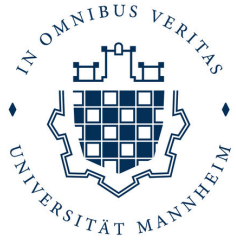
Speech Processing with Deep Learning

Leo Ebert
Tuananh Vu
Moritz Kirchner

Universidad del País Vasco

Euskal Herriko Unibertsitatea

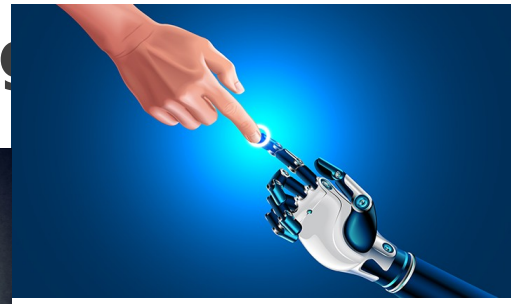


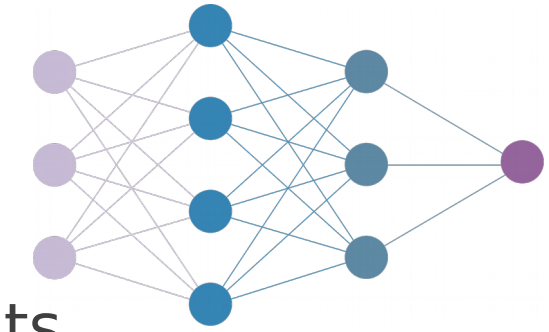
Agenda

1. Introduction
2. Preprocessing
3. Automatic Speech Recognition
4. Postprocessing
5. Demo
6. Conclusion

Speech Processing - First Question WHY

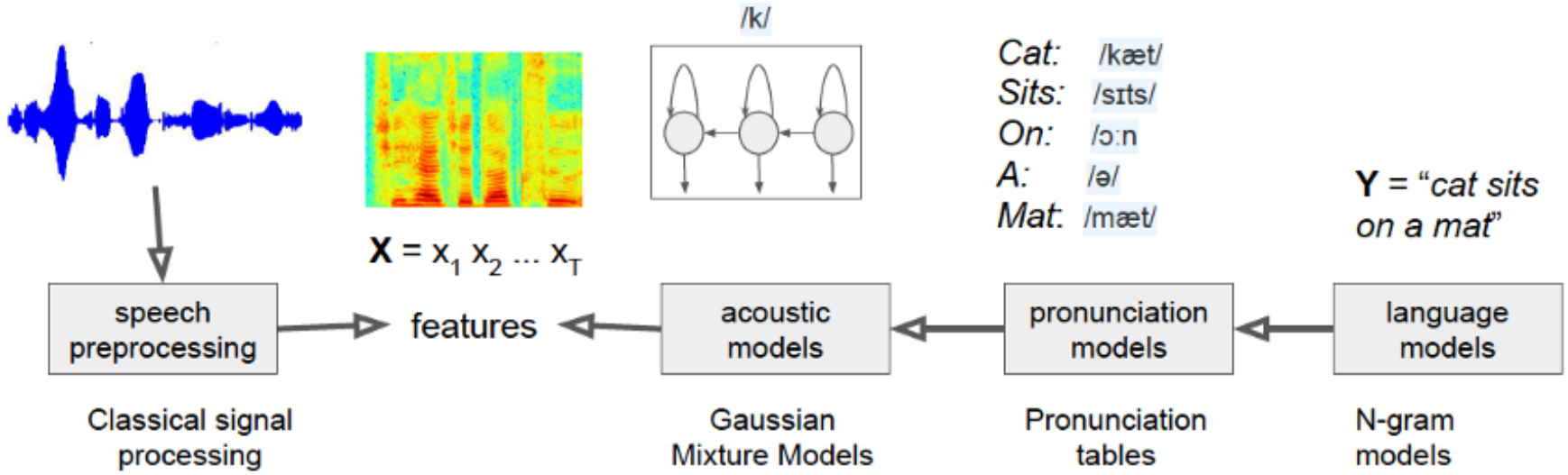
- **WHY Speech Processing**





- **WHY using NN**
  1. Language is complicated -> Dialects
  2. nothing sounds similar -> pronunciation
  3. depends on who is speaking -> Understand also people never heard speaking before
  4. need of using context to “understand”

