

# Kaldiツールキットを用いた 音声認識システムの構築

篠崎隆宏

東京工業大学 工学院 情報通信系

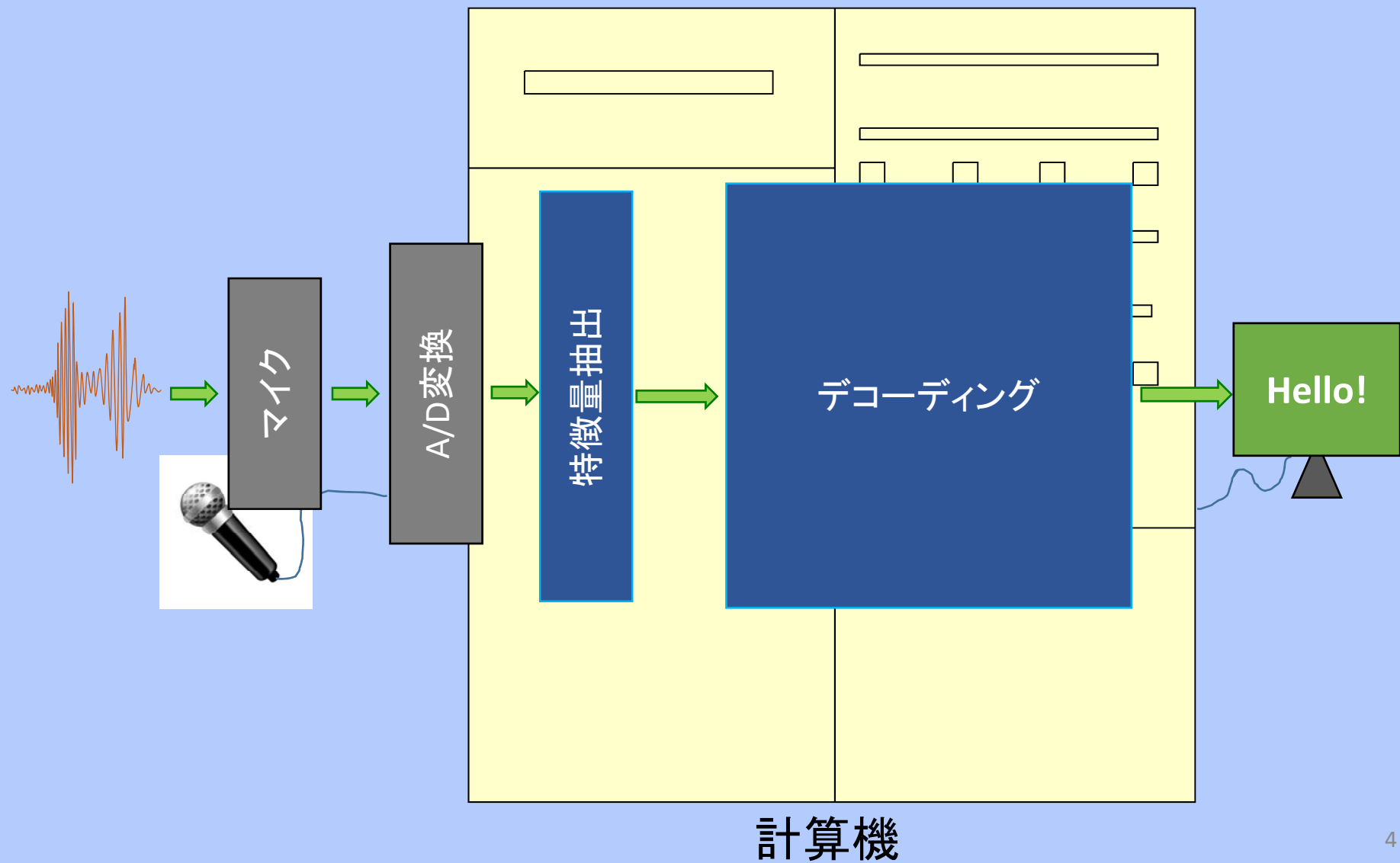
[www.ts.ip.titech.ac.jp](http://www.ts.ip.titech.ac.jp)

# 概要

- 音声認識システムの仕組み
- Kaldiツールキットの概要
- 日本語話し言葉音声認識のためのKaldi用CSJレシピ
- デモ用CSJレシピを用いたチュートリアル

# 音声認識システムの仕組み

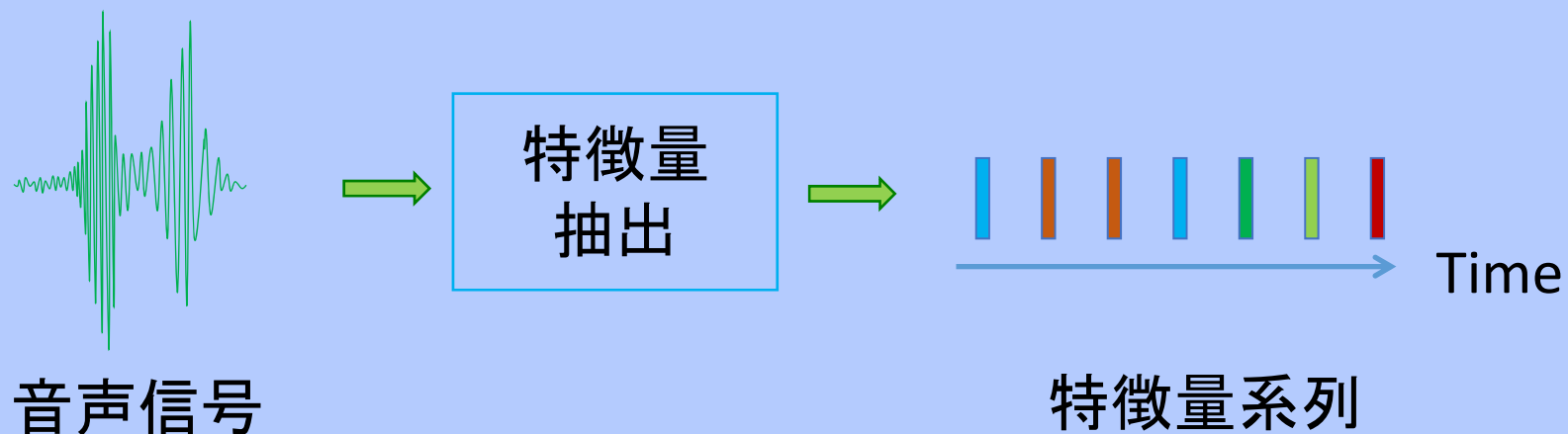
# 音声認識システムの全体構成



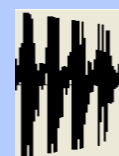
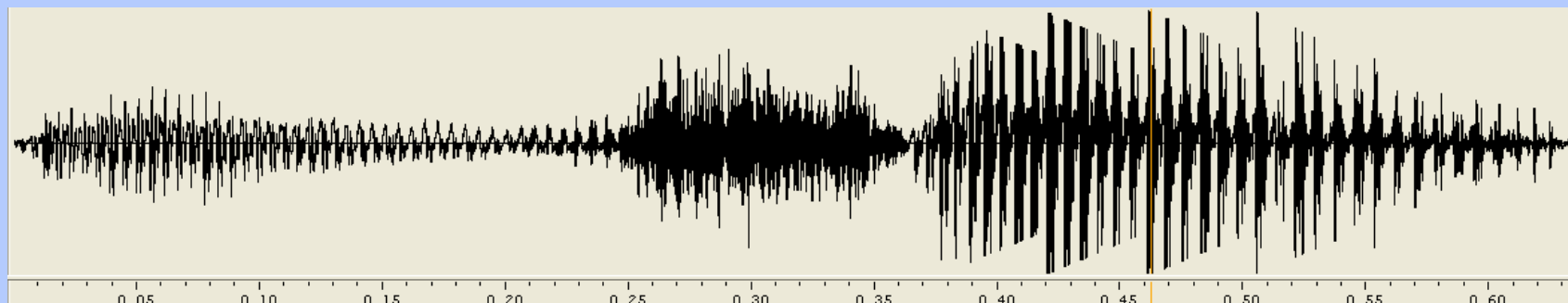
# 特徴量抽出

認識に有用な特徴を認識処理に都合の良い形で抽出

- 認識性能の向上
- 認識のための計算量やメモリ量の削減

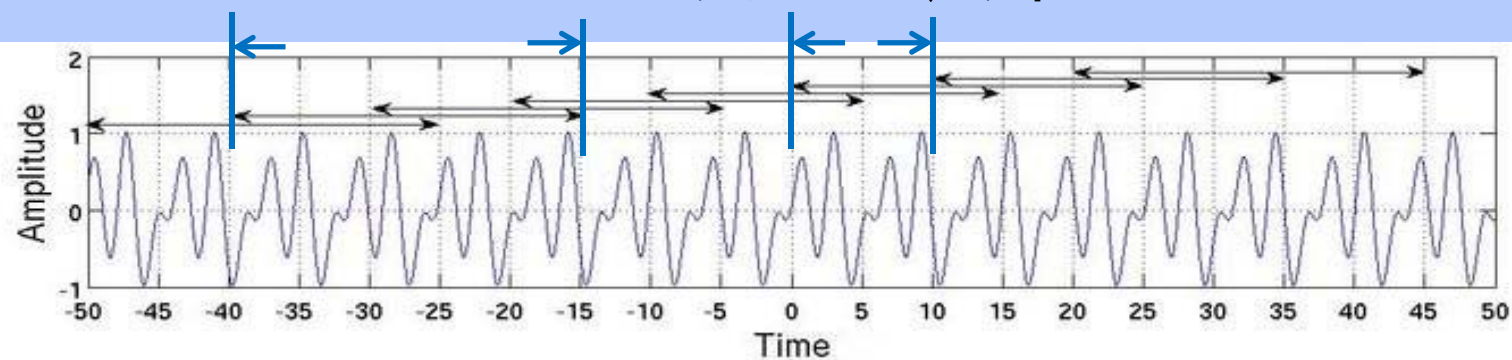


# 音声ベクトルの切り出し



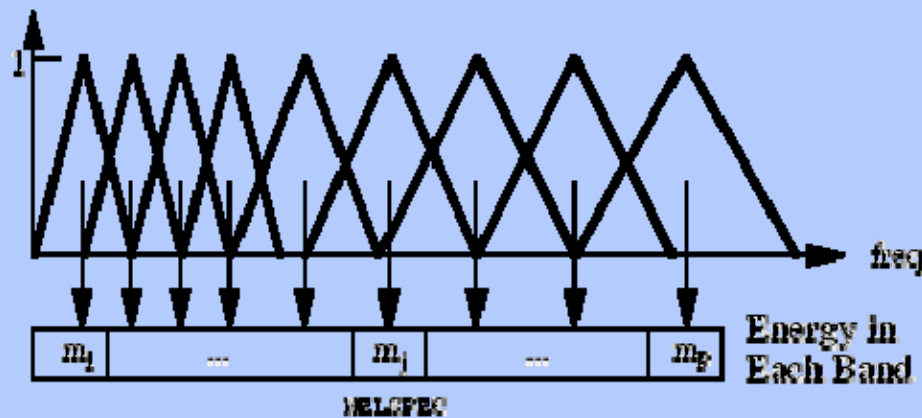
窓(フレーム) 幅

フレームシフト

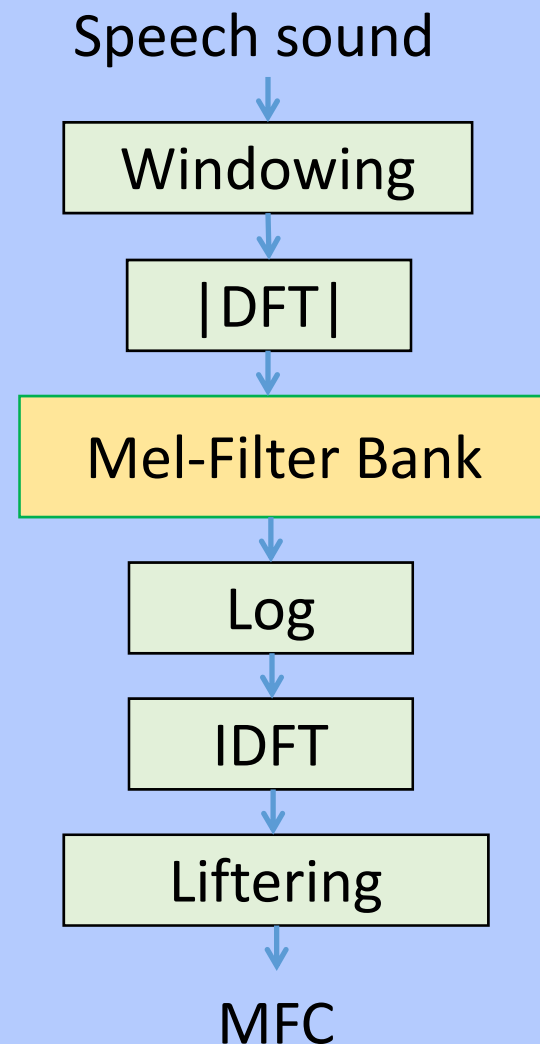


# メル周波数ケプストラム(MFC)

- 周波数の包絡情報を抽出するケプストラム特徴量の一種
- メル尺度のフィルタバンクを用いることで、人の聴覚特性をエミュレート
- 特徴量はメルケプストラム係数特徴量(MFCC)とも呼ばれる

