Latent Predictor Networks for Code Generation for Hearthstone/MTG cards

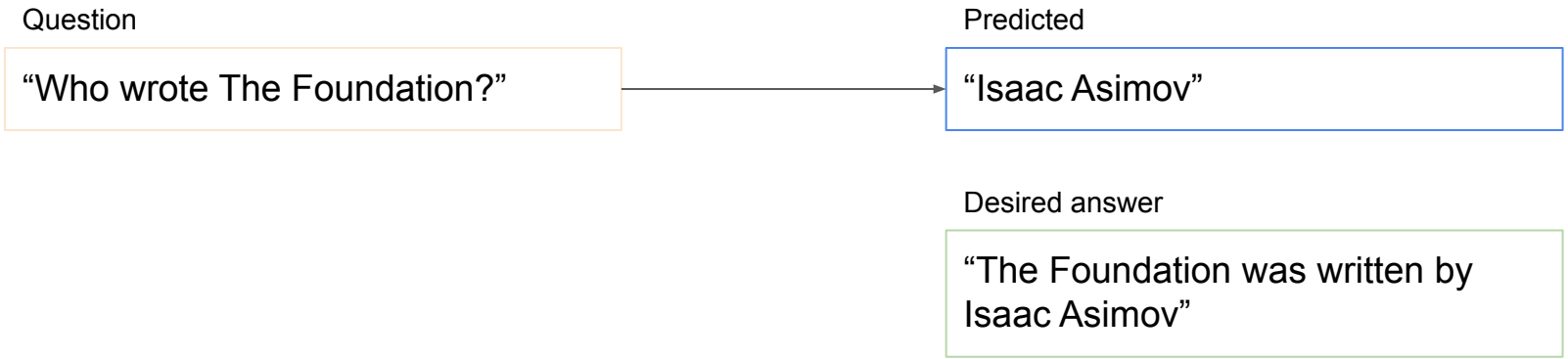
Team:
Usman Tasuev
Magomed-Emin Khatuev
Albert Matsiev
Ivan Kudryakov

TA:
Nina Mazyavkina

Motivation

The problem of generating programming code from a mixed natural language and structured specification.

Example:
predictor 1 to copy “The Foundation”
predictor 2 - find the answer “Issac Asimov”



Problem Definition

$$y' = \arg\max \log P(y/x)$$

x - description of a card,
y - code,
$y = y_1 .. y_{|y|}$

Example MTG and HS cards.

Description

Divine Spirit NAME_END -1 ATK_END -1 DEF_END 2 COST_END -1 DUR_END Spell TYPE_END Priest PLAYER_CLS_END NIL RACE_END Common RARITY_END Double a minion's Health.

Code generator

```
class DivineFavor(SpellCard):
    def __init__(self):
        super().__init__("Divine Favor", 3,
    CHARACTER_CLASS.PALADIN, CARD_RARITY.RARE)

    def use(self, player, game):
        super().use(player, game)
        difference = len(game.other_player.hand)
       - len(player.hand)
        for i in range(0, difference):
            player.draw()
```

Proposed approach

Standard

$$a_i = soft\max(u(h(x_{1j}),\ h_{t-1}))$$

$$where\ x = x_{11}, .., x_{1|x_1|}$$

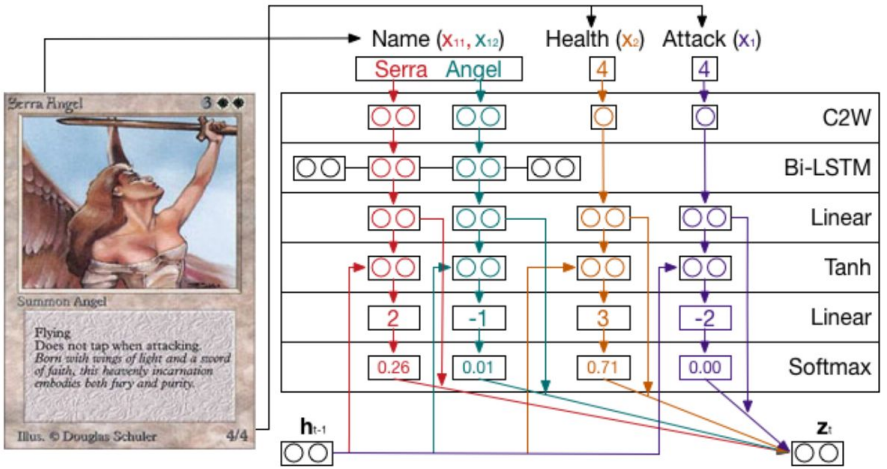
$$z_t = \sum_{i=1}^{|x_1|} a_i h(x_{1j})$$

