NLP Project Technical Report

Team Members

Students of B20-AI-01:

- Andrey Vagin
- Artyom Chernitsa
- Sergey Golubev

Project Goal and Steps

To build a model for accurate automatic speech recognition (ASR).

Our solution included the following steps.

- Data collection
- Data processing
- Choice of model architecture and implementation
  - Implementation of a neural network
  - Implementation of decoders
- Training and evaluation
- Deployment

The steps are described in details below.

Data Collection

We used two datasets for training the model.

1. SpeechCommands Dataset

- A simple and small dataset. We used it as a starting point, for quick training and evaluation of te architecture.
- Contains approximately 64.700 utterances of 35 small English words.
- **Sample audio link**. The 35 words:

> *backward, bed, bird, cat, dog, down, eight, five, follow, forward, four, go, happy, house, learn, left, marvin, nine, no, off, on, one, right, seven, sheila, six, stop, three, tree, two, up, visual, wow, yes, zero*

- Recording length: ~1 sec.
- Total length: ~17 hours.
- Dataset size: 2,1 GB.

2. LibriSpeech Dataset

- A bigger and more complex dataset. We used it for the final version of the model.
- Contains English speech data derived from audiobooks.
- **Sample audio link** and its transcription:

> *MY FATHER WAS AND STILL IS RECEVEUR GENERAL AT C HE HAS A GREAT REPUTATION THERE FOR LOYALTY THANKS TO WHICH HE WAS ABLE TO FIND THE SECURITY WHICH HE NEEDED IN ORDER TO ATTAIN THIS POSITION*

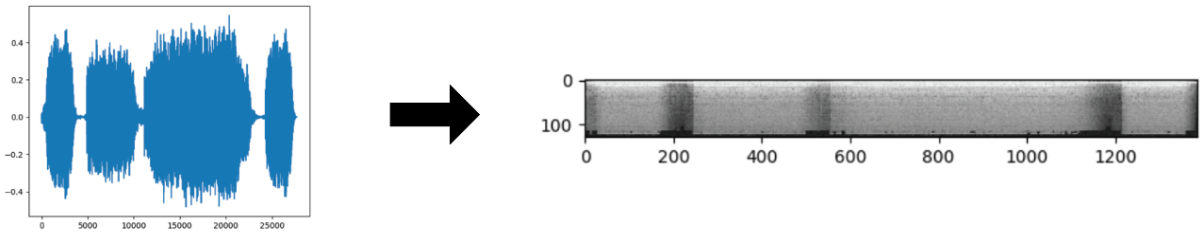
- Recording length: ~15 secs.
- Total length: ~100 hours.
- Dataset size: 7 GB.

Data Processing

The audio recording processing pipeline included the following steps:

1. **Conversion**

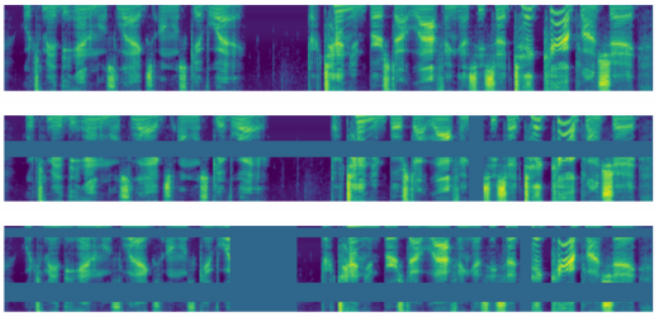
   We converted raw audio waves into Mel Spectrograms:



   This step allowed us to treat the recordings as images, which is very convenient for use of CNNs.

2. **Augmentations**

   We applied the SpecAugment [2] technique to improve the model generalization ability. SpecAugment cuts out random consecutive blocks of time and frequency from the spectrograms:

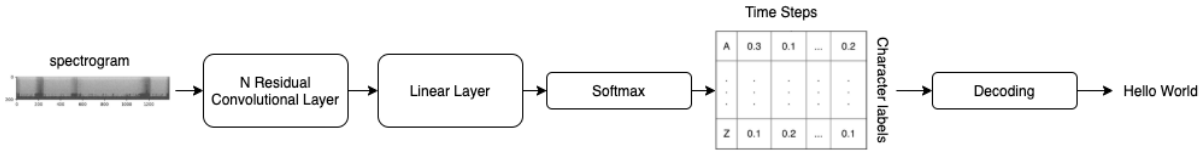


Review of Methods and Models

We discovered several architectures for ASR task:

- **CNN**

  Convolutional NN builds feature maps by applying convolutions to the spectrograms images. Feature maps are converted to a feature vector, that is fed into CTC Loss (the CTC Loss is described further).

