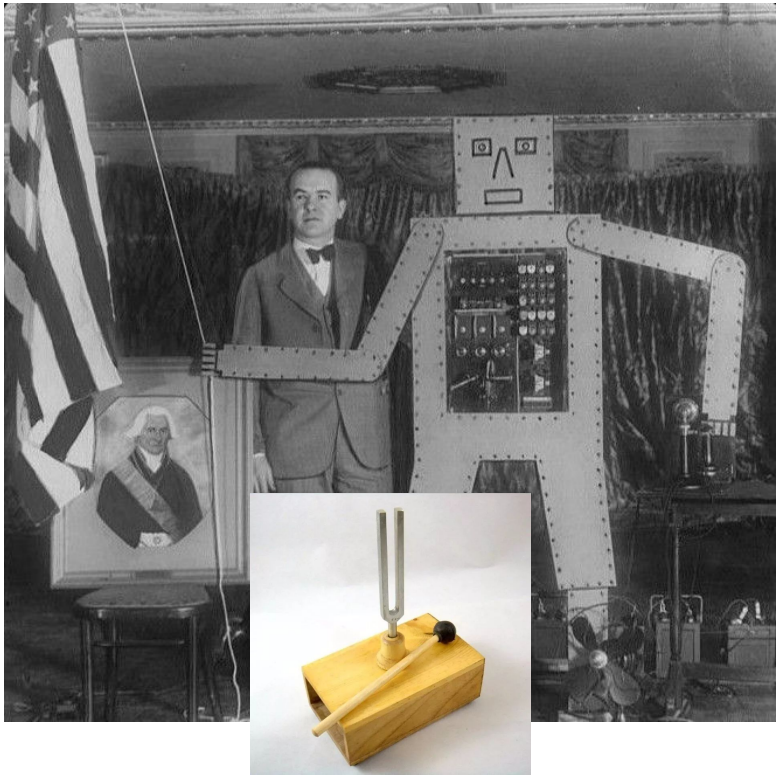
Speech Recognition

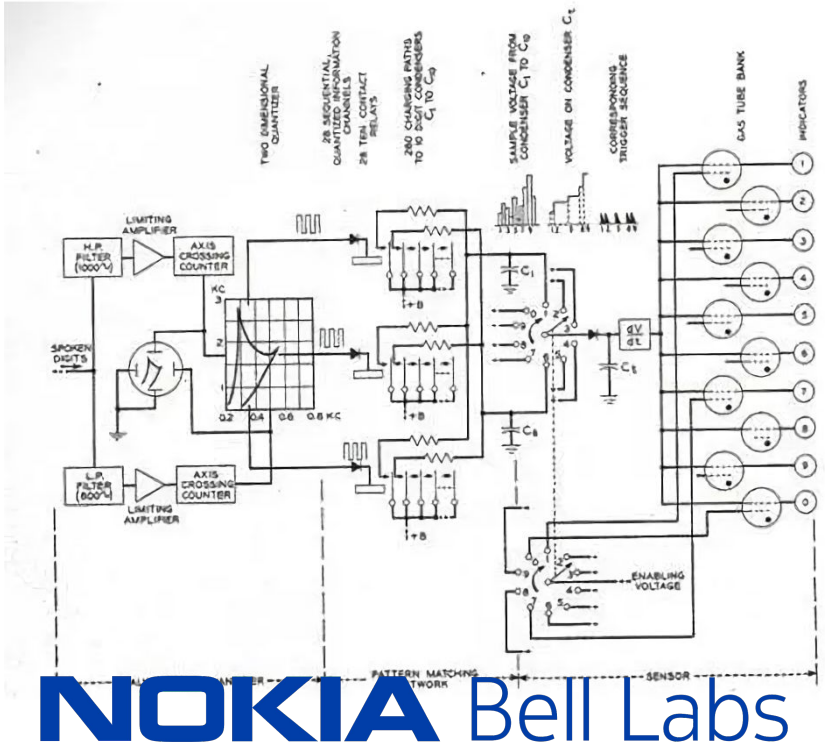
Ваньков Тимур, БПМИ-181

План презентации

1. Что такое Speech Recognition и зачем он нужен
2. Представление звука и его обработка
3. RNN + CTC
4. Deep Speech
5. Speech Recognition Вконтакте

Начало





NOKIA Bell Labs

Ближе к нашему времени

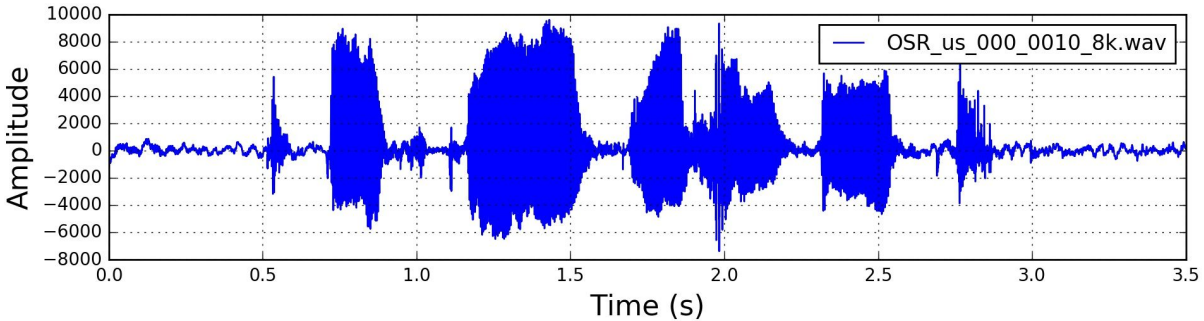
- 1960 - 200 слов
- 1971 - 1011 слов
- 1980 - 20M слов с любым голосом
- 1996 - первая коммерческая программа, способная различать беспрерывный поток естественной речи - IBM MedSpeak/Radiology
- 1997 - первый универсальный движок распознавания естественной речи
- Наше время - RNN для понимания распознанных слов, также предсказания слов в рамках контекста если оно не было распознано и CTC(нейросетевая темпоральная классификация) - для обучения, выделяет отдельные фонемы и расставляет их в нужном порядке и выявляет слова.