Proposed

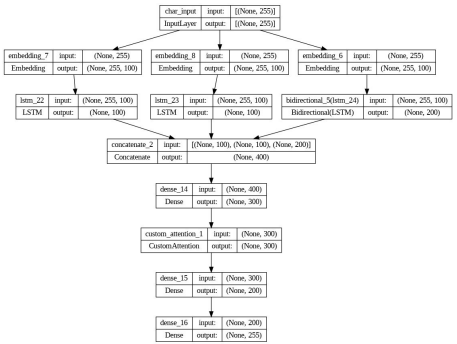
$$a_{ki} = soft\max(v(f(x_{ki}),\ h_{t-1}))$$

$$x_{ki} \in x$$

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



Structured attention mechanism



LPN network

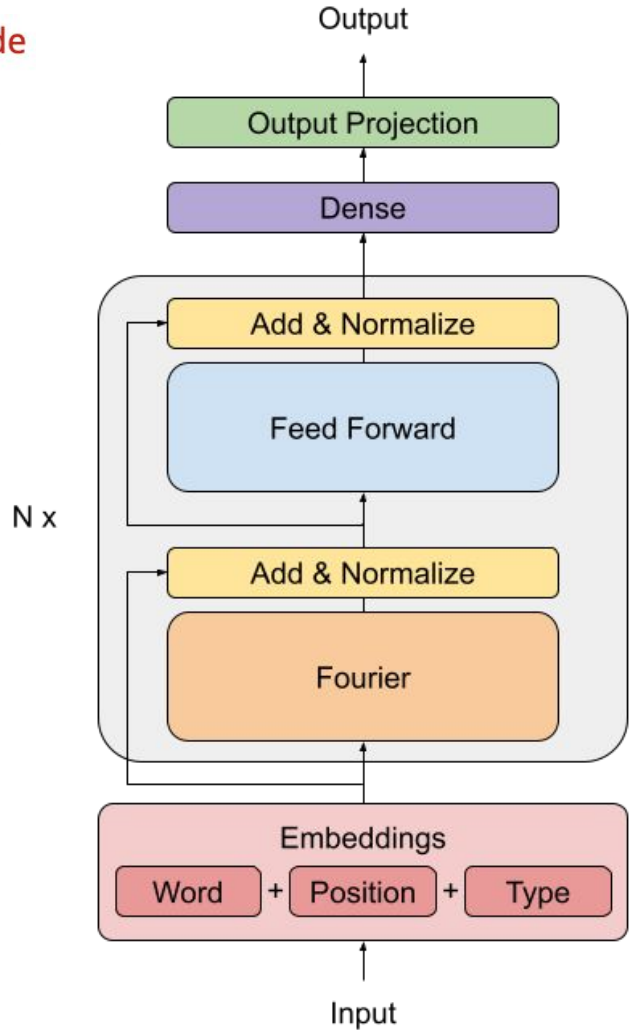
Related work. Fnet.

Author: Darshan Deshpande
Date created: 2021/10/05
Last modified: 2021/10/05

Text Generation using FNet is a method of generating text using a neural network with a novel attention mechanism called FNet. FNet replaces the traditional attention mechanism in the Transformer model with a more efficient linear attention mechanism. The model is trained on a large corpus of text data and can generate new text by sampling from its output distribution. The approach has the potential to improve the efficiency and effectiveness of language modeling and text generation tasks.

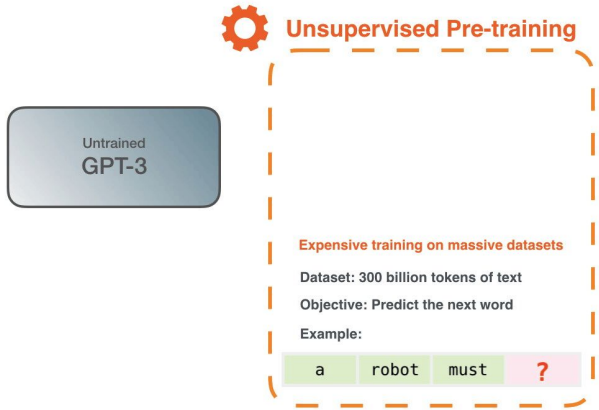
```
Epoch 17/20
9/9 [==============================] - 146s 16s/step - loss: 0.8880 - accuracy: 0.7818 - val_loss: 1.2723 - val_accuracy: 0.6963
Epoch 18/20
9/9 [==============================] - 148s 16s/step - loss: 0.8377 - accuracy: 0.7970 - val_loss: 1.3216 - val_accuracy: 0.6966
Epoch 19/20
9/9 [==============================] - 145s 16s/step - loss: 0.7966 - accuracy: 0.8083 - val_loss: 1.5412 - val_accuracy: 0.6683
Epoch 20/20
9/9 [==============================] - 145s 16s/step - loss: 0.7915 - accuracy: 0.8017 - val_loss: 1.3209 - val_accuracy: 0.7017
<keras.callbacks.History at 0x7f2fb42a7d00>
```