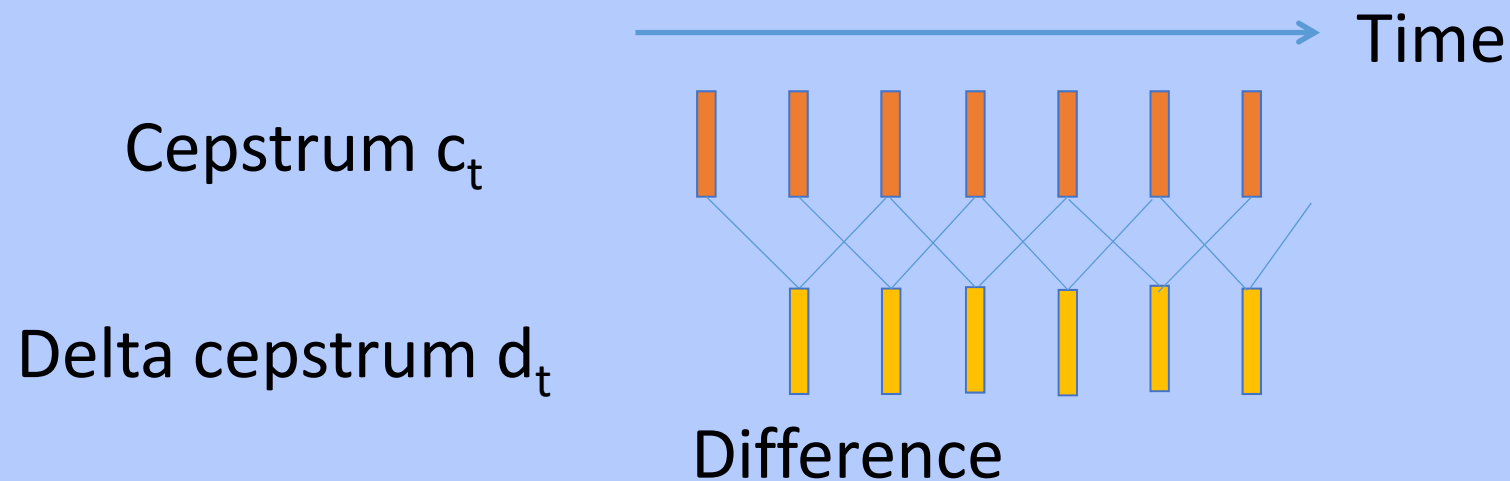


Mel-Scale Filter Bank



# デルタ特徴量

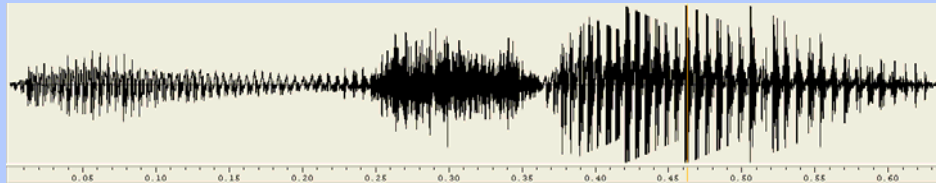
- 特徴量の変化を特徴量として利用



$$d_t = \frac{c_{t+1} - c_{t-1}}{2}$$



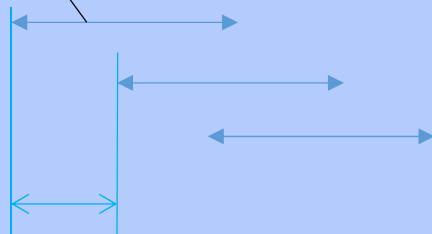
# 特徴量抽出の典型例



16kHz sampling  
16bit quantization



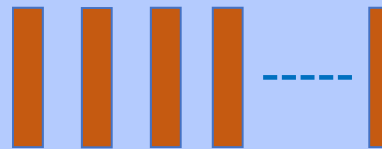
Frame length: 32ms (=512samples/16kHz)



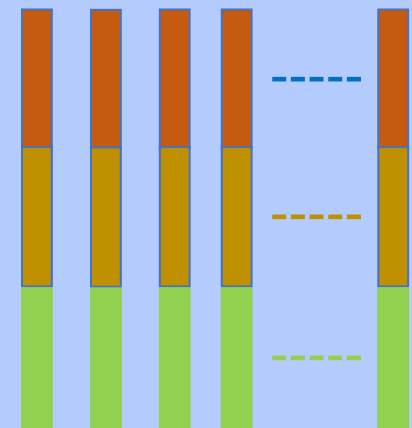
Frame shift: 10ms

Feature sequence  
(MFC etc)

Dim=12, Rate=100Hz



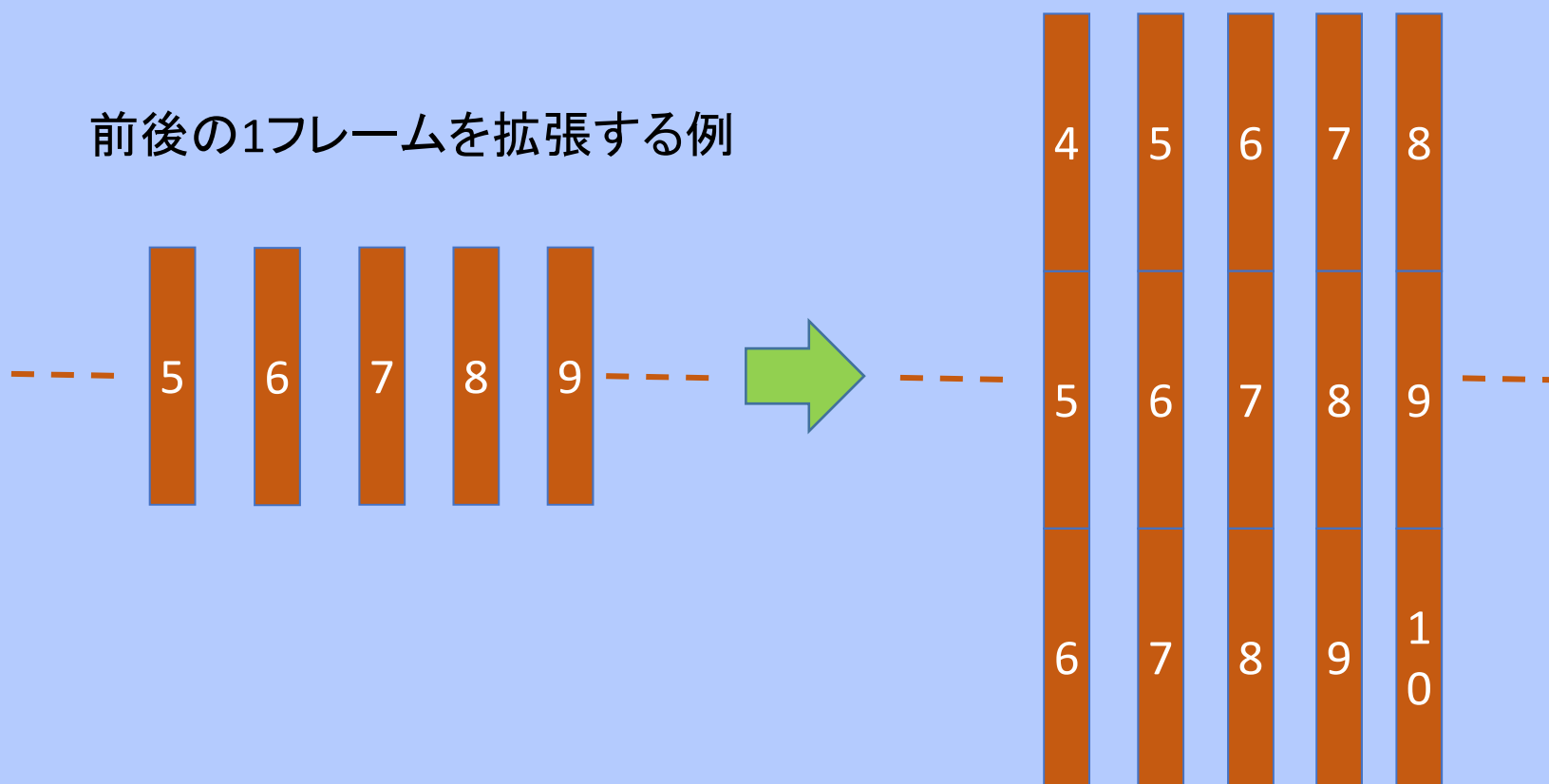
Features with  
delta and delta delta  
Dim=36, Rate=100



# 特徴量のスプライシング (Splicing)

- 各フレームの特徴量ベクトルに、前後のフレームの特徴量ベクトルを連結して拡張

前後の1フレームを拡張する例



# 統計的音声認識

音声信号(特徴ベクトル系列)  $\mathbf{o}$  が観測されたとき、  
単語系列が  $\mathbf{w}$  である事後確率  $P(\mathbf{w}|\mathbf{o})$

この確率を最大化する単語列  $\rightarrow$  認識結果

(最大事後確率則)

$$\hat{\mathbf{w}} = \arg \max_{\mathbf{w}} P(\mathbf{w} | \mathbf{o})$$

# 音響モデルと言語モデル

- 事後確率最大化の式をベイズの定理を用いて変形

$$\begin{aligned}\hat{W} &= \arg \max_W \{P(W | X)\} = \arg \max_W \left\{ \frac{P(X | W)P(W)}{P(X)} \right\} \\ &= \arg \max_W \{P(X | W)P(W)\}\end{aligned}$$