Speech Processing - an Introduction



heartbeat.fritz.ai

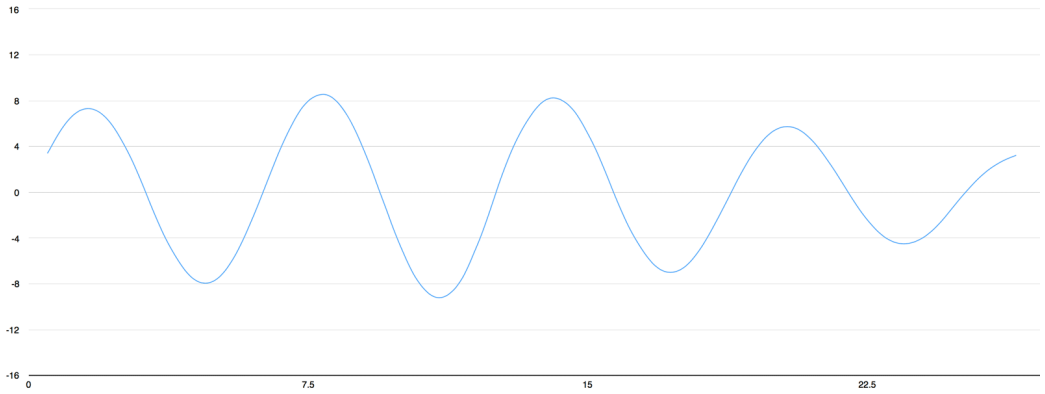
Preprocessing

First step: Feed sound waves into the computer

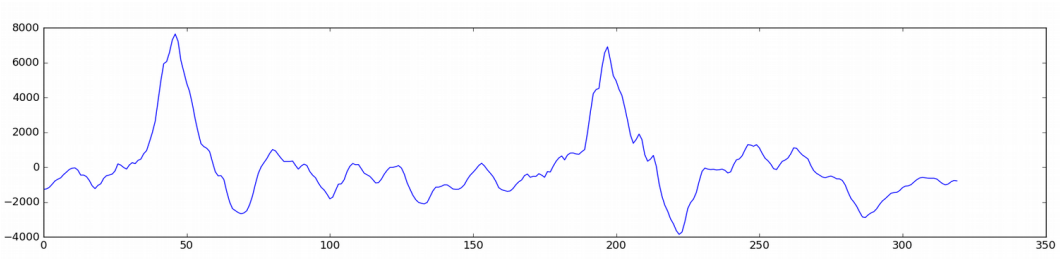
1. Turn Soundwave into numbers

2. Record the height of the wave at equally spaced points

   → Also knowns as sampling

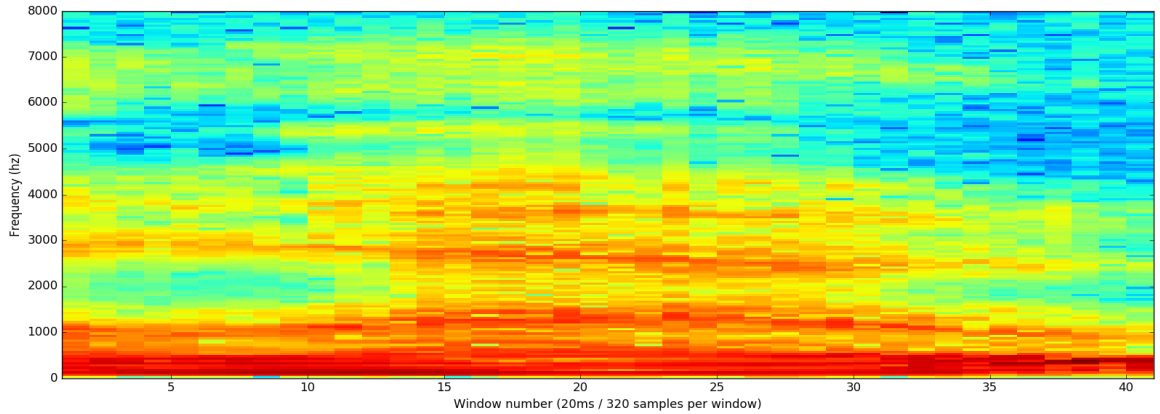


Sampling



23.2346566756,45.2314534534, 12.123254 ...

Preprocessing

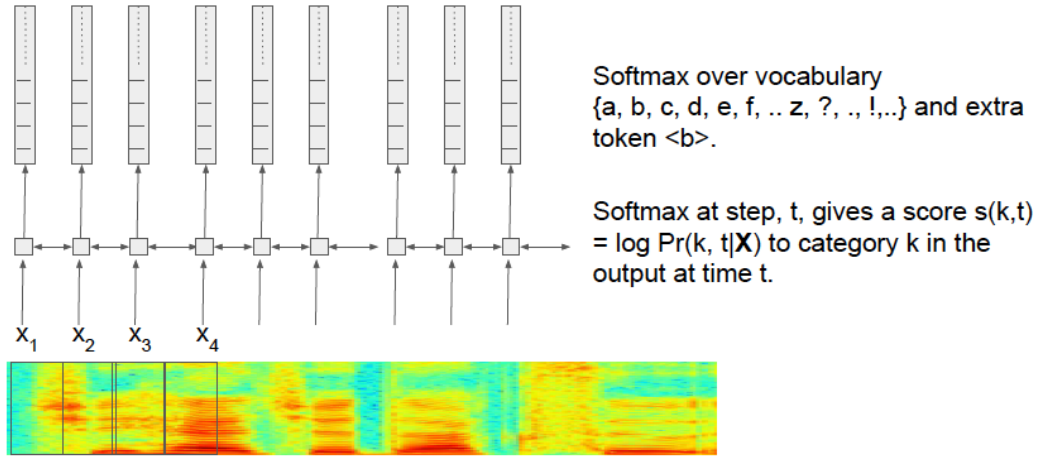


Transformation to a spectrogram

Automatic Speech Recognition

Acoustic Model

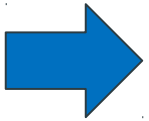
- Deals with the audio waveforms predicting what phoneme each waveform corresponds to
- Many approaches exists (Usually neural network, but N-gram possible)



Automatic Speech Recognition

Language Model

- Probability distribution over a word or word sequence
- Can base on probabilistic methods or neural networks



"I laik tu it" → "i like to eat"

Trigram Probability

Corpus: I am happy because I am learning

$$P(happy|I\ am) = \frac{C(I\ am\ happy)}{C(I\ am)} = \frac{1}{2}$$

Probability of a trigram: $$P(w_3|w_1^2) = \frac{C(w_1^2\ w_3)}{C(w_1^2)}$$

A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

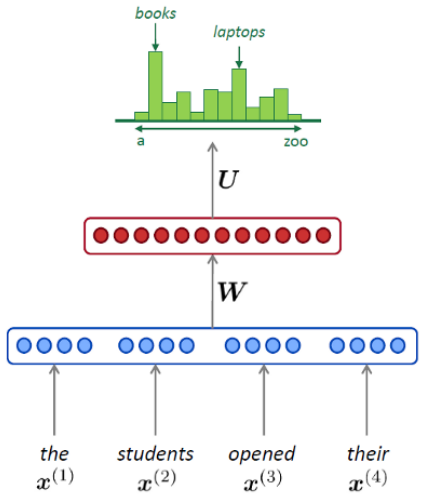
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Post-processing - considerations

