Communication parlée multimodale

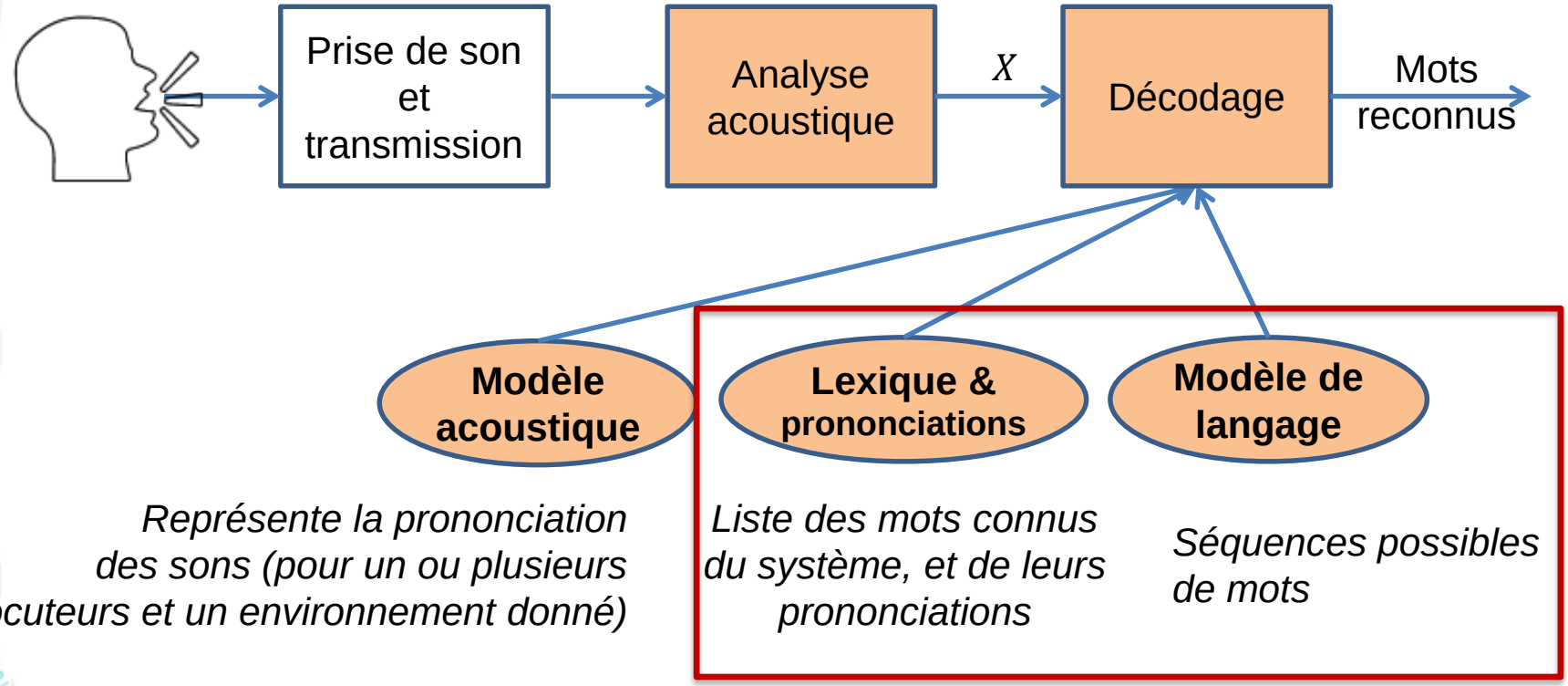
– Reconnaissance de la parole –

TD – Reconnaissance de parole avec pocketsphinx

lexique & modèle de langage

Denis Jouvet
LORIA – INRIA - Nancy

Reconnaissance (automatique) de la parole



Lexique

- Précise la liste des mots et leurs prononciations.

