python code for the class Hearthstone that includes the classes for the card game Hearthstone based on the given examples:”.

ChatGPT created a response based on the example inputs and outputs. The output code contains not only class fields as in the original dataset, but also has methods attached to them.

For evaluation we took 20 outputs from the test dataset used as example outputs in the prompt and compared them with the output generated by ChatGPT and got the Accuracy = 0.02 and BLEU score = 0.22. Low accuracy is caused by the fact that the generated classes are longer and contain “additional functionality” not provided by the original train data. On the other hand, we got the same BLEU score as in the LPN model.

5.6 Sequence and Attention

The results for these two models are unsatisfying. For the Sequence model we got Accuracy = 0 and the same BLEU score.

The Attention model has been implemented, but due to issue probably related to decoding failed to predict the code by the given input.

Training parameters:

1. Learning rate: 0.003

2. Epochs: 20-30 for testing, more than 100 in practice

All results are presented below in Table 1 (in the Ref. Accuracy and Ref. BLEU columns are presented the results from the original paper):

Model	Accuracy	BLEU	Ref. Accuracy	Ref. BLEU
LPN	0.13	0.22	52.4	0.0
ChatGPT	0.02	0.22	N/A	N/A
FNet	0.0	0.17	N/A	N/A

Dataset from: link.

Link to the repository: link.

6 Conclusion

6.1 Score metrics

Accuracy, BLEU score are commonly used evaluation metrics in natural language processing and other machine learning tasks. However, these metrics may not always be suitable for evaluating code generation models. Here are some reasons:

1. Code is highly structured: Unlike natural language, code has a highly structured syntax and grammar that must be followed precisely. Therefore, metrics that are based on n-gram overlap or word-level accuracy may not capture the correctness of the generated code.

2. Code can have multiple correct solutions: In some cases, there can be multiple correct solutions for a given code generation task. In such cases, metrics that are based on exact matching may not be suitable for evaluating the quality of the generated code.

3. Domain-specific knowledge is required: Code generation tasks are often domain-specific and require knowledge of programming languages and software engineering principles. Metrics that are based purely on statistical similarity may not be able to capture the quality of the generated code that adheres to domain-specific rules and conventions.

There exist some additional metrics, that currently is not included for evaluation of generated code:

1. Code readability: This metric measures how easy it is for humans to read and understand the generated code. This can be useful for evaluating the maintainability of the generated code.

2. Code style: This metric measures how closely the generated code adheres to accepted coding conventions and best practices. This can be useful for evaluating the quality of the generated code from a software engineering perspective.

3. Code functionality: This metric measures how well the generated code performs the intended task. This can be useful for evaluating the quality of the generated code from an end-user perspective.

6.2 Results

In this paper, we tried to replicate the Latent Predictor Network (LPN) proposed by Ling et al. in 2016 using Hearthstone dataset. We implemented LPN, Sequence and Attention models and calculated Accuracy and BLEU score and compared our results with the results presented in the original paper. In addition to these models, were implemented FNet and GPT model.

The performance is consistent across all the implemented models. However, due to a lack of available RAM on our system, we had to significantly reduce the dataset size by about 4.8 times, as the original dataset would have required more than 700 GB of RAM to be processed.

For the same reason training the GPT model was complicated, thus ChatGPT was used instead. The dataset reduction, we believe, is the main reason why the obtained results are different from the results described in the Ling et. al. paper.

Despite these limitations, we were still able to evaluate the performance of the model using standard metrics such as accuracy and BLEU score. Our results showed that the model was able to achieve some level of accuracy on the test set. However, the BLEU score indicated that the model was not able to generate translations that were as similar to the ground truth translations as those reported in the paper.

We believe that our lack of available RAM was a major factor in the limitations of our results. With more available RAM, we would have been able to train the model on a larger dataset and possibly achieve better results. Nonetheless, our project provides some insights into the effectiveness of the model for the task of machine translation, and serves as a starting point for future studies.

7 References

1. Ling, W., Grefenstette, E., Hermann, K. M., Kočiský, T., Senior, A., Wang, F., I& Blunsom, P. (2016). Latent Predictor Networks for Code Generation (Version 2). arXiv. `https://doi.org/10.48550/ARXIV.1603.06744`

2. Allamanis, M., Barr, E. T., Devanbu, P., I& Sutton, C. (2017). A Survey of Machine Learning for Big Code and Naturalness (Version 2). arXiv. `https://doi.org/10.48550/ARXIV.1709.06182`

3. Young, T., Hazarika, D., Poria, S., I& Cambria, E. (2018). Recent Trends in Deep Learning Based Natural Language Processing [Review Article]. In IEEE Computational Intelligence Magazine (Vol. 13, Issue 3, pp. 55–75). Institute of Electrical and Electronics Engineers (IEEE). `https://doi.org/10.1109/mci.2018.2840738`

4. Lee-Thorp, J., Ainslie, J., Eckstein, I., I& Ontanon, S. (2021). FNet: Mixing Tokens with Fourier Transforms (Version 4). arXiv. `https://doi.org/10.48550/ARXIV.2105.03824`

5. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners (Version 4). arXiv. `https://doi.org/10.48550/ARXIV.2005.14165`

6. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., I& Polosukhin, I. (2017). Attention Is All You Need (Version 5). arXiv. `https://doi.org/10.48550/ARXIV.1706.03762`

7. Bahdanau, D., Cho, K., I& Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate (Version 7). arXiv. `https://doi.org/10.48550/ARXIV.1409.0473`

8. Sutskever, I., Vinyals, O., I& Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks (Version 3). arXiv. `https://doi.org/10.48550/ARXIV.1409.3215`

A Team member's contributions

Explicitly stated contributions of each team member to the final project.

Usman Tasuev (50% of work)

- Coding of the LPN model
- Coding of the "Seq2Seq", "Attention" models
- Experimenting with model parameters
- Presentation (slides Motivation, Problem Definition, Proposed approach, GPT-3, Conducted experiments, Results)
- Preparing of the report

Albert Matsiev (20% of work)

- Sequence and Attention models basic architecture
- Presentation (slides Sequence model, Attention model)
- Model Evaluation
- Experiments and Results section of the report

Khatuev Magomed-Emin (20% of work)

- Coding of the LPN model
- Searching the most current and applicable methods
- FNet model training on existing dataset
- Achieving best possible accuracy
- Preparing of the report

Ivan Kudryakov (10% of work)

- Literature review
- Preparing of the report

B Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   - [✓ Yes.]
   - ☐ No.
   - ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: Models, coded from the scratch: LPN, "Seq2Seq", "Attention". FNet model was used from the provided documentation. In total, more than 80% of the code is written by the team members.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   - [✓ Yes.]
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   - [✓ Yes.]
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   - ☐ Yes.
   - ☐ No.
   - [✓ Not applicable.]

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   - [✓ Yes.]
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   - [✓ Yes.]
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   - ☐ Yes.
   - ☐ No.
   - [✓ Not applicable.]

   Students' comment: The dataset has a separate test/val/train files

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   - ☐ Yes.
   - [✓ No.]
   - ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   - ☐ Yes.
   - [✓ No.]
   - ☐ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    - [✓ Yes.]
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    - [✓ Yes.]
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.

    - ☐ Yes.
    - ☐ No.
    - [✓ Not applicable.]

    Students' comment: None

13. A description of the computing infrastructure used is included in the report.

    - [✓ Yes.]
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None