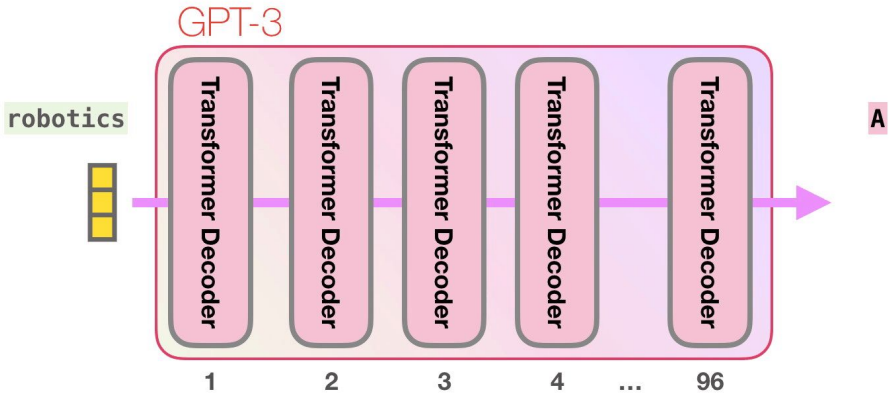
```
Epoch 18/20
9/9 [==============================] - 319s 35s/step - loss: 0.8670 - accuracy: 0.7824 - val_loss: 1.2878 - val_accuracy: 0.6990
Epoch 19/20
9/9 [==============================] - 319s 35s/step - loss: 0.8364 - accuracy: 0.7929 - val_loss: 1.3456 - val_accuracy: 0.6988
Epoch 20/20
9/9 [==============================] - 320s 36s/step - loss: 0.8022 - accuracy: 0.8010 - val_loss: 1.3412 - val_accuracy: 0.6966
<keras.callbacks.History at 0x7f0a2a7afe20>
```



GPT-3

GPT-3 is built on the Transformer architecture, which is a type of neural network designed to process sequential data like text. It has 96 layers in its largest version and is pre-trained on a large corpus of text using unsupervised learning to learn the underlying patterns and structure of language. It is then fine-tuned on specific tasks to improve its performance.





GPT-3

Train input message

“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 class for card game Hearthstone. You can do this according to these examples of input and output.”

3 examples of input and output

Test results

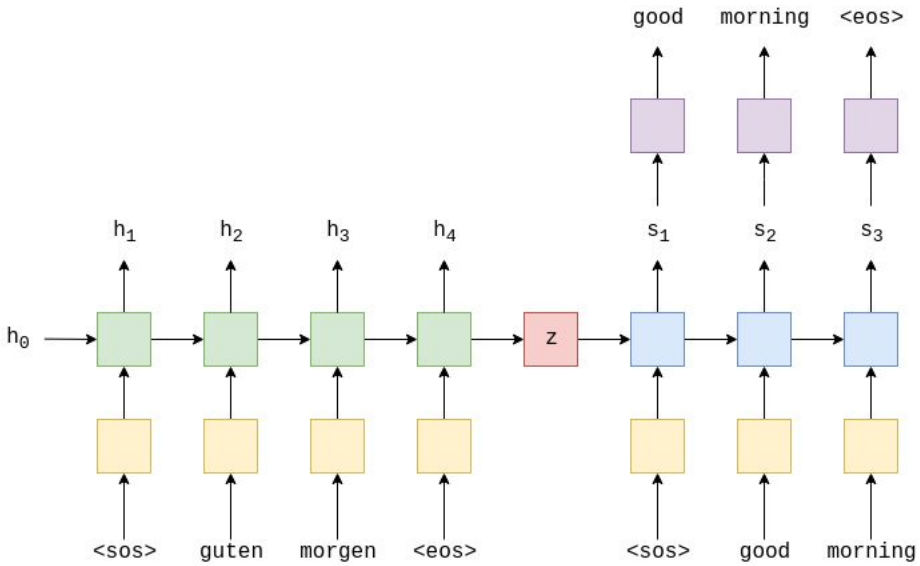
Floating Watcher NAME_END 4 ATK_END 4 DEF_END 5 COST_END -1 DUR_END Minion TYPE_END Warlock PLAYER_CLS_END Demon RACE_END Common RARITY_END Whenever your hero takes damage on your turn, gain +2/+2.

```
class FloatingWatcher(MinionCard):
    def __init__(self):
        super().__init__("Floating Watcher", 5, CHARACTER_CLASS.WARLOCK,
CARD_RARITY.COMMON, minion_type=MINION_TYPE.DEMON)
        self.bind("hero_damaged", self.hero_damaged)
        self._temp_attack = 4
        self._temp_health = 5

    def create_minion(self, player):
        return Minion(self._temp_attack, self._temp_health)

    def hero_damaged(self, player, amount):
        if player.game.current_player is player
          and player.hero is player.game.current_player.hero:
            self._temp_attack += 2
            self._temp_health += 2
            self.increase_health(2)
            self.change_attack(2)
```