$P(X | W)$     音響モデル: 発音カテゴリが与えられた  
条件での特徴量の出現確率

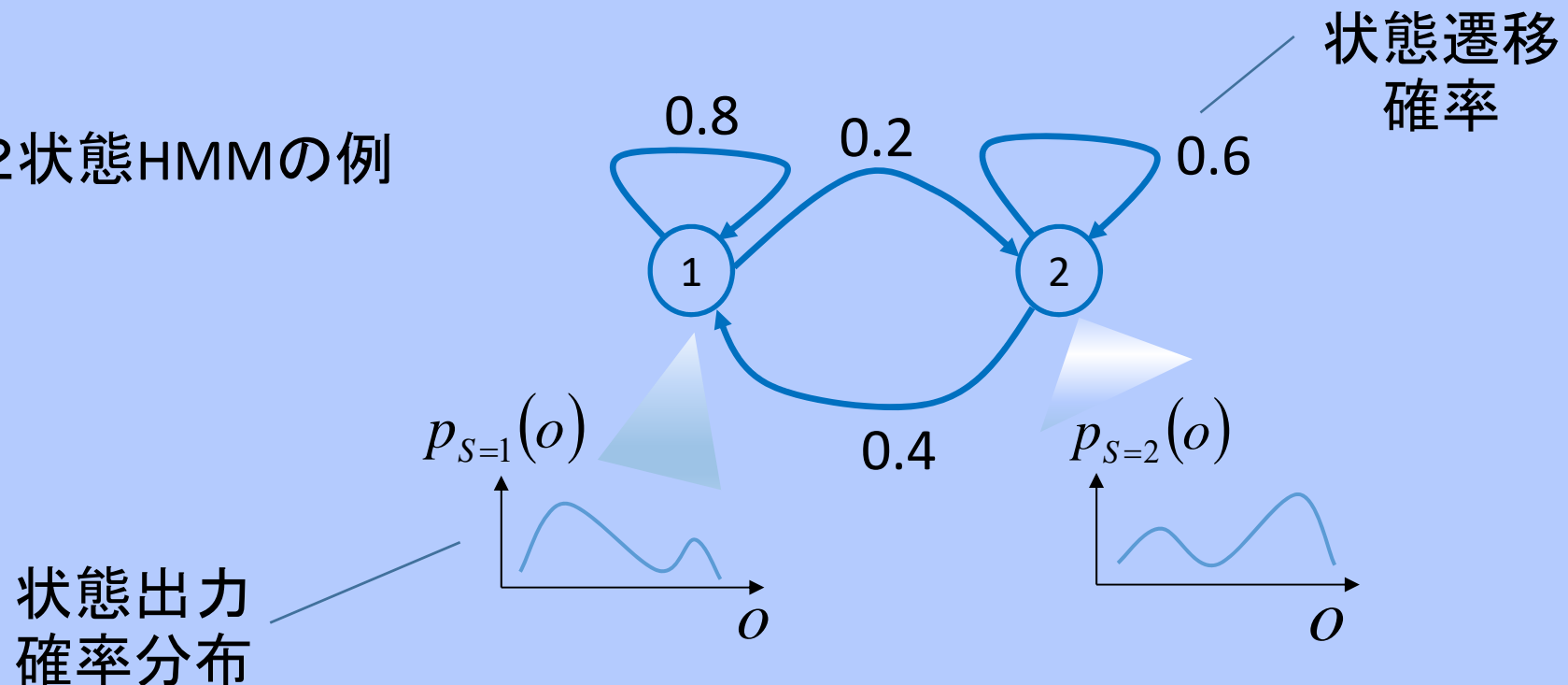
$P(W)$     言語モデル: 認識対象カテゴリの事前確率

$\arg \max_W$     認識デコーダ: 最適解の探索

# 隠れマルコフモデル (HMM)

- 時系列データに対する確率モデル
- 状態集合、状態遷移確率、状態出力確率により定義される

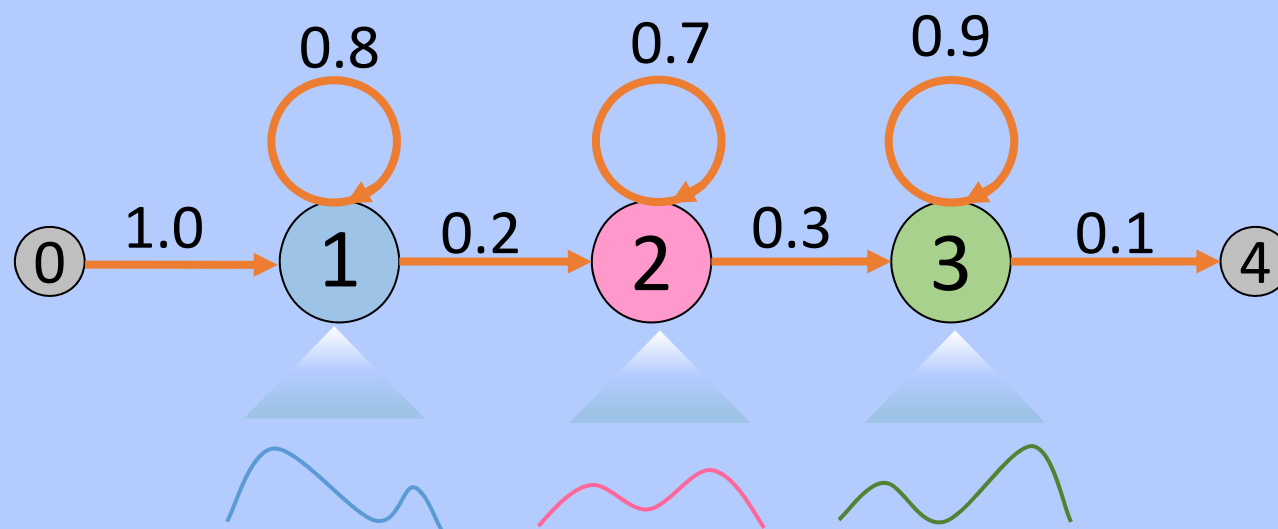
2状態HMMの例



# HMM 音響モデル

- Left-to-right型HMMの利用が一般的
  - 自己遷移か、「次」の状態への遷移のどちらかのみ
  - 出力確率分布を持たない初期状態と最終状態を用意

## 3状態 left-to-right HMMの例



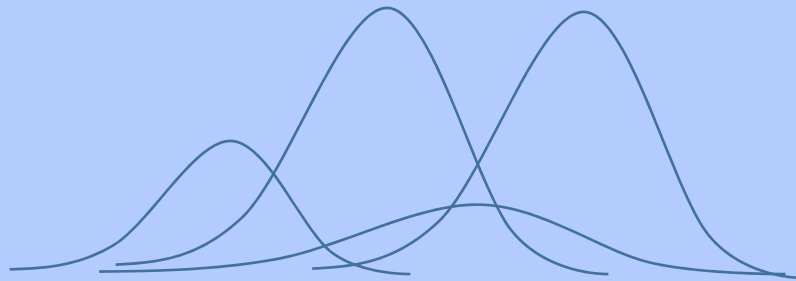
# 混合ガウス分布モデル(GMM)

- 複数のガウス分布を重み付きで重ねることで、複雑な分布の表現を可能としたもの

$$GMM(X) = \sum_i w_i N_i(X | \mu_i, S_i) \quad \sum_{m=1}^M w_k = 1.0$$

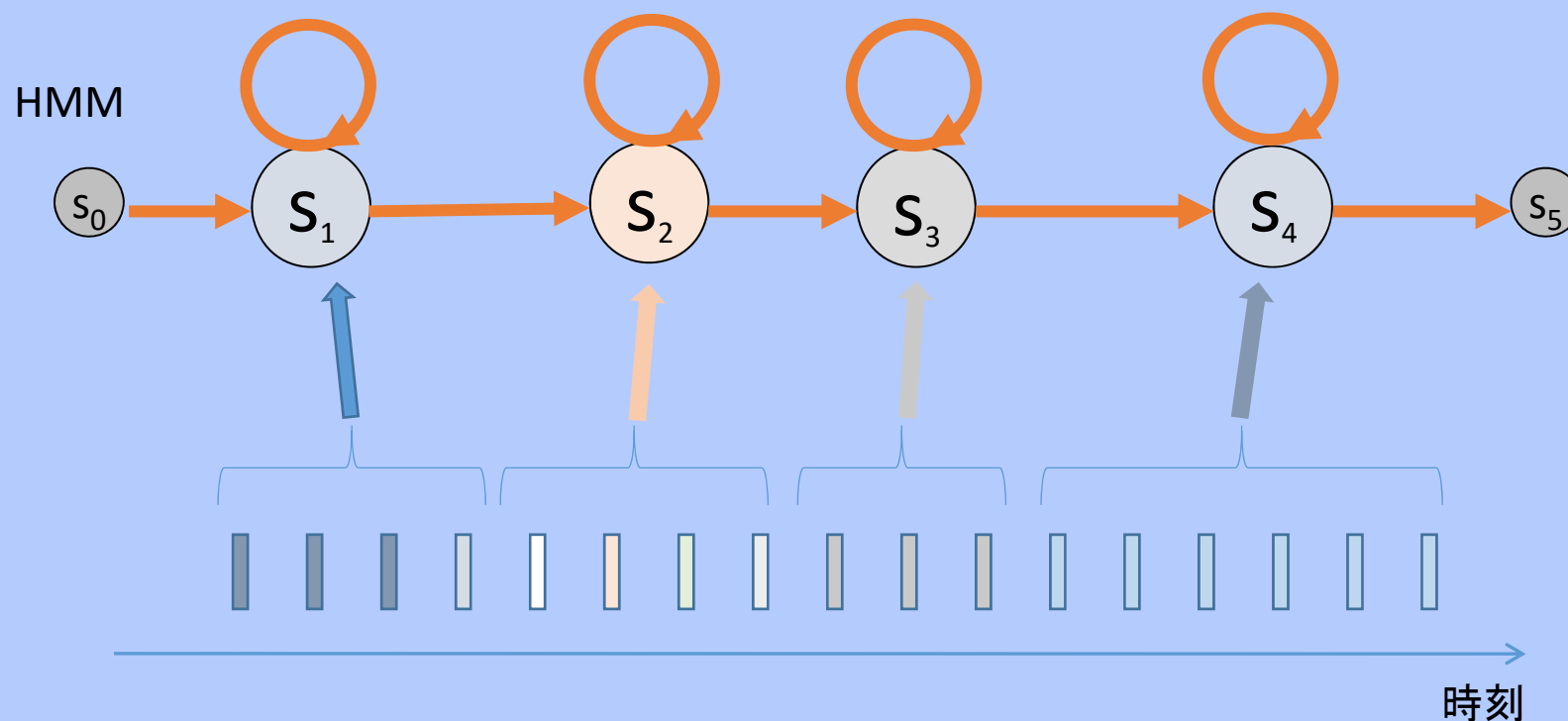
$w_i$  : 混合重み

$N_i$  : 要素ガウス分布(平均  $\mu_i$  分散共分散行列  $S_i$ )



# アライメントと状態系列

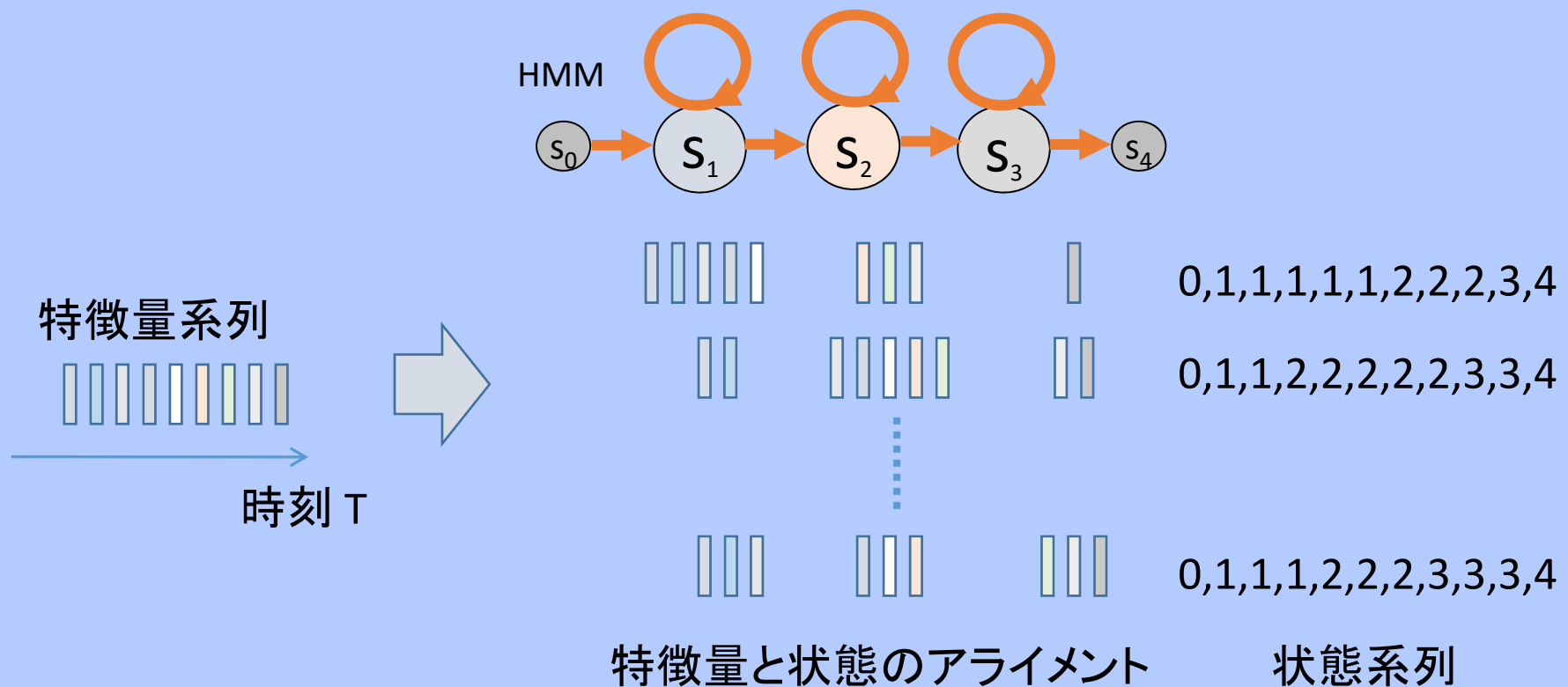
- アライメント: HMM状態と特徴量系列の対応関係
- 状態系列: アライメントに対応した初期状態から最終状態までの状態遷移の道のり