Зачем все это надо

- Инклюзивность
- Голосовой интерфейс
- Запись слов в профессиональных отраслях
- Распознавание голосовых сообщений в чатах

Как выглядит звук

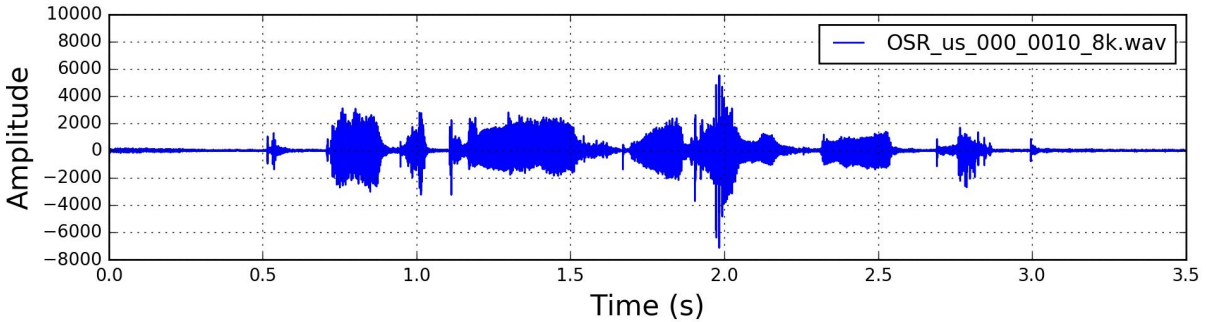
```
1.  import numpy
2.  import scipy.io.wavfile
3.  from scipy.fftpack import dct
4.
5.  sample_rate, signal = scipy.io.wavfile.read('OSR_us_000_0010_8k.wav')  # File assumed to be in the same directory
6.  signal = signal[0:int(3.5 * sample_rate)]  # Keep the first 3.5 seconds
```



Предварительный акцент

```
1.    pre_emphasis = 0.97
2.    emphasized_signal = numpy.append(signal[0], signal[1:] - pre_emphasis * signal[:-1])
```

$$y(t) = x(t) - \alpha x(t - 1)$$



Framing

```
1.  # Популярные настройки
2.  frame_size = 0.025
3.  frame_stride = 0.01
4.
5.  # Считаем длину фрейма и его шаг и остальные данные
6.  frame_length, frame_step = frame_size * sample_rate, frame_stride * sample_rate  # Convert from seconds to samples
7.  signal_length = len(emphasized_signal)
8.  frame_length = int(round(frame_length))
9.  frame_step = int(round(frame_step))
10. num_frames = int(numpy.ceil(float(numpy.abs(signal_length - frame_length)) / frame_step))  # Make sure that we have
    at least 1 frame
11.
12. # Получаем сами фреймы
13. pad_signal_length = num_frames * frame_step + frame_length
14. z = numpy.zeros((pad_signal_length - signal_length))
15. pad_signal = numpy.append(emphasized_signal, z) # Pad Signal to make sure that all frames have equal number of
    samples without truncating any samples from the original signal
16.
17. indices = numpy.tile(numpy.arange(0, frame_length), (num_frames, 1)) + numpy.tile(numpy.arange(0, num_frames *
    frame_step, frame_step), (frame_length, 1)).T
18. frames = pad_signal[indices.astype(numpy.int32, copy=False)]
```

Хемминг и Фурье и Filter Banks

Окно Хемминга

```
frames *= numpy.hamming(frame_length)
# frames *= 0.54 - 0.46 * numpy.cos((2 * numpy.pi * n) / (frame_length - 1))  # Explicit Implementation **
```

Преобразование фурье и спектр мощности

```
mag_frames = numpy.absolute(numpy.fft.rfft(frames, NFFT))  # Magnitude of the FFT
pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))  # Power Spectrum
```

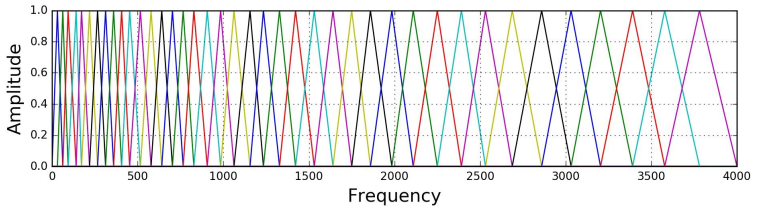
$$P = \frac{|FFT(x_i)|^2}{N}$$

Filter Banks

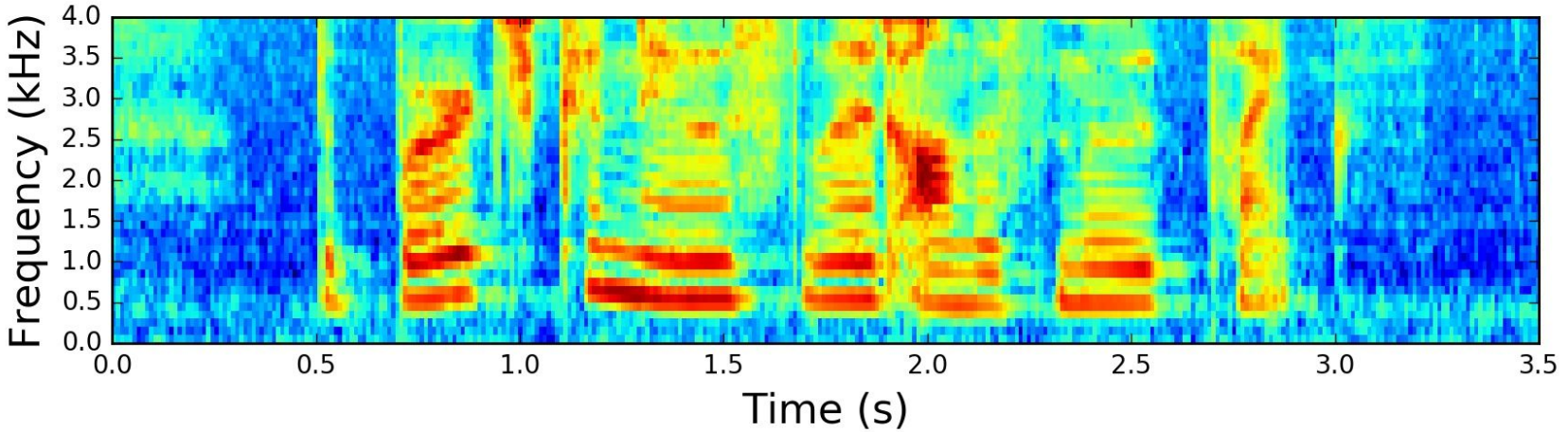
```
low_freq_mel = 0
high_freq_mel = (2595 * numpy.log10(1 + (sample_rate / 2) / 700))  # Convert Hz to Mel
mel_points = numpy.linspace(low_freq_mel, high_freq_mel, nfilt + 2)  # Equally spaced in Mel scale
hz_points = (700 * (10**(mel_points / 2595) - 1))  # Convert Mel to Hz
bin = numpy.floor((NFFT + 1) * hz_points / sample_rate)

fbank = numpy.zeros((nfilt, int(numpy.floor(NFFT / 2 + 1))))
for m in range(1, nfilt + 1):
    f_m_minus = int(bin[m - 1])   # left
    f_m = int(bin[m])             # center
    f_m_plus = int(bin[m + 1])    # right

    for k in range(f_m_minus, f_m):
        fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])
    for k in range(f_m, f_m_plus):
        fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])
filter_banks = numpy.dot(pow_frames, fbank.T)
filter_banks = numpy.where(filter_banks == 0, numpy.finfo(float).eps, filter_banks)  # Numerical Stability
filter_banks = 20 * numpy.log10(filter_banks)  # dB
```

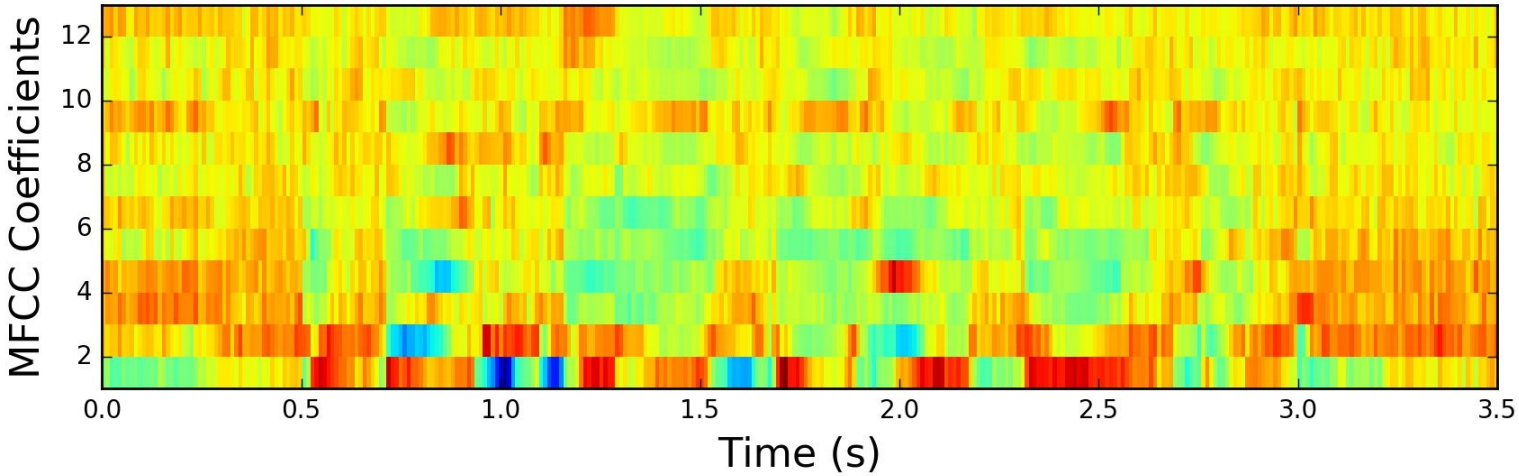


Спектрограмма

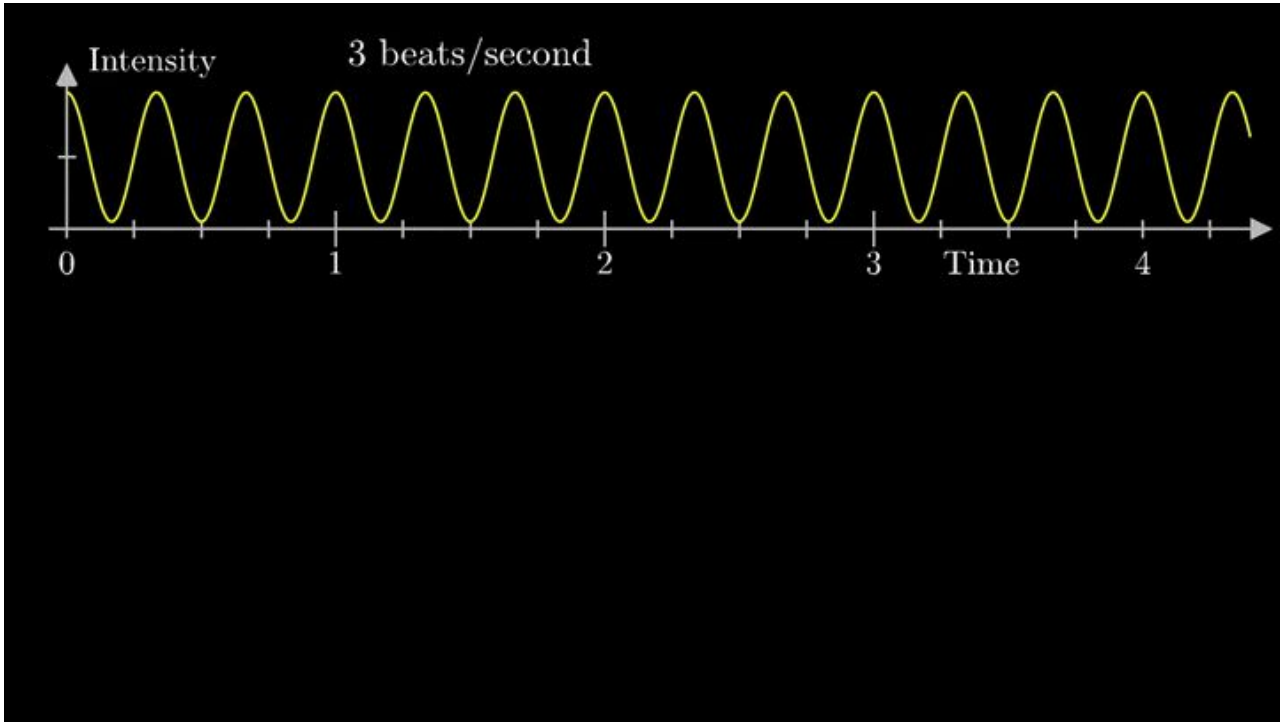


Спектрограмма

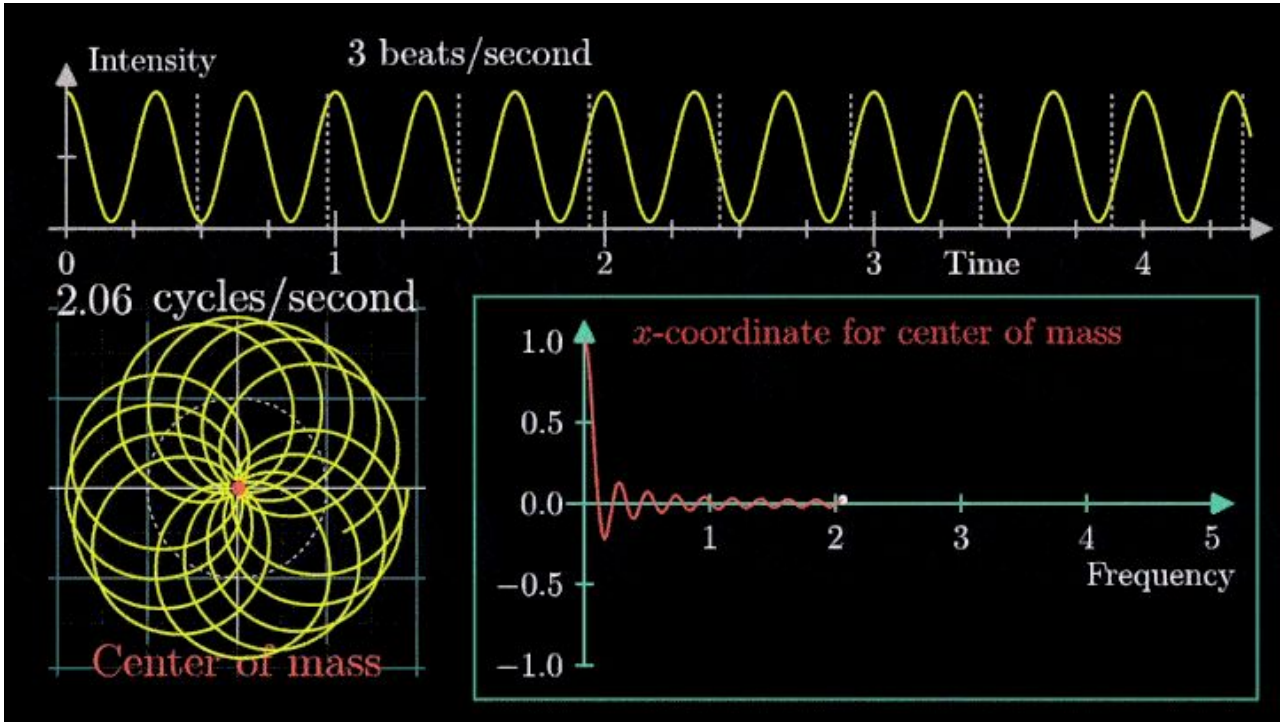
```
1.    mfcc = dct(filter_banks, type=2, axis=1, norm='ortho')[:, 1 : (num_ceps + 1)] # Keep 2-13
```



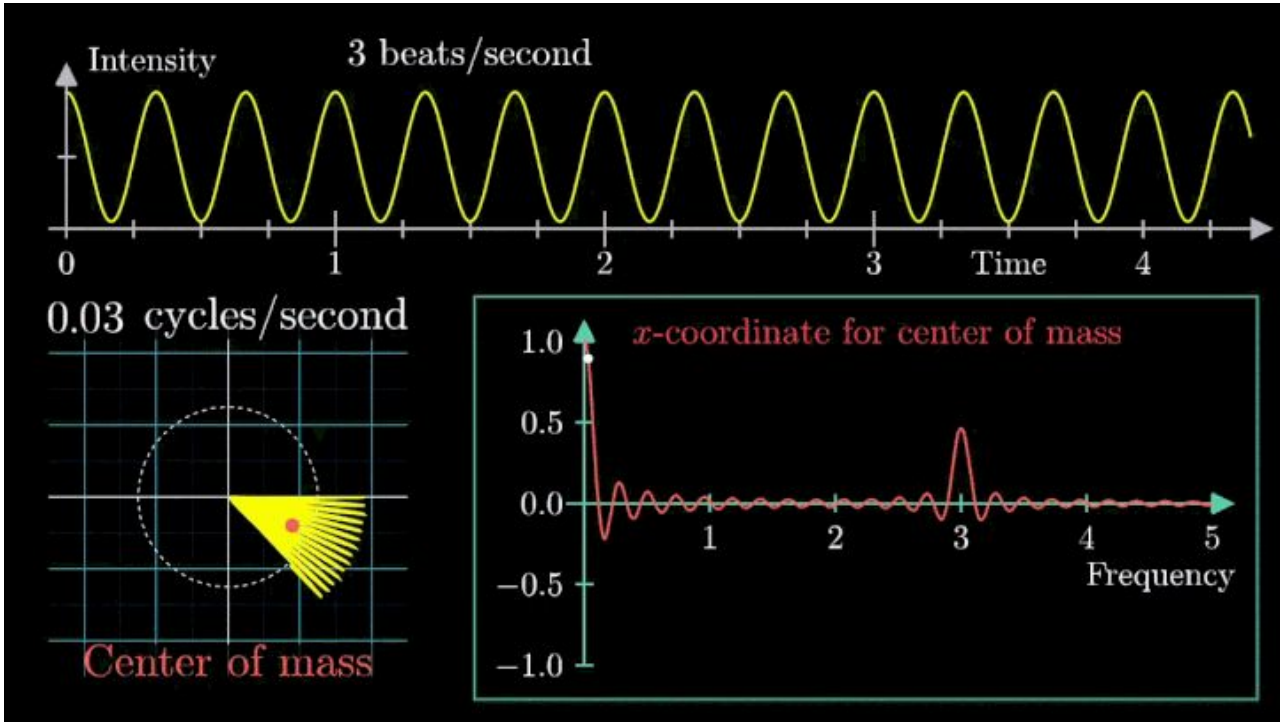
Преобразование Фурье



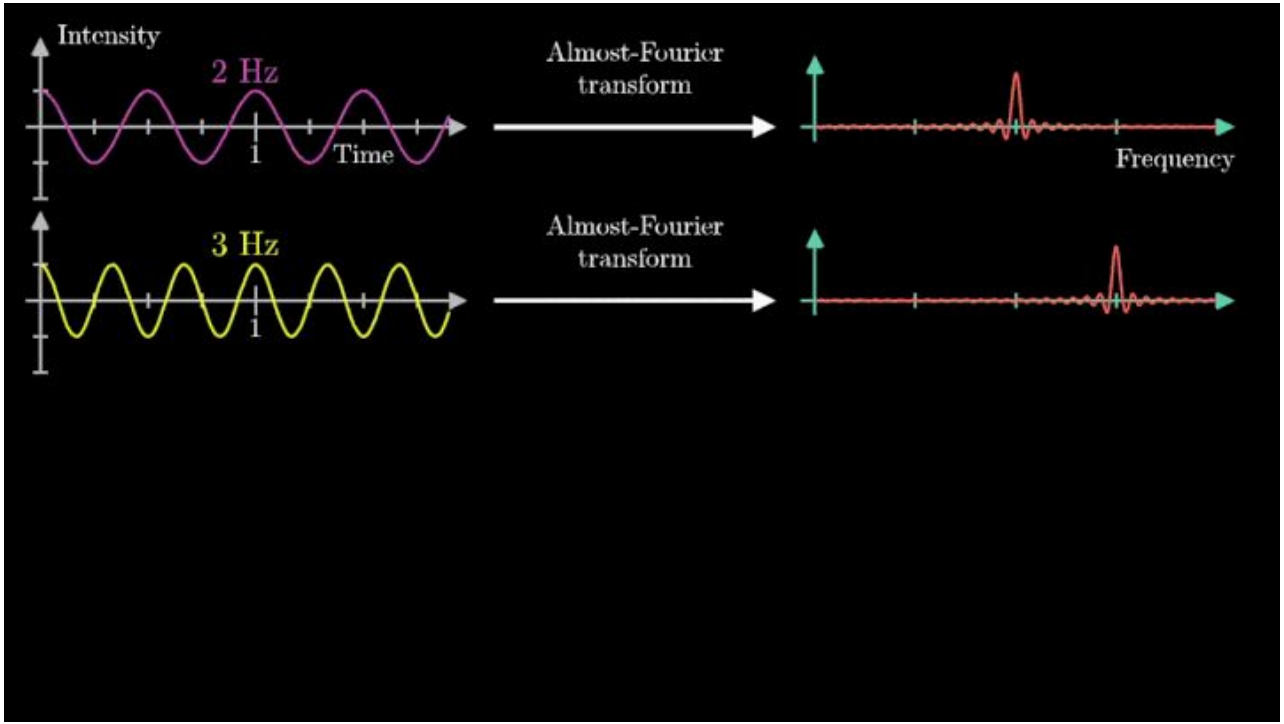
Преобразование Фурье



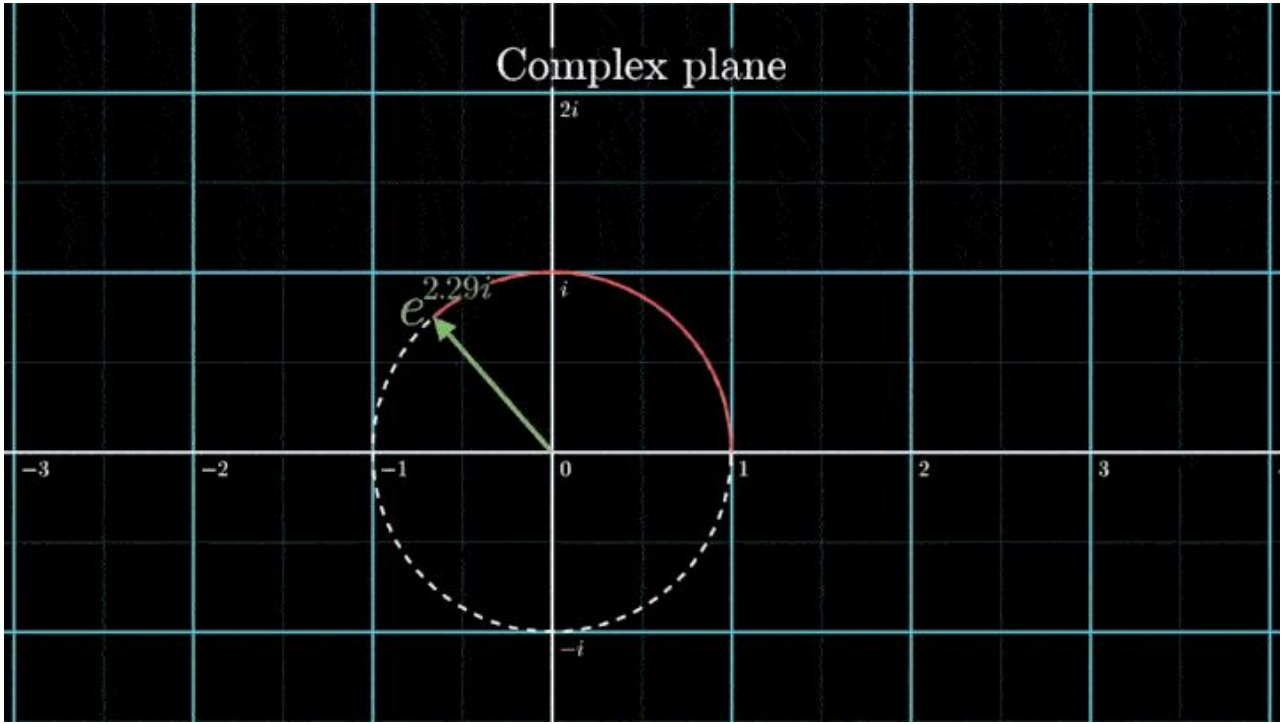
Преобразование Фурье



Преобразование Фурье



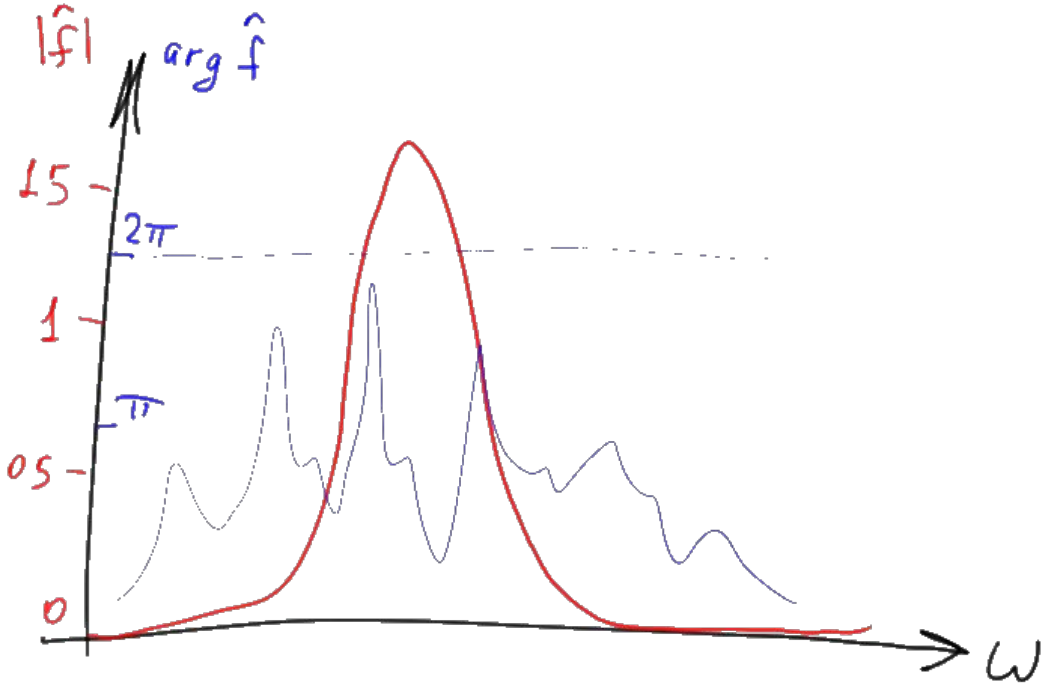
Преобразование Фурье



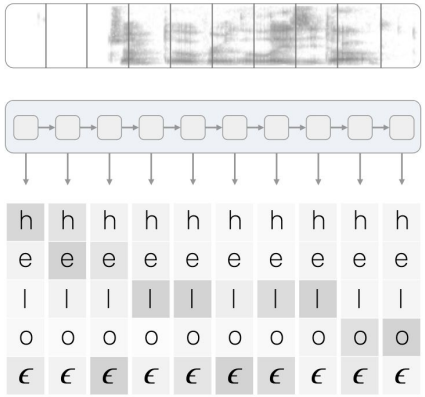
$$\frac{1}{N}\sum_{k=1}^{N} g(t_k)e^{-2\pi i f t_k}.$$

$$\frac{1}{t_2 - t_1}\int_{t_1}^{t_2} g(t)e^{-2\pi i f t}dt,$$

Преобразование Фурье



CTC Loss



We start with an input sequence, like a spectrogram of audio.

The input is fed into an RNN, for example.

The network gives $p_t(a \mid X)$, a distribution over the outputs {h, e, l, o, ϵ} for each input step.

With the per time-step output distribution, we compute the probability of different sequences

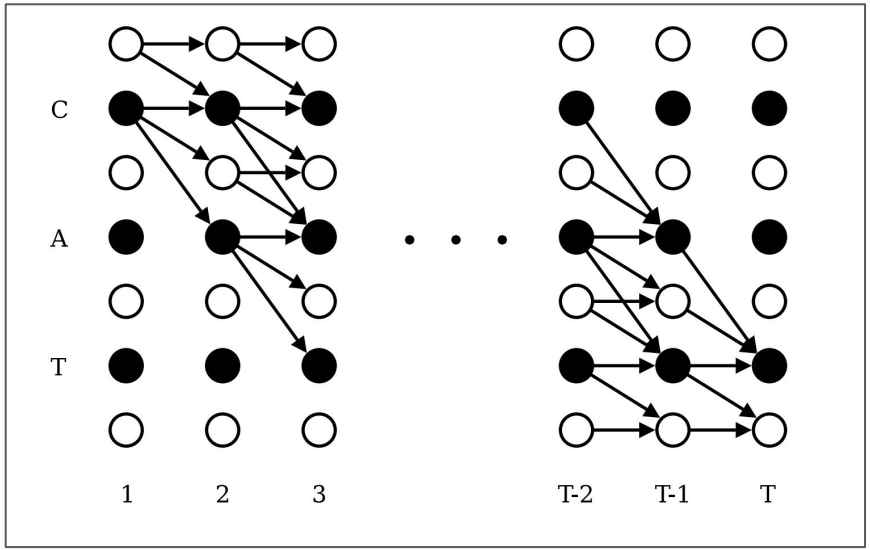
By marginalizing over alignments, we get a distribution over outputs.

$$p(Y \mid X) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X)$$

The CTC conditional **probability**

marginalizes over the set of valid alignments

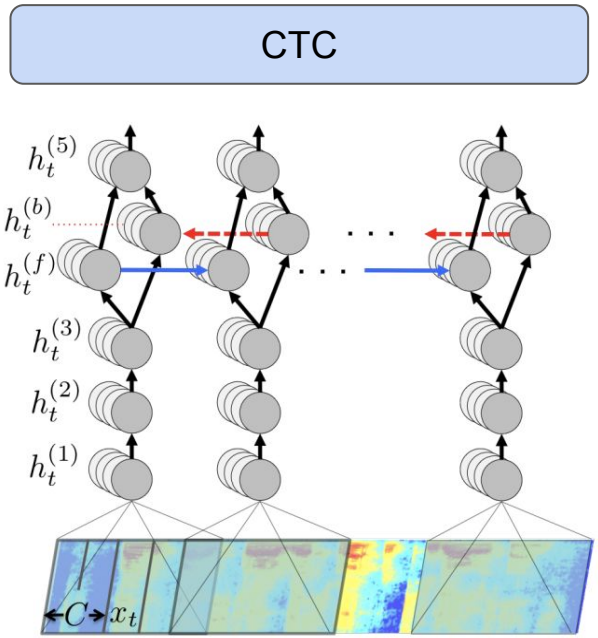
computing the **probability** for a single alignment step-by-step.