- Exemple (extrait du lexique de pocketsphinx)

```
one             HH W AH N
one(2)          W AH N
person          P ER S AH N
quarter         K AO R T ER
quarter(2)      K W AO R T ER
quarters        K W AO R T ER Z
quit            K W IH T
```

mot

numérotation
des variantes
de prononciation

prononciation

Modèles de langage

- Apporte des informations au décodeur sur les enchaînements possibles de mots

- Grammaires régulières ou hors contexte
  - Impose des contraintes «globales» sur la phrase

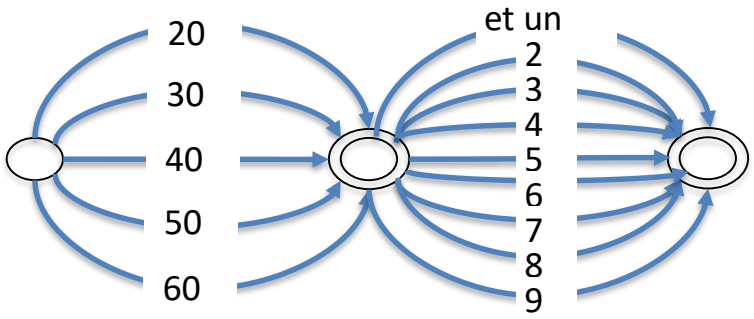
- Modèle statistique n-gram
  - Estimation des paramètres à partir de corpus

- Modèles à base de réseaux de neurones
  - Différentes versions proposées dans la littérature

Grammaires régulières ou hors contexte

- Règles ou graphes qui décrivent exactement les phrases (enchaînements de mots) possibles (ex. chiffre isolé, nombre, …)
  - Ex. Nombre_entre_20_et_69

Nombre_entre_20_et_69 = Dizaine_entre_20_et_60
| Dizaine_entre_20_et_60 . Unité_entre_1_et_9 ;
Dizaine_entre_20_et_60 = 20 | 30 | 40 | 50 | 60 ;
Unité_entre_1_et_9 = « et_un » | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ;



- Impose des contraintes «globales» sur la phrase
- Complexité de la tâche pour grands vocabulaires
- Rend impossible la reconnaissance de phrase syntaxiquement incorrectes
- Difficulté à définir ou à estimer (inférence grammaticale) la grammaire

jsgf ⇔ JSpeech Grammar Format

- *The JSpeech Grammar Format (JSGF) is a platform-independent, vendor-independent textual representation of grammars for use in speech recognition*

- Exemple de grammaire JSGF

```
#JSGF V1.0;
/**
 * JSGF Grammar for Turtle example
 */
grammar goforward;
public <move> = go forward ten meters;
public <move2> = go <direction> <distance> [meter | meters];
<direction> = forward | backward;
<distance> = one | two | three | four | five | six | seven |
eight | nine | ten;
```

Modèle de langage n-gram - principe

- Modèle statistique
- Probabilité d'une suite de mots

$$P(w_1, \dots, w_N) = P(w_1) \prod_{i=2..N} P(w_i | w_1, \dots, w_{i-1})$$

- N-grams prennent en compte des séquences de n mots

$$P(w_i | w_1, \dots, w_{i-1}) \triangleq P\left(w_i \middle| w_{i-(n-1)}, \dots, w_{i-1}\right)$$

- Les paramètres des modèles N-gram sont calculés à partir de grands corpus textuels

Pocketsphinx

- *PocketSphinx is a lightweight speech recognition engine, specifically tuned for handheld and mobile devices, though it works equally well on the desktop*
- https://github.com/cmusphinx/pocketsphinx
- Existe en tant que paquet python – donc facile à installer et à utiliser
- Remarque:
  - La qualité de la reconnaissance dépend de la qualité des modèles acoustiques utilisés (et de leur adéquation avec les données de test)

Objectif du TD sur lexique & modèle de langage

- TD autour des lexiques et modèles de langage
  - Lexique et modèle générique de pocketsphinx (i.e. ngram)
  - Grammaire boucle de chiffres (à écrire)
  - Grammaire suite de chiffres de longueur connue (à écrire)