Sequence model

Highlights:

- Introduced in paper “Sequence to Sequence Learning with Neural Networks” by Sutskever et al, 2014;
- Model architecture is widely used in natural language translation tasks;
- Deep LSTM with 4 layers;
- Deep LSTM outperforms shallow one;
- Reversing the source sentences increases BLEU score by ~17% (25.9 – 30.6) and decreases Perplexity by 18.97% (5.8 – 4.7) due to the introduction of many short term dependencies to the dataset;



Accuracy: 0.59

Attention model

Highlights:

- Introduced in paper “Neural Machine Translation by jointly learning to Align and Translate”, by Dzmitry Bahdanau, Kyung Hyun, Cho Yoshua Bengio, 2015;
- Points out that the use of a fixed-length vector is a bottleneck in improving the performance basic encoder–decoder architecture.
- Proposes approach which allows a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly;
- Encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation;
- Good at translating long sentences.

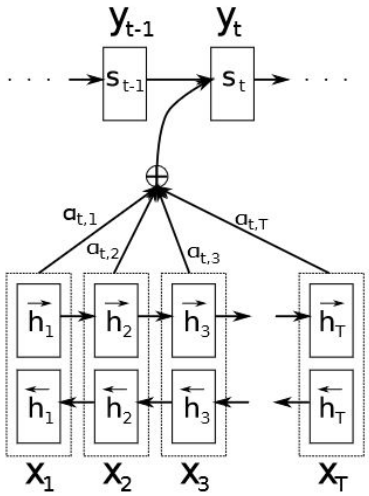


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Conducted experiments

Models:

- LPN - tested on a different attention mechanisms, architecture modifications;
- Gpt3 - not sufficient resources for training, chatGPT used for testing;
- Seq2Seq - trained simple lstm model;
- Attention - in progress;
- FNet - trained, accuracy 70% on our dataset.

Results

24.03

LPN network:
- attention layer - debugging
- other elements are finished

Seq2Seq model:
- architecture is finished

Attention model
- in progress

FNet model
- Finished and tested

GPT model
- Tested

By the 26.03

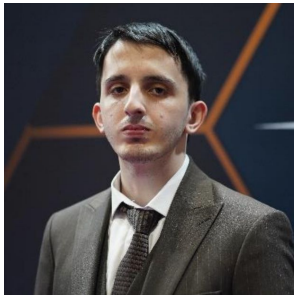
LPN network:
+ tested

Seq2Seq model:
+ tested

Attention model
+ tested

+ Report
+ Github repo

Team



Usman Tasuev, PE
U.Tasuev@skoltech.ru

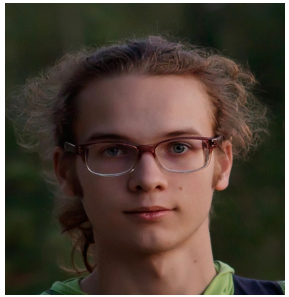
Contribution:

- Coding the main algorithm
- Preparing the GitHub Repo
- Preparing the section about paper

Albert Matsiev, PE
A.Matsiev@skoltech.ru

Contribution:

- Benchmark models (Sequence and Attention)
- Models evaluation



Kudryakov Ivan, ML
Ivan.Kudryakov@skoltech.ru

Contribution:

- Preparing report
- Literature overview

Emin Khatuev, PE
E.Khatuev@skoltech.ru

Contribution:

- Finding current applicable method
- Implementation on our dataset
- Checking related works