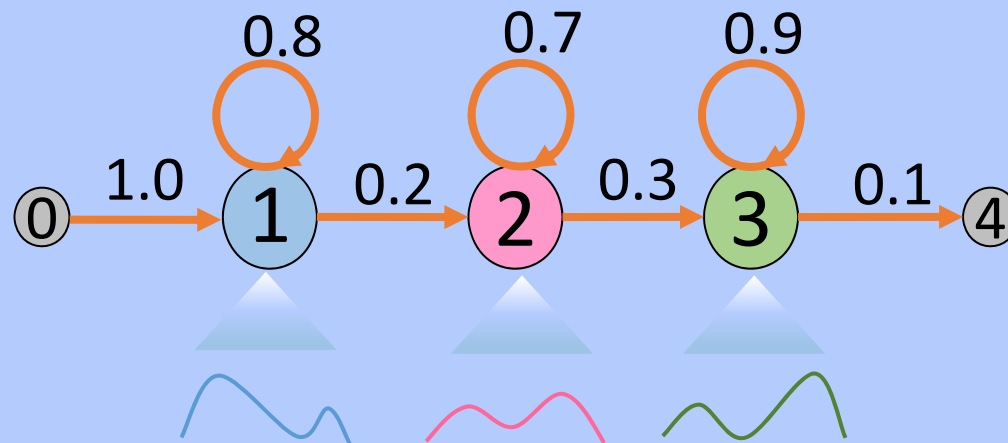
# 「隠れ」マルコフモデル

- 特徴量系列が与えられても、対応する状態系列は一意には定まらない
  - 状態が外から見ると隠れているので「隠れ」マルコフモデル



# HMMによる確率の計算

- 初期状態から最終状態へたどるパスに沿って遷移確率と出力確率を積算
- それをすべてのパスについて計算し、和をとる



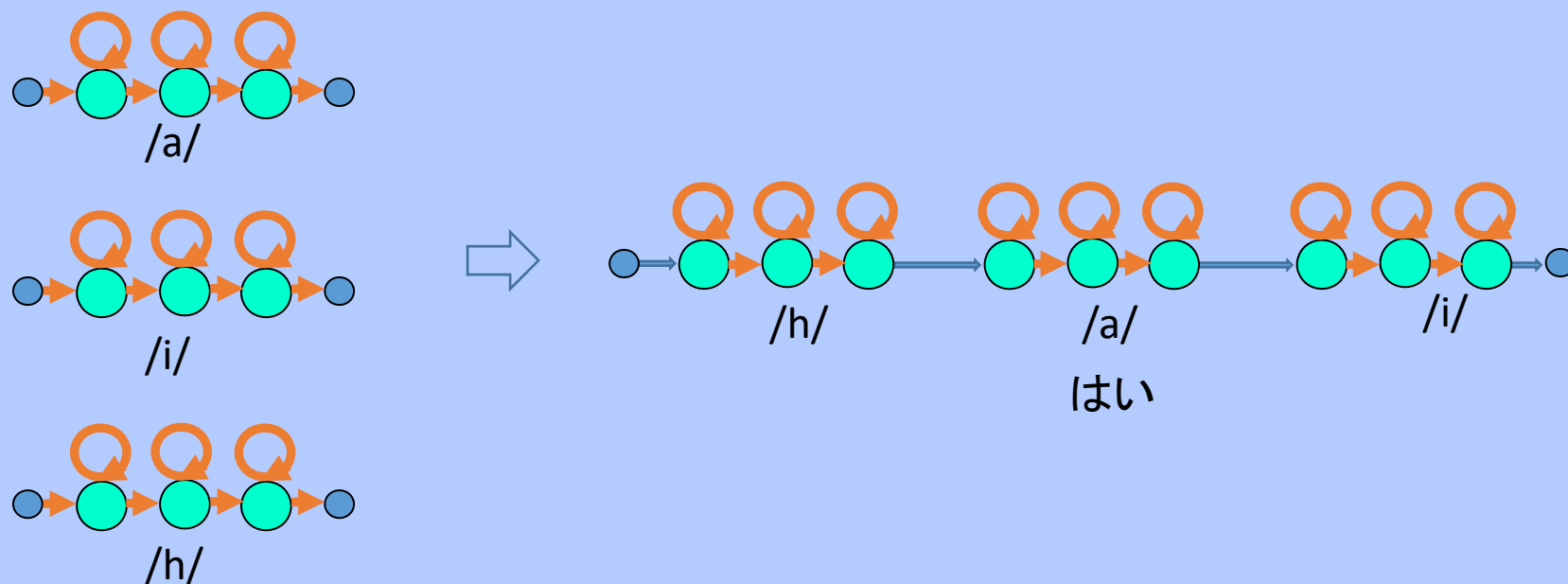
長さ $T$ の特徴量系列  $O=<o_1, o_2, \dots, o_T>$  の確率:

$$P(O) = \sum_{S \in SS} \left\{ \prod_{t=1}^T P(s_t | s_{t-1}) P(o_t | s_t) \right\} P(s_{Fin} | s_T), \quad s_0 = 0, \quad s_{Fin} = N + 1, \quad s_t \in \{1, 2, \dots, N\}$$

$S=<0, s_1, s_2, \dots, s_T, N+1>$ ,  $SS$  は全ての可能な状態系列

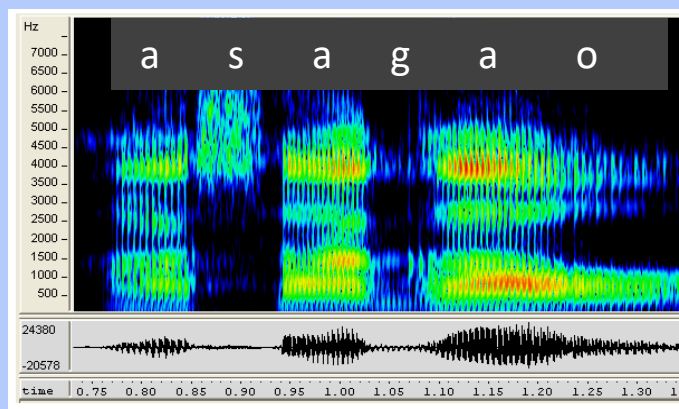
# 音素モデルと発話HMMの合成

- 音素モデル(大語彙音声認識で一般的)
  - 各音素を1つのHMMで表現
  - 単語や文章は、音素HMMを繋げて表現

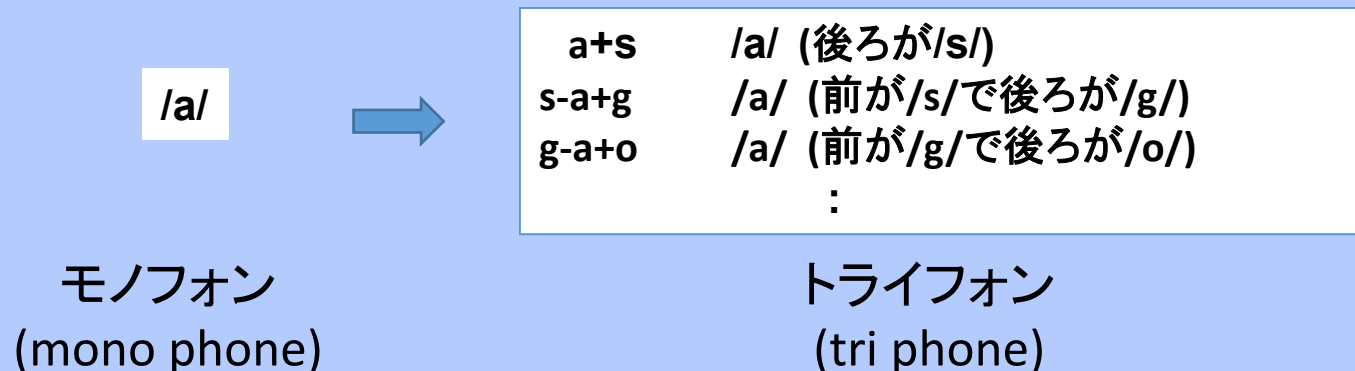


# 音素モデルのコンテキスト依存化

- ・ 調音結合: 同じ音素でも前後の音素に影響されてスペクトルが異なる

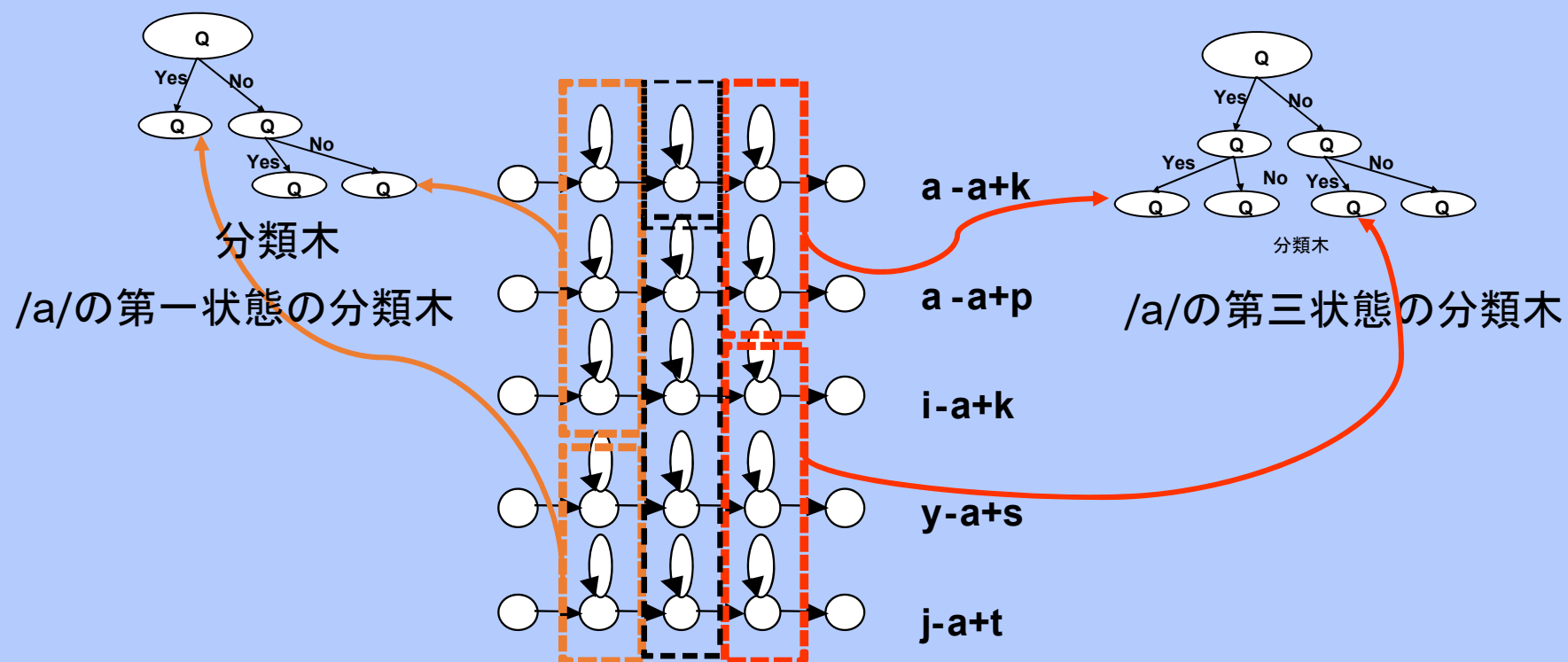


- ・ トライホンモデル: 音素モデルを前後の音素に応じて場合分けすることで、高精度化する



# 状態共有トライホン

- クラスタリングに基づき状態を共有化

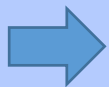


# Feature (Constrained) MLLR

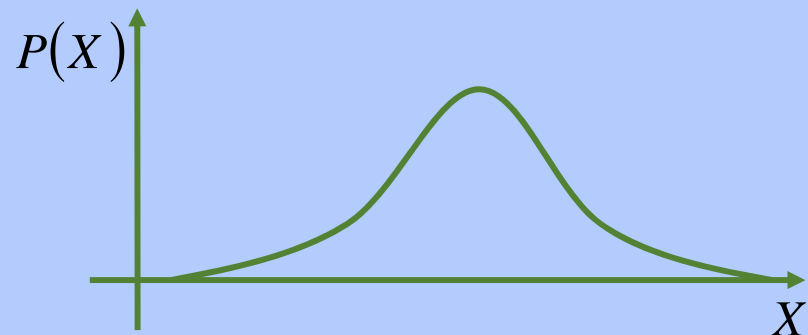
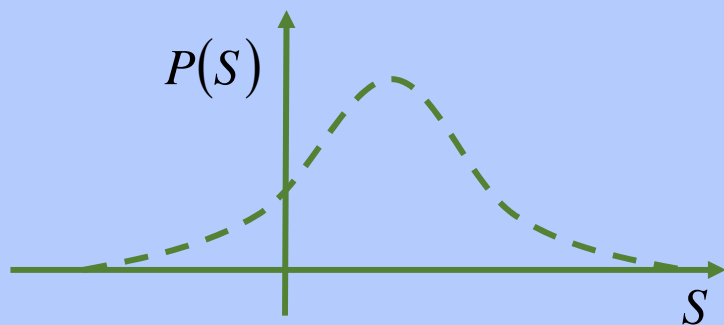
仮定:

1. データ $X$ (話者 $x$ さんの音声)の分布  $P(X)$  は分かっているが、実際に扱うデータ(話者 $s$ さんの音声)  $s$  は違う分布に従う
2.  $s$ は $X$ に対してアフィン変換を適用したものである
3. アフィン変換のパラメタ  $A$  と  $b$  は未知

$$S = f(X) = AX + b$$

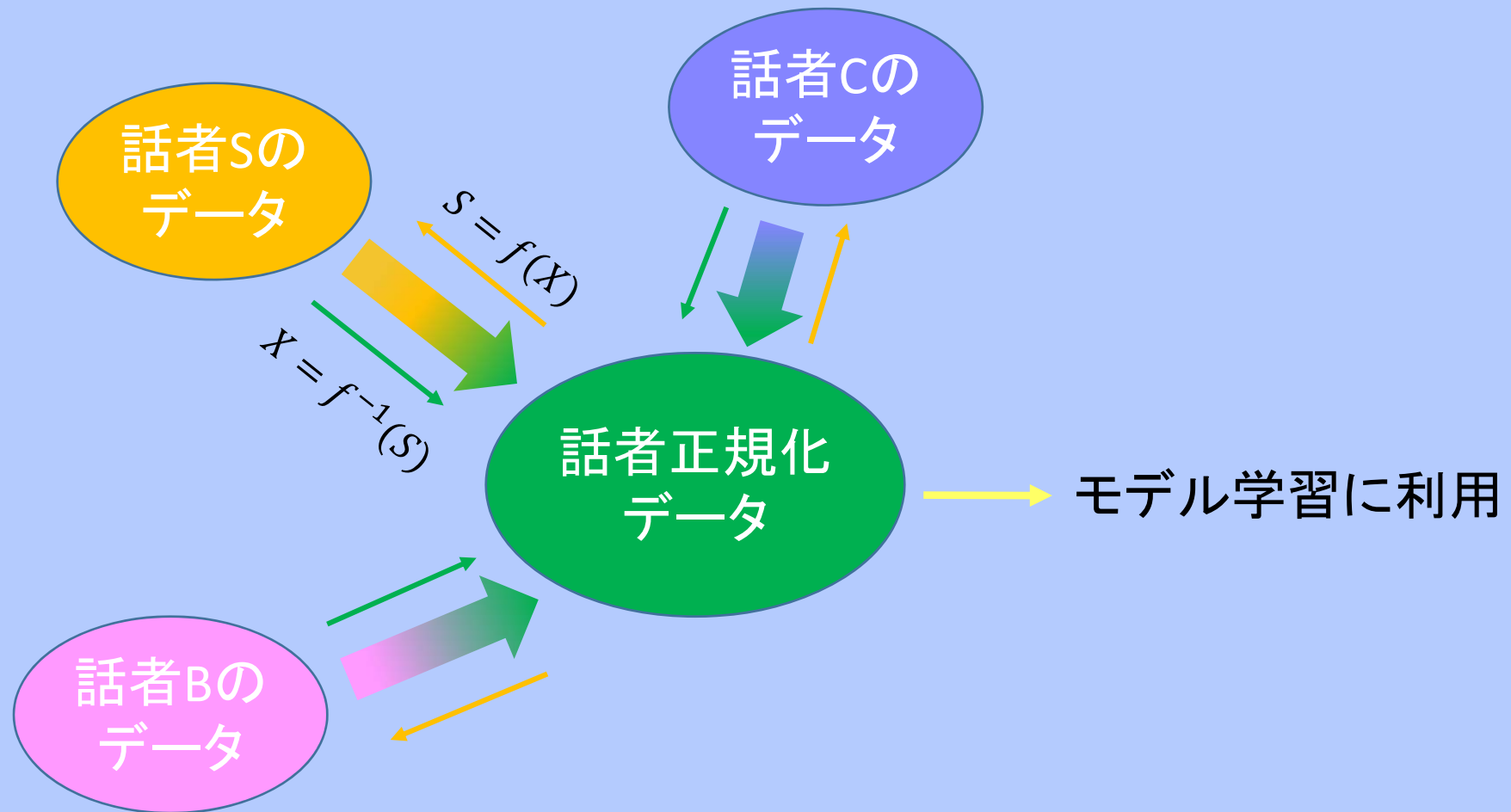


最尤法により $A$  と  $b$  を推定、  
データ $S$ に逆変換をかけることで分布  $P(X)$ とのミスマッチを解消



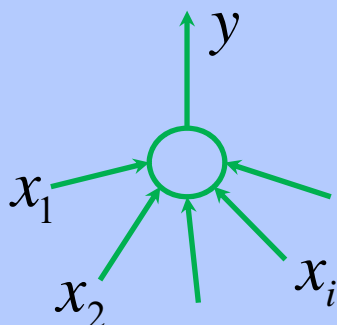
# 話者適応学習(SAT)

- 話者適応技術を応用することで、学習時における話者の違いを正規化



# 多層パーセプトロン(MLP)

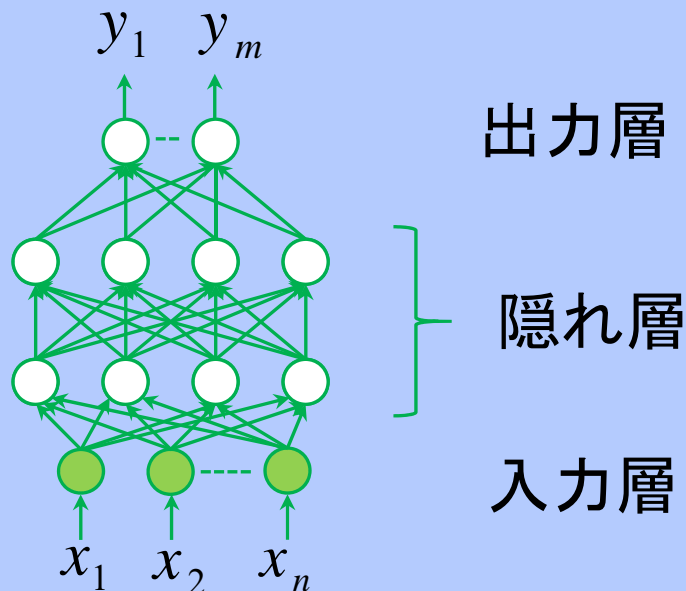
- 構成ユニット(神経細胞に相当)



$$y = h\left(\sum_i w_i x_i + b\right)$$

$h$ : activation function  
 $w$ : weight  
 $b$ : bias

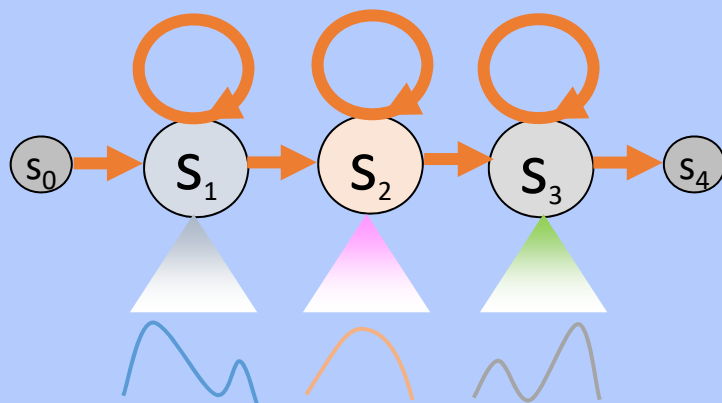
- 多層パーセプトロンはユニットが集まったレイヤーが多階層に積み重なったもの





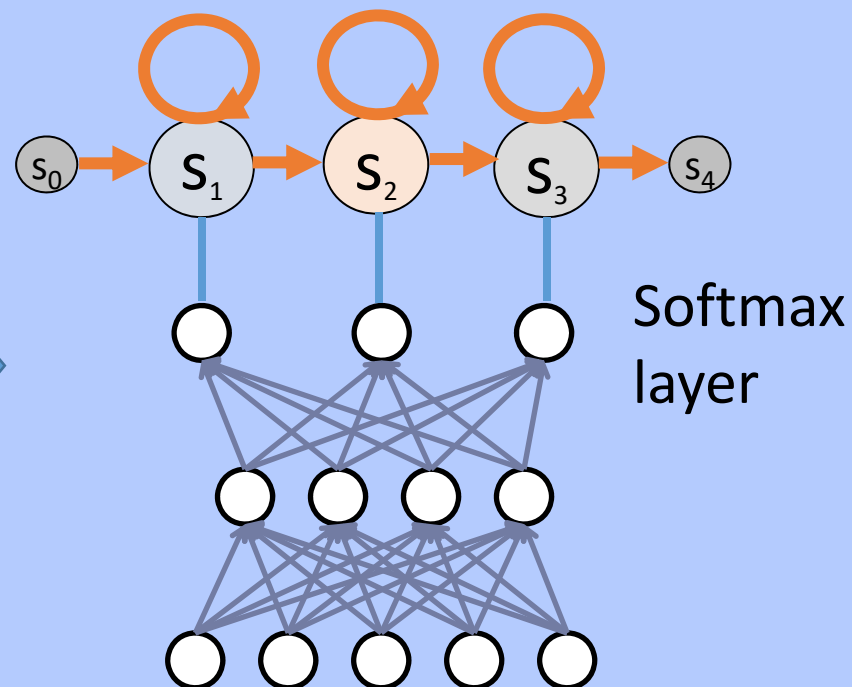
# HMM状態出力分布へのMLPの利用

$$p(X | s) = GMM_s(X)$$



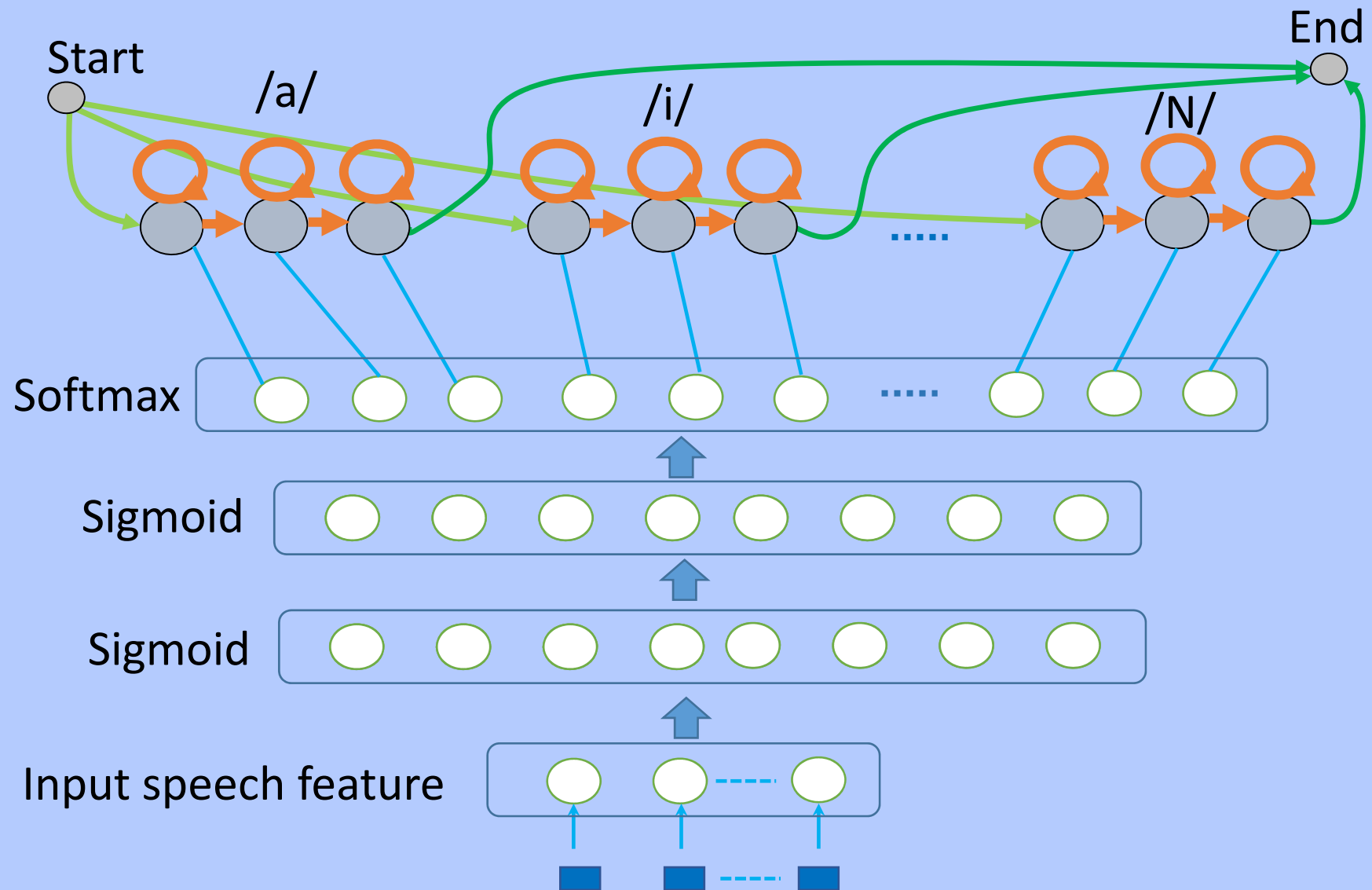
GMM-HMM

$$p(X | s) \propto \frac{p(s | X)}{p(s)} = \frac{MLP(s | X)}{p(s)}$$

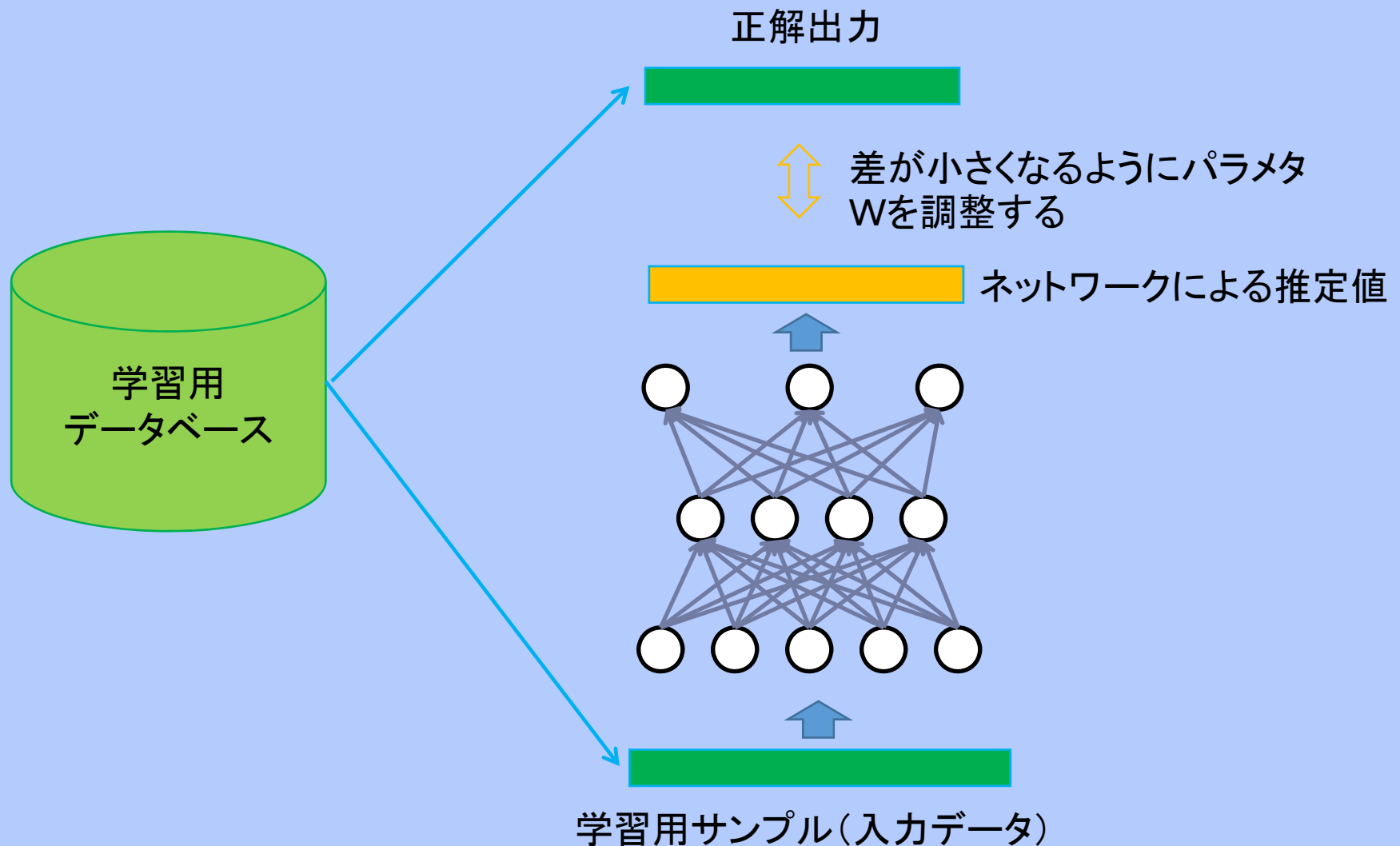


MLP-HMM

# 音素HMMでのMLPの利用



# 学習の枠組み



# 最急降下法によるパラメタ推定

- 最急降下法ではn次元の変数 $W$ を入力とする関数 $f=f(W)$ の局所最小解を以下により求める

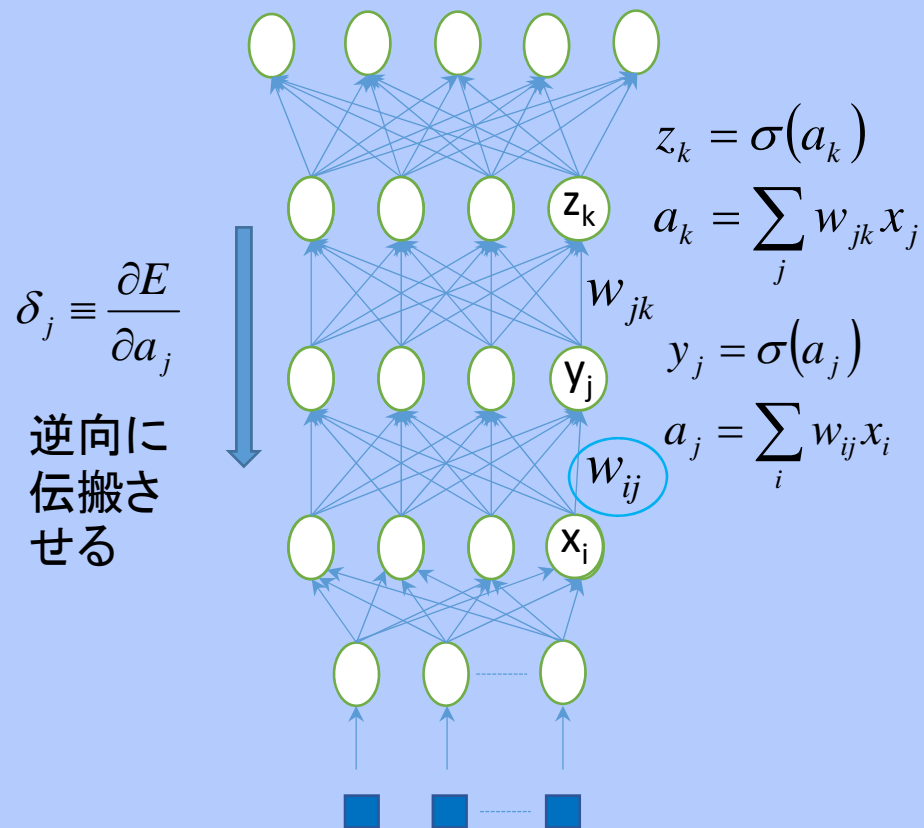
1.  $W$ の初期値を $W_0$ 決める
2. 次式により $W$ の各要素を更新する

$$W_{t+1} = W_t - \varepsilon \frac{\partial f(W)}{\partial W_t} \quad \varepsilon : \text{Learning rate (小さな正の数)}$$

3. 収束が得られるまでステップ2を繰り返す

# バックプロパゲーション

## • MLPにおける偏微分の評価を効率的に行う方法



1. 重み $w_{ji}$ の誤差関数 $E$ への影響は $a_j$ を必ず経由することから、以下が成り立つ

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \frac{\partial E}{\partial a_j} x_i$$

2. 和 $a_j$ の誤差関数 $E$ への影響は、 $y_j$ につながるいずれかのユニットを必ず経由することから、以下が成り立つ。すなわち $a_k$ が求まっていれば $a_j$ は容易に求まる

$$\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

3. ネットワークの出力側から入力側に向かって順に $\delta_j \equiv \frac{\partial E}{\partial a_j}$ を伝搬させていけば、フォワードプロパゲーションの結果と合わせることで全ての重みについての偏微分が効率的に求まる

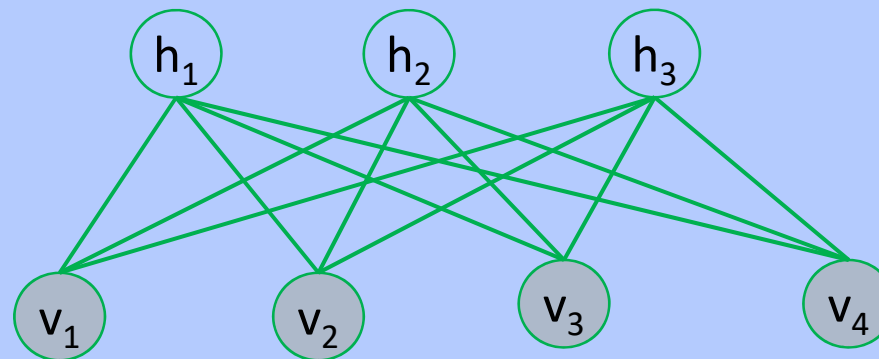
# 制約付きボルツマンマシン(RBM)

- 2部グラフの構造を持つボルツマンマシンで、片側が全て観測変数(可視層)、他方が全て隠れ変数(隠れ層)であるもの

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i,j} v_i w_{ij} h_j - \sum_i a_i v_i - \sum_j b_j h_j, \quad a_i, b_j : \text{bias}$$

$$p(h_j = 1 | \mathbf{v}) = \text{sigmoid} \left( \sum_i v_i w_{ij} + b_j \right), \quad p(v_i = 1 | \mathbf{h}) = \text{sigmoid} \left( \sum_j h_j w_{ij} + a_i \right)$$

RBMの例

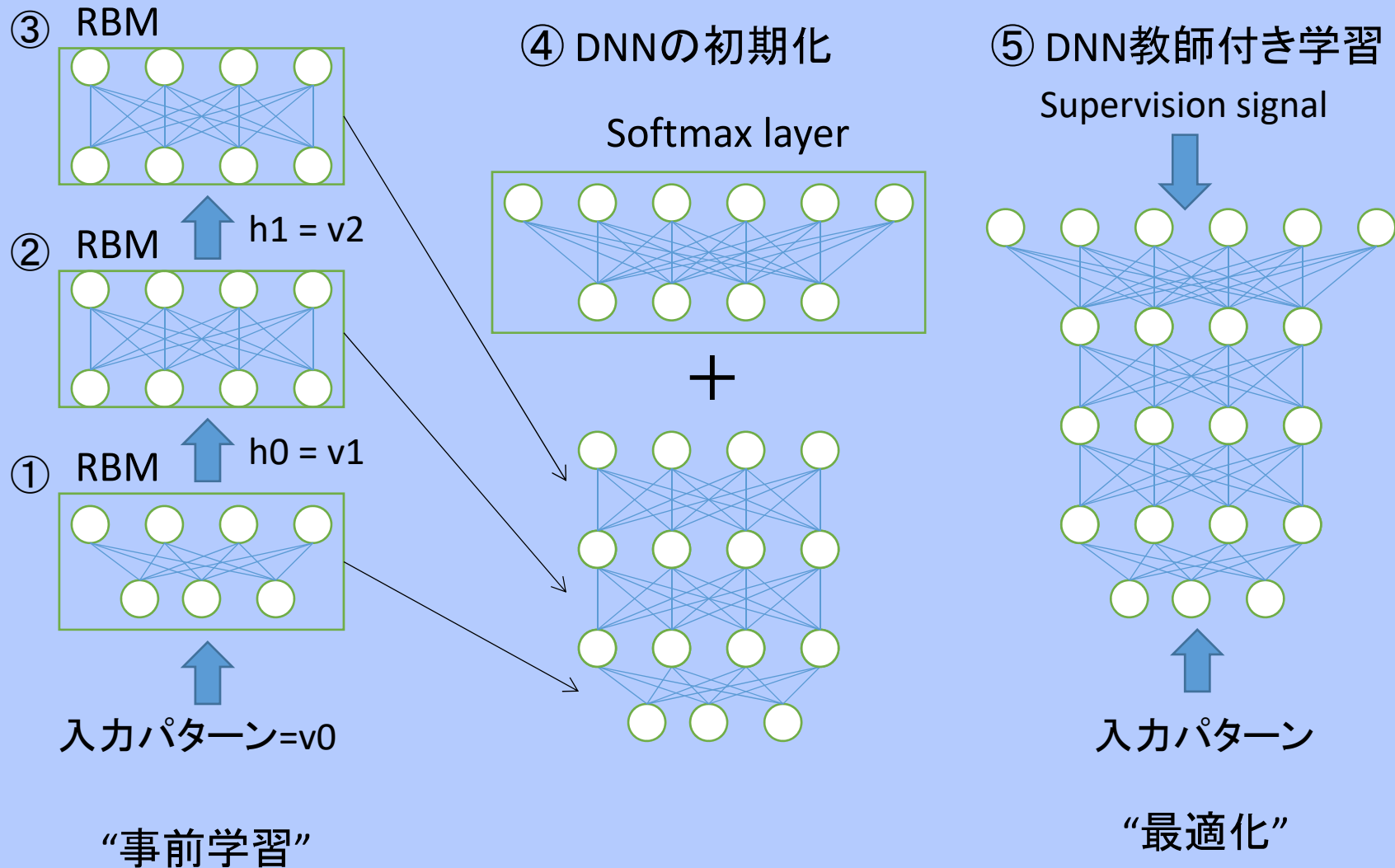


隠れ層

可視層

$$S = v_1, v_2, v_3, v_4, h_1, h_2, h_3$$

# RBM事前学習を用いたDNNの学習



# DNN-HMM音響モデルの学習

1. GMM-HMMを通常の手順で学習
2. 全学習データについて、HMM状態のフレームアライメントを求める
3. 入力特徴量からHMM状態を推定するDNNを学習。入力はセグメント特徴量とするのが一般的
4. GMM-HMMのGMMをDNNに置き換えDNN-HMMを得る

$$P(x_t | s_t) = \frac{P(s_t | x_t)}{P(s_t)} P(x_t) \approx \frac{P(s_t | x_t)}{P(s_t)} \cdot C, \quad C: \text{定数}$$


5. (DNN-HMMを用いた再アライメント)



# N-gram言語モデル

- 単語出現の条件付き確率のコンテキストを直近のN-1単語に限定したモデル

$$\begin{aligned} P(w_1 w_2 w_3 \cdots w_T) &= P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2) P(w_4 | w_1 w_2 w_3) \cdots P(w_T | w_1 w_2 \cdots w_{T-1}) \\ &= P(w_1) \prod_{t=2}^T P(w_t | w_1 \cdots w_{t-1}) \\ &\approx P(w_1) \prod_{t=2}^T P(w_t | w_{t-N+1} \cdots w_{t-1}) \end{aligned}$$

 **N-gram近似**  
(N-1単語以前の  
コンテキストを無視)

例(2-gram):

$$P(\text{本日 は 晴天 なり}) = P(\text{本日})P(\text{は} | \text{本日})P(\text{晴天} | \text{は})P(\text{なり} | \text{晴天})$$

例(3-gram):

$$P(\text{本日 は 晴天 なり}) = P(\text{本日})P(\text{は} | \text{本日})P(\text{晴天} | \text{本日 は})P(\text{なり} | \text{は 晴天})$$

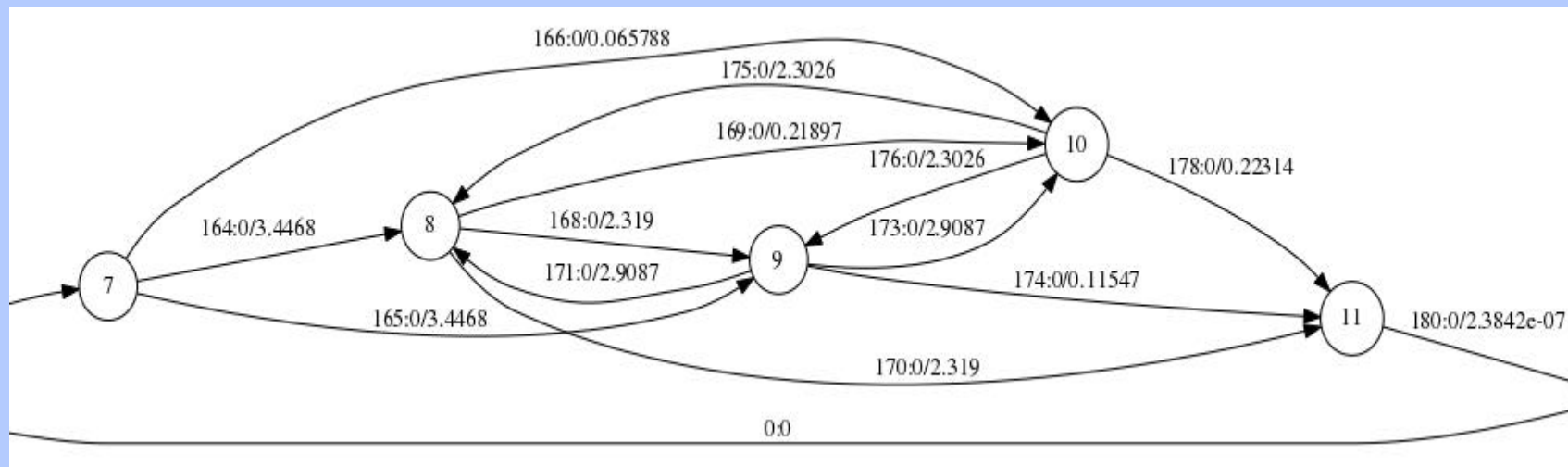
# デコーディング

- 各種モデルを統合して巨大な探索空間を構成
- 最小コストパス探索問題を解くことで認識単語列を求める

HMM  
音響モデル

発音辞書

N-gram  
言語モデル



# Kaldiツールキットの概要

# Kaldi 音声認識ツールキット

- 歴史:

- 2009年のJohns Hopkins University workshopが起源
- 国際的な開発チームにより非常に活発に開発が継続されている
- 最新の音声認識技術が多く取り入れられている

- 入手

- Githubから無料で配布
- <http://kaldi-asr.org/doc/index.html>
- Apache license, version 2.0

```
git clone https://github.com/kaldi-asr/kaldi.git kaldi --origin upstream
```

# Kaldi実行に必要な計算機のスペック

- 大語彙認識システムを構築する場合の例

CPU  
Core i7

メモリ  
32GByte



GPU\*  
GeForce  
GTX 970

\* ゲーム用のGPUを科学技術計算に用いるのはメーカーの保証外(自己責任)。動作保証が必要な場合はTesla K20Xなど

\*ディープニューラルネットワーク(DNN)の学習を行うのにGPUはほぼ必須

\*Kaldiのインストールの前にCUDAをインストール

\*コマンドの動作を試してみるだけならノートパソコンでも可

# Kaldiツールキットの構成

各種認識システム用スクリプト群(レシピ)  
英語、日本語、耐雑音、etc.  
(egs)

Kaldi  
独自コマンド  
(src)

外部ツールキット  
Openfst, IRSLM, etc.  
(tools)

# Kaldiのインストール

```
pikaia1 $ cd kaldi/
```

```
pikaia1 $ ls
```

```
COPYING INSTALL README.md egs misc src tools windows
```

```
pikaia1 $ less INSTALL # インストール手順の説明を読む
```

```
pikaia1 $ cd tools/ # 外部ツールキットの自動ダウンロードとコンパイル
```

```
pikaia1 $ make -j 4
```

```
pikaia1 $ cd ../src/ # kaldiコマンドのコンパイル
```

```
pikaia1 $ ./configure
```

```
pikaia1 $ make depend -j 4
```

```
pikaia1 $ make -j 4
```

# 一部手動ダウンロードの必要なツール

- 大部分の依存ツール(openfst等)はtoolsディレクトリに自動でダウンロードされインストールされますが、一部ライセンスの関係で手動ダウンロードの必要なツールもあります
- SRILMのインストール例
  - SRILMのソースコードを入手  
<http://www.speech.sri.com/projects/srilm/>
  - Toolsディレクトリに設置  
\$ mv srilm.tgz kaldi/tools
  - Kaldiに用意されているインストーラを用いてインストール  
\$ cd kaldi/tools  
\$ ./install\_srilm.sh



# Kaldiにおける入出力

- パイプラインが多用されている
- 各種ファイルをKaldi独自のark形式で保存
- Rspecifier/Wspecifierによる入出力の統一的な取扱い

# データの読み込み:Rspecifier

rspecifier	意味
ark:foo.ark	アーカイブファイルfoo.arkを読み込む
scp:foo.scp	スクリプトファイルfoo.scpを読み込む
ark:-	標準入力から入力する
ark:gunzip -c foo.ark.gz	圧縮されたfoo.ark.gzを読み込む

# データの書き込み: Wspecifier

wspecifier	意味
ark:foo.ark	アーカイブをfoo.arkに書き込む
scp:foo.scp	スクリプトをfoo.scpに書き込む
ark:-	(バイナリ形式で)標準出力に出力する
ark,t:-	テキスト形式で標準出力に出力する
ark,t: gzip -c > foo.gz	テキスト形式の出力を圧縮しfoo.gzに書込
ark,scp:foo.ark,foo.scp	アーカイブ・スクリプトの両形式で同時に書込

# 日本語話し言葉音声認識 のためのKaldi用CSJレシピ

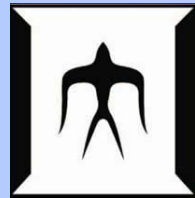
# CSJレシピを用いた日本語音声認識システムの構築

## 用意するもの

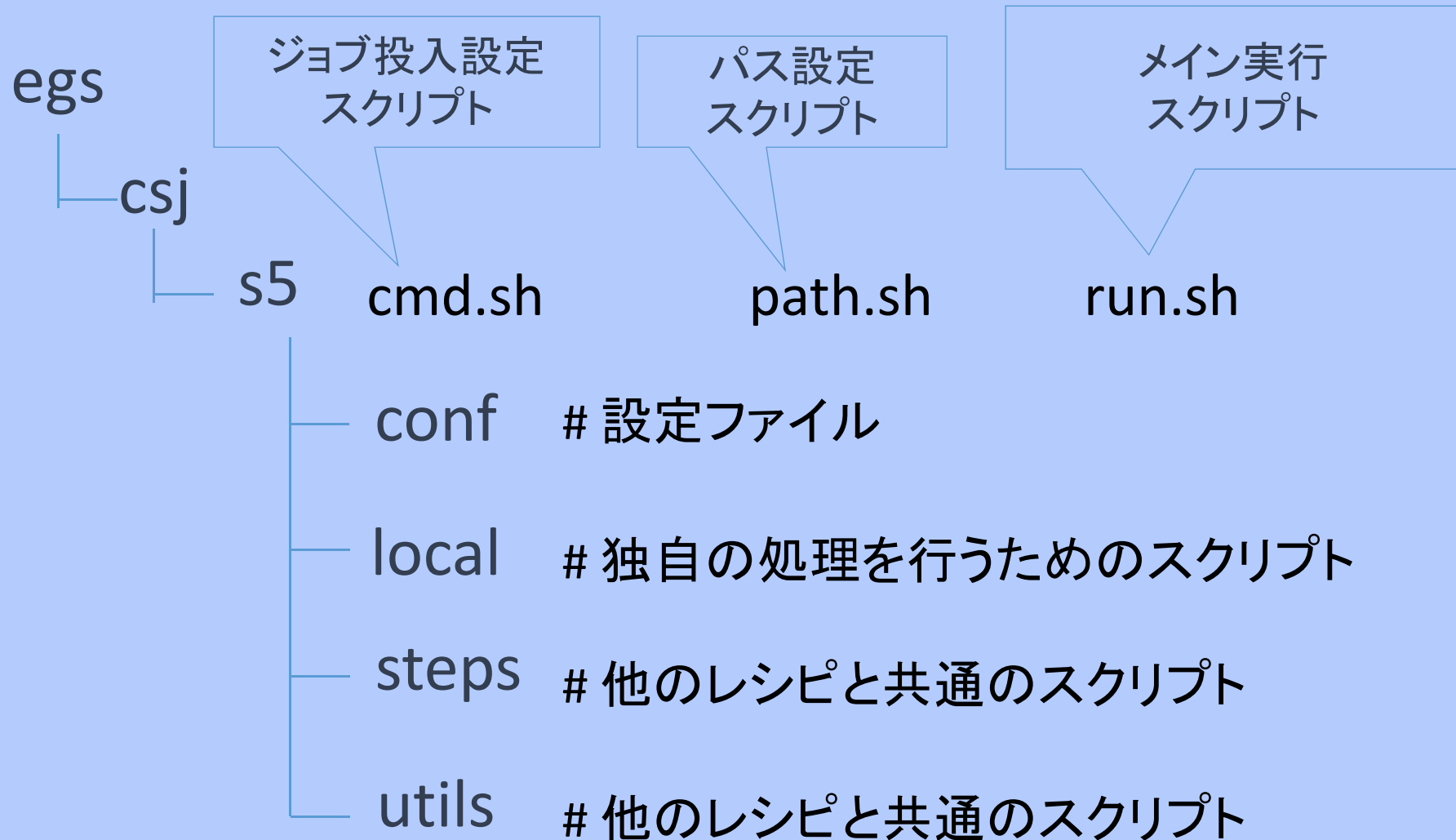
- KaldiがインストールされたLinuxマシン
  - 日本語を扱うためnkfも必要
- 日本語話し言葉コーパス(CSJ)
  - 音声データはKaldiに含まれていないので別途入手
  - CSJは国立国語研究所より購入可能  
[http://pj.ninjal.ac.jp/corpus\\_center/csj/](http://pj.ninjal.ac.jp/corpus_center/csj/)
- モデル学習に必要な時間
  - フルの学習を行う場合、先のスペックのPCで  
3-4週間

# CSJレシピについて

- 東工大篠崎研究室メンバーとアメリカMERL研究所渡部により共同開発
- DNN構造や学習条件などのシステム  
パラメタは東工大のスーパーコンピュータ  
TSUBAME2.5を用い、進化計算により最適化
- CSJ標準評価セットで91%超の認識精度を実現