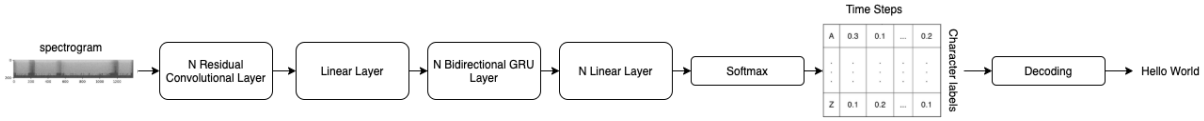


CNN-based architecture for ASR

  This is a simple solution, however this architecture is not powerful enough to perform well on natural speech recognition task.

- **CNN + RNN**

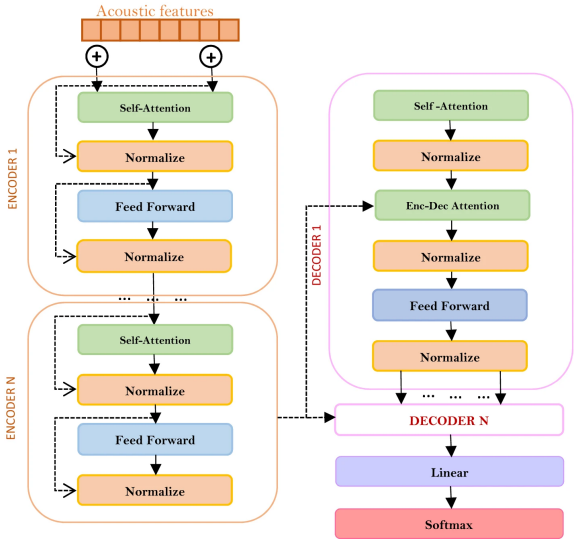
  The same CNN builds feature maps. But the feature vector is fed into a recurrent NN. Since ASR is a seq2seq task, adding an RNN significantly increases the model performance.



Hybrid CNN + RNN architecture for ASR

  We found this hybrid model to be a sweet spot between weak CNN-based and heavy transformer-based architectures.

- **Transformers**



Transformer-based architecture for ASR

Architecture and Implementation

CNN

CNNs are good at extracting abstract features. We use a CNN to extract features from the audio spectrograms.

- We added several **Res-CNN** layers to the model. It is a CNN with residual connections. They make the loss surface “flatter” and help the model learn faster and generalize better.

RNN

RNNs are good at sequence modeling that is useful when working with audio.

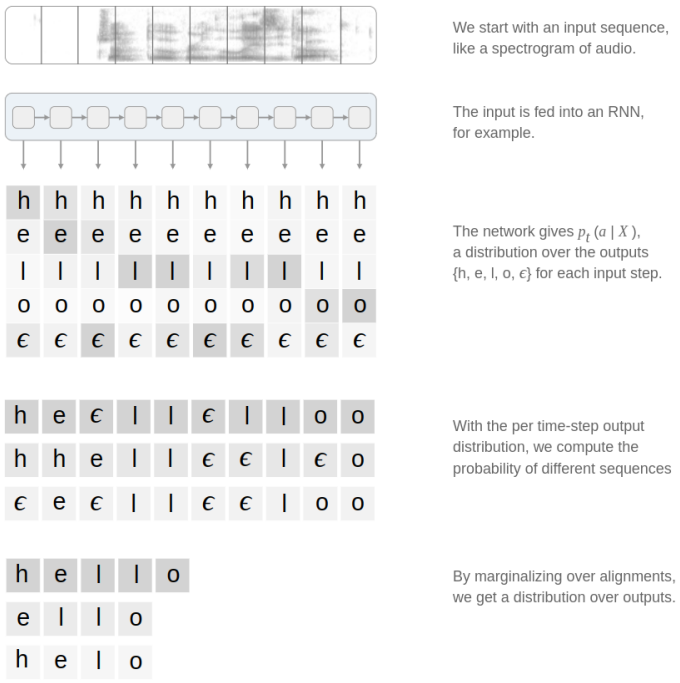
- We implemented a **Bidirectional RNN**. BiRNN takes into consideration the context of both previous and next cells.
- We trained different models with **GRU** and **LSTM** units. The results and comparison are introduced below.

CTC Loss

CTC Loss [3] is a complex loss function for sequence problems. It is widely used in ASR and handwriting recognition tasks.

- Its main advantage is that it’s **alignment-free**. It means the loss is computed without the alignment between the input and the output. We only need a recording and its subscription to train the model.
- CTC Loss is **robust to change in the absolute and relative lengths of R and T** sequences (recording and its transcription).
- **CTC Loss input**: an input sequence R, represented as an output of an RNN.