- Corpus support pour évaluations
  - Suites de chiffres (1 chiffre, 3 chiffres & 5 chiffres)
  - Avec ajout de bruit (rapport signal à bruit de 35 dB, 25 dB, 15 dB et 05 dB)

- Evaluations à faire – taux d'erreur en fonction
  - Des différents modèles de langage
  - De la longueur des suites de chiffres
  - Du groupe de locuteurs
  - Du rapport signal à bruit

Corpus de chiffres

Dans fichier « `td_corpus_digits.zip` »

```
td_corpus_digits
    SNR35dB
        man
            seq1digit_200_files     ➔ fichiers *.wav & *.ref de chiffres isolés
            seq3digits_100_files    ➔ fichiers *.wav & *.ref de suites de 3 chiffres
            seq5digits_100_files    ➔ fichiers *.wav & *.ref de suites de 3 chiffres
        woman
            Même organisation de données que pour « man »
        boy
            Même organisation de données que pour « man »
        girl
            Même organisation de données que pour « man »
    SNR25dB
        Uniquement données « man »; chiffres isolés et suites de 3 & 5 chiffres
    SNR15dB
        Uniquement données « man »; chiffres isolés et suites de 3 & 5 chiffres
    SNR05dB
        Uniquement données « man »; chiffres isolés et suites de 3 & 5 chiffres
```

Lexiques & grammaires

- Dans fichier « `ps_data.zip` »

```
ps_data
    model
        en_us                   ➔ Modèle acoustique anglais
    lex
        cmudict-en-us.dict      ➔ Lexique générique anglais
        turtle.dic              ➔ Lexique réduit pour grammaire jsgf
    lm
        en-us.lm.bin            ➔ Ngram générique anglais
    jsgf
        goforward.gram          ➔ Exemple de grammaire jsgf
    exemple
        goforward.raw           ➔ Fichier signal de parole
```

Exemples pocketsphinx

Dans fichier « `ps_exemples.zip` »

`ps_exemples`

- `decoder_ngram.py` (https://raw.githubusercontent.com/cmusphinx/pocketsphinx/master/swig/python/test/decoder_test.py)
  - Exemple extrait du web – modèle langage ngram générique
  - Modifié pour utiliser lexique et modèles dans ps_data
  - Envoi données au décodeur par petits paquets
- `decoder_jsgf.py` (https://raw.githubusercontent.com/cmusphinx/pocketsphinx/master/swig/python/test/jsgf_test.py)
  - Exemple extrait du web – modèle langage jsgf
  - Modifié pour utiliser lexique et modèles dans ps_data
  - Envoi données au décodeur par petits paquets
- `decoder_utt_ngram.py`
  - Modification de `decoder_ngram.py` pour envoi global du fichier à décoder
- `decoder_utt_jsgf.py`
  - Modification de `decoder_jsgf.py` pour envoi global du fichier à décoder

Pré-requis / installation

Installation des modules nécessaires dans un environnement virtuel python

```
python3 -m venv asr-env
```
➔ Création environnement virtuel python

```
source asr-env/bin/activate
```
➔ Activation environnement virtuel

```
pip install pocketsphinx
```
➔ Installation de pocketsphinx

```
    python ps_exemples/decoder_ngram.py
```
➔ Vérification fonctionnement

```
    python ps_exemples/decoder_utt_ngram.py
```
➔ Vérification fonctionnement

```
    python ps_exemples/decoder_jsgf.py
```
➔ Vérification fonctionnement

```
    python ps_exemples/decoder_utt_jsgf.py
```
➔ Vérification fonctionnement

```
pip install asr-evaluation
```
(usage: `wer -i toto.ref toto.hyp`) ➔ Calcul taux d'erreur

….. ➔ Suite du TD…

```
Deactivate
```
➔ Sortie environnement virtuel

Exemple code pocketsphinx
decoder_ngram.py

```
# Create a decoder with certain model
config = Decoder.default_config()
config.set_string('-hmm',  'ps_data/model/en-us')
config.set_string('-lm',   'ps_data/lm/en-us.lm.bin')
config.set_string('-dict', 'ps_data/lex/cmudict-en-us.dict')

# Decode streaming data.
decoder = Decoder(config)

decoder.start_utt()
stream = open('ps_data/exemple/goforward.raw', 'rb')
while True:
     buf = stream.read(1024)
     if buf:
          decoder.process_raw(buf, False, False)
     else:
          break
decoder.end_utt()

hypothesis = decoder.hyp()
print ('Best hypothesis: ', hypothesis.hypstr)
```

Calcul du taux d'erreur

- Nécessite deux fichiers
  - `Data.ref` ⇔ contient les références (transcriptions des prononciations)
  - `Data.hyp` ⇔ contient les sorties de la reconnaissance.
- Attention:
  - la n-ème ligne de `Data.ref` doit correspondre à la n-ème ligne de `Data.hyp` i.e., référence du n-ème enregistrement, et décodage (reco) associé

- Exemple
  - `Data.ref`

```
eight four five
five four seven
zero seven six
oh one three
four five one
…
```

  - `Data.hyp`

```
eight eight four five
five four seven
zero seven six
five oh one three
four five one
…
```

Calcul du taux d'erreur (suite)

Utiliser le module `wer.py`

```
wer -i Data.ref Data.hyp
```

Exemple de résultat

```
REF: ***** eight four five
HYP: EIGHT eight four five
SENTENCE 1
Correct          = 100.0%    3   (     3)
Errors           =  33.3%    1   (     3)
REF: five four seven
HYP: five four seven
SENTENCE 2
Correct          = 100.0%    3   (     3)
Errors           =   0.0%    0   (     3)
REF: zero seven six
HYP: zero seven six
SENTENCE 3
Correct          = 100.0%    3   (     3)
Errors           =   0.0%    0   (     3)
REF: **** oh one three
HYP: FIVE oh one three
SENTENCE 4
Correct          = 100.0%    3   (     3)
Errors           =  33.3%    1   (     3)
```

Calcul du taux d'erreur & intervalle de confiance

La fin du fichier (produit par `wer`) ressemble à

```
Sentence count: 40
WER:    19.167% (      23 /      120)
WRR:    99.167% (     119 /      120)
SER:   100.000% (      40 /       40)
```

Taux d'erreur mot — Nombre d'erreurs — Nombre de mots

Intervalle de confiance ⇔ incertitude sur la mesure du taux d'erreur

Intervalle de confiance à 95%:

$$1.96\sqrt{\frac{P\,.\,(1-P)}{N}}$$

Où P est le taux d'erreur, et N le nombre de mots des données de test

Ici:

$1.96\sqrt{\frac{0.19\,.(1-0.19)}{120}} = 0.070$ soit un taux d'erreur de 19.2%±7.0%

Préparation lexique et modèle de langage

- Fichier lexique correspondant aux chiffres
  - Lexique : *zero, one, …, nine, oh*
  - Extraire les prononciations du lexique de pocketsphinx
- Grammaire jsgf « boucle de chiffres »
  - Grammaire permetant une suite de chiffres de longueur quelconque
- Grammaire jsgf de suites de chiffres de longueur connue
  - 3 entrées correspondant à 1 chiffre, 3 chiffres et 5 chiffres
  - Peuvent être spécifiées dans un même fichier de grammaire jsgf

Mise au point des programmes pour la reconnaissance de la parole

1. Adapter les scripts fournis pour traiter les grammaires jsgf de chiffres

2. 3 scripts à mettre au point et à tester sur un ou quelques fichiers (suites de chiffres)
   - Modèle de langage ngram générique
   - Grammaire boucle de chiffres
   - Grammaire suite de chiffres de longueur connue (correspondant à la longueur de l'enregistrement à reconnaître)