Ускорения обычной модели

Оптимизации:

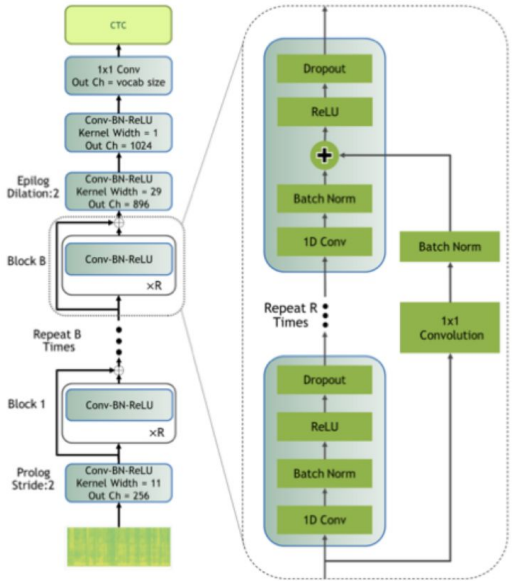
1. Data parallelism
2. Model parallelism
3. Striding



Быстрая акустическая модель

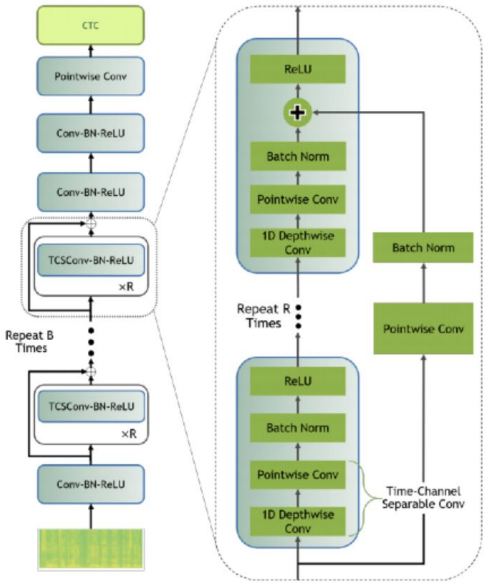
1. PyTorch + TorchSrcipt to cpp
2. Препроцессинг
   2.1. Вход: сырой звук - амплитуда колебаний
   2.2. Pre-Emphasis filter: усиливаем высокие частоты
   2.3. Фрейминг: разрезаем дорожку на перекрывающиеся фреймы
   2.4. Выполняем оконное преобразование Фурье на каждом фрейме и вычисляем power spectrogram
   2.5. На основе mel filter bank получаем mel spectrogram. Нормализуем
3. Акустическая модель
   3.1. CNN QuartzNet x ContexNet
   3.2. CTC/SpecCutout/Novograd + Cosine LR/Balanced Distributed Sampler для обучения

Jasper (NVIDIA, 2019)



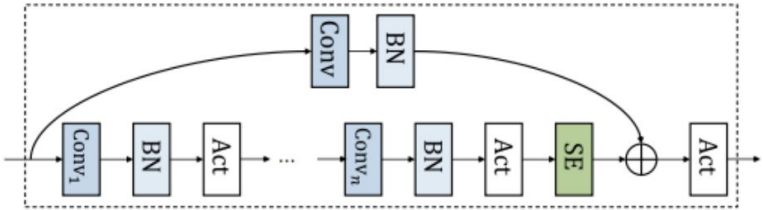
https://arxiv.org/pdf/1904.03288.pdf

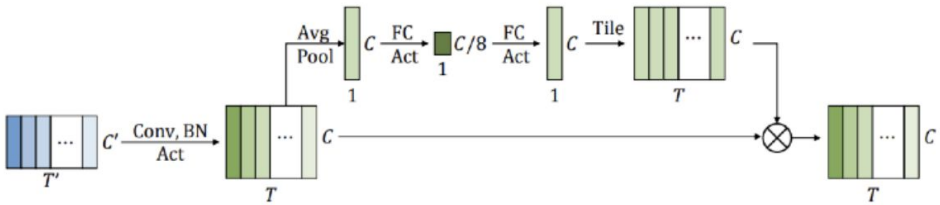
QuartzNet (NVIDIA, 2019)



https://arxiv.org/pdf/1910.10261.pdf

ContextNet (Google, 2020)





https://arxiv.org/pdf/2005.03191.pdf

Орфография и пунктуация

1. Языковая модель
   1.1. basic beam search decoder - выдает результаты из ctc лосса
   1.2. n-gram model - угадываем слова из предыдущих
   1.3. text normalization - нормализуем текст
2. Пунктуационная модель

_привет	1,	_Привет,
_со	1	_Со
_ня	0!	_ня!
_пойдем	1	_Пойдем
_сегодня	0	_сегодня
_в	0	_в
_кин	0	_кин
_чик	0?	_чик?

Инфраструктура

1. Требования
   1.1. 200кк аудиосообщений в день
   1.2. Несколько секунд на ответ
   1.3. 90 видеокарт
2. Алгоритм
   2.1. Забрать из очереди ссылку на аудиосообщение
   2.2. Скачать его
   2.3. Раздекодить
   2.4. Прогнать через все модели
   2.5. Отправить текст в результирующую очередь
3. Основные способы решения
   3.1. Группировать аудиоданные в большие метабатчи
   3.2. Разделять метабатч на батчи необходимого размера
   3.3. Параллелизм

Ссылки

1. https://vk.com/wall-147415323_6918
2. https://arxiv.org/pdf/1412.5567v2.pdf
3. https://www.cs.toronto.edu/~graves/icml_2006.pdf
4. https://docs.google.com/presentation/d/1J9cJfamAQJqXPOTT4TRWlbg7LaxWrY5d5oVoVBfpeKs/edit#slide=id.g967c007adc_0_28
5. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html
6. https://proglib.io/p/fourier-transform/