  CTC Loss output: probability distribution of all possible output sequences T

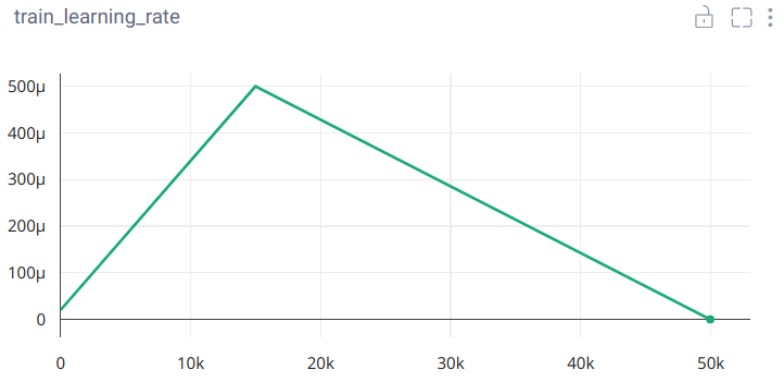


CTC Loss Pipeline

Learning Rate Scheduler Optimizer

We used **AdamW optimizer** and **One Cycle Learning Rate Scheduler** to train our models.

- **Adam** optimizer is easy to use and provides quick convergence. **AdamW** is simply Adam with fixed weight decay implementation.
- **One Cycle LR Scheduler** introduced in the paper [4] allows to train NNs faster, preserving the generalization abilities. Training starts with a low LR, that linearly reaches a maximum value (hyperparameter), and then linearly decays to 0.



- Adam does not generalize as well as **SGD** optimizer. We tried to use SGD with both One Cycle and Step LR schedulers, however had no success.

Evaluation and Conclusion

Metrics

We used two metrics to evaluate model performance. Both compute **Levenshtein distance** between transcription and prediction strings.

1. **Character Error Rate (CER)**

   Compares transcription and prediction texts **in char-level**.

   $$CER = \frac{S_c + D_c + I_c}{N_c}$$

   where S_c is the number of characters substituted,
   D_c is the number of characters deleted,
   I_c is the number of characters inserted,
   N_c is the number of characters in the transcription.

2. **Word Error Rate (WER)**

   Compares transcription and prediction texts **in word-level**. Uses Levenshtein distance to compute the formula:

   $$WER = \frac{S_w + D_w + I_w}{N_w}$$

   where S_w is the number of words substituted,
   D_w is the number of words deleted,

I_w is the number of words inserted,
N_w is the number of words in the transcription.

Trained Models

We trained three different models on both datasets. The difference between the models is shown in the table below:

Model Name	# CNN Layers	CNN Feature Vector Dim.	RNN Unit Type	# RNN Layers	RNN Hidden Vector Dim.	# Parameters
3-GRU	3	128	GRU	3	512	14.3 m
3-LSTM	3	128	LSTM	3	512	18.5 m
5-LSTM	3	128	LSTM	5	512	31.1 m

Evaluation Results

Note: WER is only evaluated for LibriSpeech dataset, since SpeechCommands dataset contains exactly one word per recording.

Model Name	Dataset	# Epochs	CER	WER
3-GRU	simple (SpeechCommands)	20	4.6%	—
3-LSTM	simple	20	6.2%	—
5-LSTM	simple	7	6.1%	—
3-GRU	complex (LibriSpeech)	30	26.9%	91.3%
5-LSTM	complex	14	17.9%	68.5%

Note: number of epochs varies as it was chosen empirically and individually for each model, in order to avoid overfitting.

Good results in speech recognition task require huge amounts of data and computational resources. To get SOTA results, one will need to train a model on thousands of hours of audio, using a cluster of GPUs.

Since we are limited in time and resources, we consider the obtained results to be quite good.

Space for Improvements

- **Bigger dataset** should lead to increase in performance and generalization ability.
  - We used only a subset of LibriSpeech dataset (100 hours) to train the models. There is a version with 360 hours of audio.
  - We can build a mixed dataset of several ASR datasets. This step could decrease performance yet increase generalization ability.
- Reach convergence with **SGD optimizer**.
- Perform a **hyperparameter search**.
- Implement a more powerful **decoder**:
  - Greedy decoder with heuristics
  - Beam search decoder
  - Language model based decoder

Team Members Contribution

Artyom:

- Research on available datasets. Data collection. Data processing (characters indexing, CER, WER). Research on audial data augmentation approaches.

Andrey:

- Research on ASR approaches and architectures. Baseline model implementation. Model evaluation. Working with optimizer and scheduler. Bonus: adversarial attacks on the model.

Sergey:

- Heuristics for greedy decoder. Extension to LSTM-based RNN. Training the models. Deployment. Reporting.

Github Link

https://github.com/pigeongcc/speech-recognition/tree/main

References

1. Building an End-to-End Speech Recognition Model in PyTorch
2. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition
3. Sequence Modeling With CTC
4. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates