



DEPARTMENT OF DIGITAL SYSTEMS



NCSR DEMOKRITOS  
INSTITUTE OF INFORMATICS AND  
TELECOMMUNICATIONS

# DL Project Presentation

## Chat Bot Answering Python Questions


**Authors:** Varsou Penny Gkatsis Vasileios

# Introduction

- Chat bots are becoming more and more used in everyday tasks.
- Create an AI assistant able to answer simple or complicated questions.
- Get data from StackOverflow website.
- The challenge is to capture the technical nature of Qs&As.



The project consists of the following steps:

- Download and pre-process data[7].
  - Implement a seq2seq model.
  - Train encoders and decoders using mini-batches.
  - Use greedy search decoding.
  - Interact and evaluate the chatbot.
- 

# Pre-Processing Part1

- Keep all questions with at least one answer.
- Choose the answer with the best vote score.
- Remove any question that requires code in order to be answered.

## Pre-Processing Part2

- Clear all sentences from punctuation and special characters.
- Remove html and markdown tags.
- Filter sentences with length greater than a given value.
- Filter pairs of Qs&As containing words with a frequency less than a given value.

# Prepare Data for Models

- Create torch tensors to feed the models.
- Train with mini-batches instead of 1 sentence at a time.
  - Improve convergence
  - Create tensors of shape (max\_length, batch\_size)
- Tensors are zero-padded.
- Create tensors of length for each sentence in the batch.
- Create mask tensors.

# Models

- The main model for our project is a Seq2Seq.
- Consists of 2 RNNs[3]:
  - One acts as an Encoder.
  - The other acts as a Decoder.

# Encoder

- The Encoder iterates through the input sentence one word at a time, at each time step outputting an:
  - Output vector -> Gets recorded.
  - Hidden state vector. -> The hidden state vector is passed to the next time step.
- Bidirectional variant of the multi-layered Gated Recurrent Unit [4],
  - Two independent RNNs.
  - One that is fed the input sequence in normal sequential order.
  - And one that is fed the input sequence in reverse order.
  - The outputs of each network are summed at each time step. Using a bidirectional GRU.



# Decoder

- The decoder RNN generates the response sentence in a token-by-token fashion using:
  - context vectors
  - and internal hidden statesfrom the encoder to generate the next word in the sequence.

In order to minimize information loss during encoding process we will use the **Global attention** mechanism by [5] which improved upon Bahdanau et al.'s [6] attention mechanism.

# Attention Mechanisms

- Key differences:
  - Global attention considers all of the encoder's hidden states.
  - Local attention considers the encoder's hidden state from the current time step.
  - Global attention calculates attention weights, using the hidden state of the decoder from the current time step.
  - Bahdanau et al.'s attention calculation requires knowledge of the decoder's state from the previous time step.
  - Luong et al. provides various methods to calculate the attention energies between the encoder output and decoder output which are called "score functions".

# Seq2Seq Flow

1. Get embedding of current input word.
2. Forward through unidirectional GRU.
3. Calculate attention weights from the current GRU output.
4. Multiply attention weights to encoder outputs to get new "weighted sum" context vector.
5. Concatenate weighted context vector and GRU output using Luong.
6. Predict next word.
7. Return output and final hidden state.

# Masked Loss

Since we are dealing with batches of padded sequences, we cannot simply consider all elements of the tensor when calculating loss.

We need to calculate our loss based on our decoder's output tensor, the target tensor, and a binary mask tensor describing the padding of the target tensor. Our masked loss function calculates the average negative log likelihood of the elements that correspond to a *1* in the mask tensor.

# Training Procedure

1. Forward pass entire input batch through encoder.
2. Initialize decoder inputs as SOS\_token, and hidden state as the encoder's final hidden state.
3. Forward input batch sequence through decoder one time step at a time.
4. If teacher forcing: set next decoder input as the current target else set next decoder input as current decoder output.
5. Calculate and accumulate loss.
6. Perform backpropagation.
7. Clip gradients.
8. Update encoder and decoder model parameters.

During the training process we use a these tricks to aid in convergence:

- **Teacher forcing:** At some probability, set by **teacher\_forcing\_ratio**, we use the current target word as the decoder's next input rather than using the decoder's current guess.
- **Gradient clipping.** Commonly technique for countering the “exploding gradient” problem. In essence, by clipping or thresholding gradients to a maximum value, we prevent the gradients from growing exponentially and either overflow (NaN), or overshoot steep cliffs in the cost function.

# Evaluation

## **Decoding Method**

1. Forward input through encoder model.
2. Prepare encoder's final hidden layer to be first hidden input to the decoder.
3. Initialize decoder's first input as SOS\_token.
4. Initialize tensors to append decoded words to.
5. Iteratively decode one word token at a time:
  1. Forward pass through decoder.
  2. Obtain most likely word token and its softmax score.
  3. Record token and score.
  4. Prepare current token to be next decoder input.
6. Return collections of word tokens and scores.

## Greedy decoding

- Greedy decoding is the decoding method that we use during training when we are **NOT** using teacher forcing.
- For each time step we choose the word from `decoder_output` with the highest softmax value.
- This decoding method is optimal on a single time-step level.



## Evaluation Process

- Format the sentence to be evaluated as an input batch of word indexes with *batch\_size==1*.
- Create a **lengths** tensor which contains the length of the input sentence.
- Obtain the decoded response sentence tensor using **GreedySearchDecoder**.
- Convert the response's indexes to words and return the list of decoded words.

When chatting with the bot this evaluation process is followed in order for it to respond.

# Results

## Final Results.

- Not good.
- A few descent answers.
- Majority of answers is not useful.
- Nonsense is included (e.g. repetitions).

# Some Good Examples

> How to read from csv

Bot: you can use the csv module

>

> How to install Python on Windows

Bot: you can use the py2exe module

>

# Some Bad Examples

> Are arrays or lists better?

Bot: is a good way to do it its support library to your question its examples examples

> How to create an api

Bot: you can use

>

> How to create a class?

Bot: you can use swig to create a custom class

> How to concatenate two strings?

Bot: i think you need to use the pexpect module

>

> How to read from file?

Bot: you can use

> How to read from a txt file?

Bot: you can use

The slide features decorative curved lines in the top-left and bottom-right corners. These lines are composed of multiple overlapping layers in shades of light green and light blue, creating a sense of depth and movement.

## Further Work

- Handle code blocks.
- Use more sentences.
- Use a better GPU.

# References

1. ChatbotTutorial by Matthew Inkawhich  
[https://pytorch.org/tutorials/beginner/chatbot\\_tutorial.html](https://pytorch.org/tutorials/beginner/chatbot_tutorial.html)
2. Pytorch Chatbot by Wu,Yuan-Kuei  
<https://github.com/ywk991112/pytorch-chatbot>
3. Sutskever et al.  
<https://arxiv.org/abs/1409.3215>
4. Cho et al.  
<https://arxiv.org/pdf/1406.1078v3.pdf>
5. Luong et al.  
<https://arxiv.org/abs/1508.04025>
6. Bahdanau et al.  
<https://arxiv.org/abs/1409.0473>
7. Python Questions from Stack Overflow  
<https://www.kaggle.com/stackoverflow/pythonquestions>