- **punctuation restoration and capitalization**

`Let's eat, kids. ´ vs. `Let's eat kids. ´

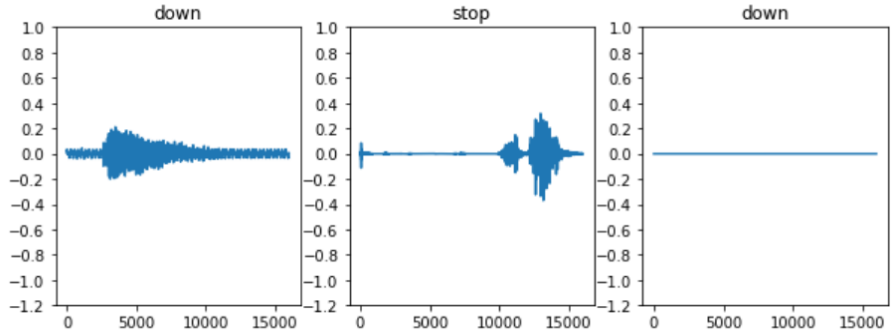
- **Ability to process unknown words**

→ as always **use preferably as little computational resource**

Demo

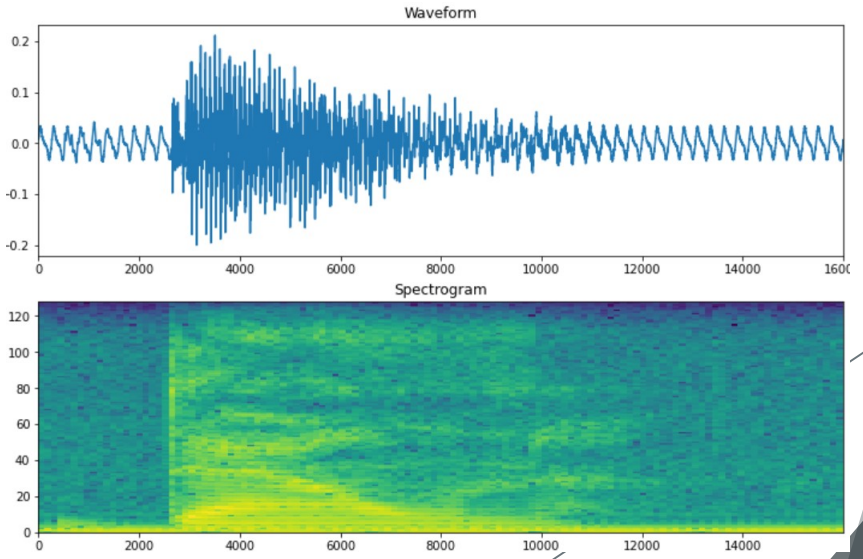
```
import seaborn as sns
import tensorflow as tf
```

```
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(10, 12))
for i, (audio, label) in enumerate(waveform_ds.take(n)):
  r = i // cols
  c = i % cols
  ax = axes[r][c]
  ax.plot(audio.numpy())
  ax.set_yticks(np.arange(-1.2, 1.2, 0.2))
  label = label.numpy().decode('utf-8')
  ax.set_title(label)

plt.show()
```



simple_audio.ipynb - Colaboratory (google.com)

```
def plot_spectrogram(spectrogram, ax):
  # Convert to frequencies to log scale and transpose so that the time is
  # represented in the x-axis (columns). An epsilon is added to avoid log of zero.
  log_spec = np.log(spectrogram.T+np.finfo(float).eps)
  height = log_spec.shape[0]
  width = log_spec.shape[1]
  X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
  Y = range(height)
  ax.pcolormesh(X, Y, log_spec)


fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])
plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.show()
```



Conclusion

Hey Siri

amazon alexa

Appendix

Post-processing - models

	+	-
n-gram based approach	fast results	- needs a lot of disk space - handling of out-of-vocabulary words
bidirectional RNN	high accuracy	size of input ↑ => time per token ↑

- **TruncBiRNN**
  - forwards normal RNN
  - backwards only a specified window is considered

Post-processing - models