3. Modifier les scripts pour traiter un ensemble de fichiers et écrire les résultats dans un fichier, par exemple

```
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_040.raw :: five
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_022.raw :: oh five eight eight
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_023.raw :: zero eight oh three oh
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_024.raw :: one nine three five
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_025.raw :: nine eight six
```

Mise au point des programmes pour le calcul des taux d'erreur

1. Faire un script qui lit un fichier de résultats, et crée
   1. Un fichier type « `Data.ref` » en récupérant pour chaque enregistrement la prononciation de référence
   2. Un fichier type « `Data.hyp` » contenant les résultats de reconnaissance

2. Appliquer le script de comptage des erreurs

```
wer –i  Data.ref  Data.hyp  >  Data.results
```

3. Examiner le fichier « `Data.results` »

4. En extraire le taux d'erreur sur l'ensemble des données, et **calculer l'intervalle de confiance associé**

1 – Impact du modèle de langage

- Données
  - SNR35dB ⇔ très peu de bruit
  - man ⇔ locuteurs hommes
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 fichiers

- Modèles de langage
  - Modèle ngram générique
  - Grammaire JSGF correspondant à une boucle de chiffres
  - Grammaire JSGF correspondant à une suite de chiffres de longueur connue (1 digit ou 3 digits ou 5 digits)

- Evaluations à faire
  - **Pour chaque modèle de langage**, calculer le taux d'erreur mot, et l'intervalle de confiance associé, sur les 400 fichiers correspondant à des données homme très peu bruitées (35 dB SNR)

2 – Variabilité selon les groupes de locuteurs

- Données
  - SNR35dB ⇔ très peu de bruit
  - man, woman, boy, girl ⇔ i.e., tous les groupes de locuteurs
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 fichiers par groupe de locuteurs

- Modèles de langage
  - Grammaire JSGF correspondant à une suite de chiffres de longueur connue (1 digit ou 3 digits ou 5 digits)

- Evaluations à faire
  - **Pour chaque groupe de locuteur** *(man, woman, boy, girl)*, calculer le taux d'erreur mot, et l'intervalle de confiance associé, sur les 400 fichiers correspondant à chaque groupe de locuteurs, avec données très peu bruitées (35 dB SNR)

3 – Performances en fonction des longueurs des séquences de chiffres

- Données
  - SNR35dB ⇔ très peu de bruit
  - man, woman ⇔ i.e., seulement les locuteurs adultes
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 fichiers par groupe de locuteurs

- Modèles de langage
  - Grammaire JSGF correspondant à une suite de chiffres de longueur connue (1 digit ou 3 digits ou 5 digits)

- Evaluations à faire
  - **Pour chaque longueur de suites de chiffres** *(1 digit, 3 digits, 5 digits*), calculer le taux d'erreur mot, et l'intervalle de confiance associé, sur les fichiers correspondant à chaque catégorie (400 fichiers pour 1 digit, 200 fichiers pour 3 & 5 digits), avec données très peu bruitées (35 dB SNR) des locuteurs adultes

4 – Impact du niveau de bruit

- Données
  - SNR35dB, SNR25dB, SNR15dB & SNR05dB
  - man ⇔ i.e., seulement les locuteurs hommes
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 fichiers par groupe de locuteurs

- Modèles de langage
  - Grammaire JSGF correspondant à une suite de chiffres de longueur connue (1 digit ou 3 digits ou 5 digits)

- Evaluations à faire
  - **Pour chaque niveau de bruit** (SNR35dB, SNR25dB, SNR15dB & SNR05dB), calculer le taux d'erreur mot, et l'intervalle de confiance associé, sur les 400 fichiers correspondant à chaque catégorie, avec données des locuteurs hommes

Résultats à retourner

(De préférence par dépôt sur arche, ou par mail, dans des fichiers zip)

- Par groupe
  - Les programmes écrits
  - Les logs correspondants aux tests de reconnaissance, et aux résultats du calcul des taux d'erreur (sortie du script `wer`)

- Individuellement
  - Une synthèse commentée du TD, et en particulier des résultats (évaluations) obtenus
    (2 à 3 pages maximum)