# CSJレシピのディレクトリ構成

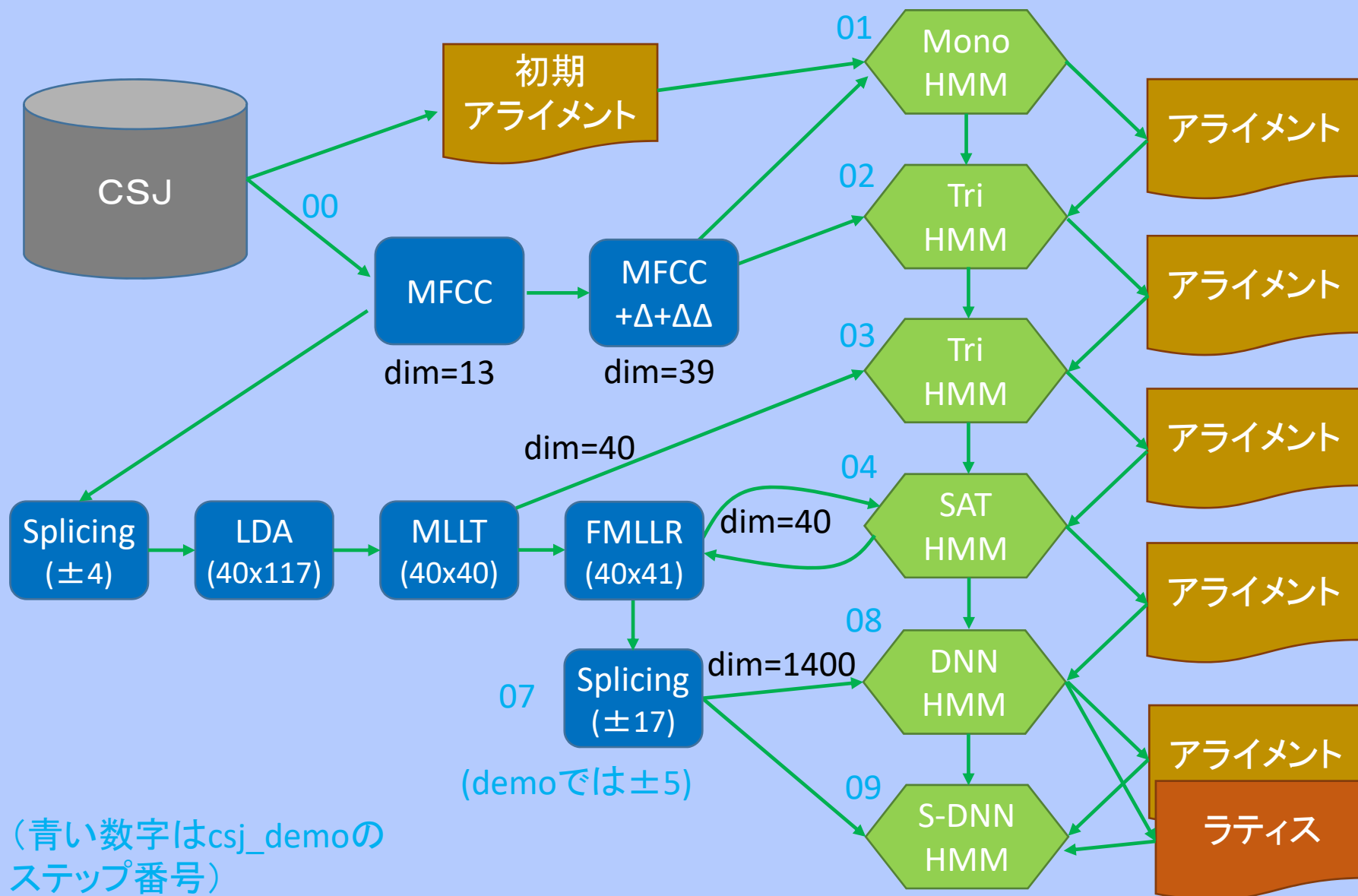


# 事前設定

- cmd.sh
  - バッチジョブシステム実行かローカル実行かを指定する  
(PCでローカル実行の場合はrun.plの使用を指定)
- path.sh
  - Kaldiパッケージをインストールした場所等の設定
- run.sh
  - CSJデータのパス(CSJDATATOP)
  - CSJのバージョン(CSJVER)



# CSJレシピの学習プロセス



# DNN音声認識システムの構築と認識実験

```
pikaia1 $
```

```
pikaia1 $
```

```
pikaia1 $
```

```
pikaia1 $ nohup ./run.sh >& run.log &
```

```
# Wait 3-4 weeks .....
```

```
.....
```

```
.....
```

# 認識結果の確認

- GMM-HMM, DNN-HMMの各学習ステージ毎に認識評価が自動で行われている
  - GMM-HMM (tri-phone)の単語誤り率の確認  
`less exp/tri3/decode_eval1_csj/wer_10`
  - DNN-HMM(系列学習)の単語誤り率の確認  
`less exp/dnn5b_pretrain-dbn_dnn_smbr_i1lats/decode_eval1_csj/wer_10`

単語誤り率(*WER*)

$$= \frac{\text{誤認識単語数}}{\text{正解単語数}} = \frac{\text{置換誤り} + \text{挿入誤り} + \text{削除誤り}}{\text{正解単語数}}$$

# CSJ以外のレシピについて

- データの入手先
  - 多くのデータは有料  
入手先: LDC等  
<https://www ldc.upenn.edu/>
  - 一部無料のデータあり  
tedlium, voxforge, etc.
- 実行時の注意
  - SWB等レシピ開発がクラスタマシン上で行われているものがあり、デフォルトの並列実行数のままPC上で実行すると過負荷でマシンが落ちる。。。

# デモ用CSJレシピを用いた チュートリアル

# デモ用CSJレシピ (csj\_demo)

- 概要:

- ノートパソコンで手軽に動作確認できるように使用するデータをぎりぎりまで切り詰め、また初心者の学習用に各コマンドを独立して試せるようにしたもの
- デコーディングの際にラティスを作成しないようにしている以外は、基本的に一般公開版と同じ動作手順

- 入手方法

- 篠崎研究室のホームページで公開しています  
<http://www.ts.ip.titech.ac.jp/demos/index.html>

\*大学の組織改革で、近いうちにURLを変更する予定です  
私の所属は東京工業大学・工学院・情報通信系です

# メインスクリプトと確認用スクリプト

- メインスクリプト
  - ファイル/ディレクトリ名の番号順にスクリプトを実行することで、GMM-HMMおよびDNN-HMMの学習とそれらを用いた認識が実行される
- ファイル名にsynopが含まれているスクリプト
  - 基本となるコマンドをコマンドラインから手で入力して動作を確認するためのもの
  - 実行結果は全てsynoptmpディレクトリに保存され、後段ステップには影響しない

# 各ステップの概要

- A) 00\_\* : データの前処理と特徴量抽出
- B) 01\_\* - 06\_\* : GMM-HMMの学習
- C) 07\_\* - 09\_\* : DNN-HMMの学習



# A) データの前処理と特徴量抽出

# 本ステップの概要

- 1.sdbファイルからラベルファイルを作成
- 2.言語モデルの学習
- 3.音声特徴量の抽出

# 実行方法

```
$ cd ~/kaldi/egs/csj_demo/s5_demo1  
$ ./00_0prep_csj2kaldi.sh  
$ ./00_1make_mfcc_cmvn.sh
```

# 音素/HMM状態セットの構成

- 音素セット
  - 単語内の位置で場合分けしている
    - Begin (e.g. a\_B)
    - End (e.g. a\_E)
    - Internal (e.g. a\_I)
    - Singleton (e.g. a\_S)
- HMM状態数
  - utils/prepare\_lang.shで設定

# 音声データの切り出し

- 切り出したwavデータをアーカイブ形式で保存

実行例 (00\_1mfcc\_synop.shを参照)

行末の¥は改行を  
無いことにする

```
$ source path.sh
$ extract-segments ¥
  scp,p:wav.scp(wavデータのIDおよびパス) ¥
  segments(セグメントの時間情報) ¥
  ark:extract_segment(出力先)
```

# 特徴量の抽出

- MFCC特徴量の抽出

実行例 (00\_1mfcc\_synop.shを参照)

```
$ compute-mfcc-feats ¥  
  --config=mfcc.conf(コンフィグファイル) ¥  
  ark:extract_segment(wavデータ) ¥  
  ark:mfcc(出力先)
```

# CMVN正規化係数の計算

- CMVN統計量の計算

実行例 (00\_1mfcc\_synop.shを参照)

```
$compute-cmvn-stats ¥  
  --spk2utt=ark:spk2utt(話者ごとの発話リスト) ¥  
  ark:mfcc(mfccファイル) ¥  
  ark:cmvn(出力先)
```

# Wavデータのセグメント情報の参照

```
$ wav-to-duration ark:extract_segment ark,t:destfile
```

```
[kalditest@suzukake s5_demo1]$ wav-to-duration  
ark:synoptmp/extract_segment ark,t:- | head  
wav-to-duration ark:synoptmp/extract_segment a  
rk,t:-  
A01M0007_0000082_0001655 1.573  
A01M0007_0002264_0003113 0.849  
A01M0007_0003905_0009286 5.381
```



# 特徴量ファイルのテキスト化

```
$ copy-feats ark:mfcc.ark ark,t:destfile
```

```
[kalditest@suzukake s5_demo1]$ copy-feats ark:mfcc/raw_mfcc_train.1.ark ark,t:- | head
copy-feats ark:mfcc/raw_mfcc_train.1.ark ark,t:-
A01M0007_0000082_0001655 [
  52.69612 -18.56213 -7.228602 -8.509803 1.596313 1.16476 -24.81916 18.5693 8.05776 12.
5.269132 -11.20466
```

```
$ copy-feats ark:cmvn.ark ark,t:destfile
```

```
[kalditest@suzukake s5_demo1]$ copy-feats ark:mfcc/cmvn_train.ark ark,t:- | head
copy-feats ark:mfcc/cmvn_train.ark ark,t:-
A01M0007 [
  1.009269e+07 98301.73 -267101.7 363554.7 -1020593 -903582.4 -664024.4 -1364881
377120.6 -580517.9 -409309.8 277517.5 -1468187 133815
```

(正規化ファイルも特徴量と同じフォーマット)

## B) GMM-HMMの学習

# 本ステップの概要

1. モノフォンの学習
2. トライフォンの学習
3. 特徴量変換(LDA,MLLT)を用いた  
トライフォンの学習
4. fMLLRを用いた話者適応学習(SAT)
5. bMMIによる識別学習
6. bMMI + f-bMMIによる識別学習

# モノフォンの学習

# 実行方法

```
$ cd 01_mono  
$ ./01_run.sh
```

# 処理の流れ

steps/train\_mono.sh

- i. モノフォンモデルの学習準備と初期化
- ii. モノフォンのEM学習

utils/mkgraph.sh

- iii. 認識用WFSTの作成

steps/decode\_nolat.sh

- iv. 評価セットの認識

## i. モノフォンモデルの学習準備と初期化

1. MFCCにCMVNを適用し、動的特徴量を付加
2. 初期モデルを作成
3. 発話ごとのFSTファイルを作成(発話ラベル)
4. 初期パラメタの簡易推定
  1. アライメントファイル(均等分割)の作成
  2. 統計量の蓄積
  3. 最尤推定によるパラメタ更新

# CMVNの適用と動的特徴量の付加

実行例 (01\_synops.sh を参照)

```
$ apply-cmvn ¥  
  --utt2spk=ark:utt2spk(発話と話者のマッピング) ¥  
  scp:cmvn.scp(cmvn正規化定数のリスト) ¥  
  scp:feats.scp(特徴量のリスト) ¥  
  ark:-(標準出力) | ¥  
  add-deltas ¥  
  ark:-(標準入力) ¥  
  ark:feats_norm(出力先)
```

パイプで2つの  
コマンドを接続



# 初期モデルの作成

実行例 (01\_synops.sh を参照)

```
$ gmm-init-mono ¥  
  --shared-phones=sets.int(確率分布を共有する音素) ¥  
  --train-feats=ark:feats_norm(特徴量) ¥  
  topo(HMMトポロジー) ¥  
  39(特徴量の次元数) ¥  
  0.mdl(初期モデルの出力先) ¥  
  tree(treeファイルの出力先)
```

# 発話ごとのFSTファイルを作成

実行例 (01\_synops.sh を参照)

```
$ compile-train-graphs ¥  
  tree (treeファイル) ¥  
  0.mdl (モデルファイル) ¥  
  L.fst (辞書のFSTファイル) ¥  
  ark:text.int (textのint形式ファイル) ¥  
  ark:fsts (出力先)
```

# 均等分割アライメントの作成

実行例 (01\_synops.sh を参照)

```
$ align-equal-compiled ¥  
ark:fsts(発話ごとのFST) ¥  
ark:feats_norm(特徴量ファイル) ¥  
ark,t:align_equal_compiled(出力先)
```

# 統計量の蓄積

実行例 (01\_synops.sh を参照)

```
$ gmm-acc-stats-ali ¥  
0.mdl(初期モデル) ¥  
ark:feats_norm(特徴量ファイル) ¥  
ark:align_equal_compiled(アライメントファイル) ¥  
0.acc(出力先)
```

# 最尤推定によるパラメタ更新

実行例 (01\_synops.sh を参照)

```
$ gmm-est ¥  
  --mix-up=136(総混合数) ¥  
  0.mdl(入力モデル) ¥  
  0.acc(入力統計量) ¥  
  1.mdl(出力先)
```

## ii. モノフォンのEM学習

初期パラメタの簡易推定の後、EM学習を行う

- ① アライメントをとる
- ② 統計量を蓄積する
- ③ 混合数を増加/モデルを更新する

①から③を繰り返して学習を行う

# アライメントの計算

実行例 (01\_synops.sh を参照)

```
$ gmm-align-compiled ¥  
  --transition-scale=1.0(遷移確率重み) ¥  
  --acoustic-scale=0.1(音響尤度重み) ¥  
  --beam=6 (beam幅) ¥  
  --retry-beam=24(二度目のbeam幅) ¥  
  1.mdl_bsil(モデル) ¥  
  ark:fsts(FSTファイル) ¥  
  ark:feats_norm(特徴量ファイル) ¥  
  ark,t:ali.1(出力先)
```

# 統計量の蓄積

実行例 (01\_synops.sh を参照)

```
$ gmm-acc-stats-ali ¥  
  1.mdl(モデル) ¥  
ark:feats_norm(特徴量ファイル) ¥  
ark:ali.1(アライメントファイル) ¥  
  1.acc(出力先)
```



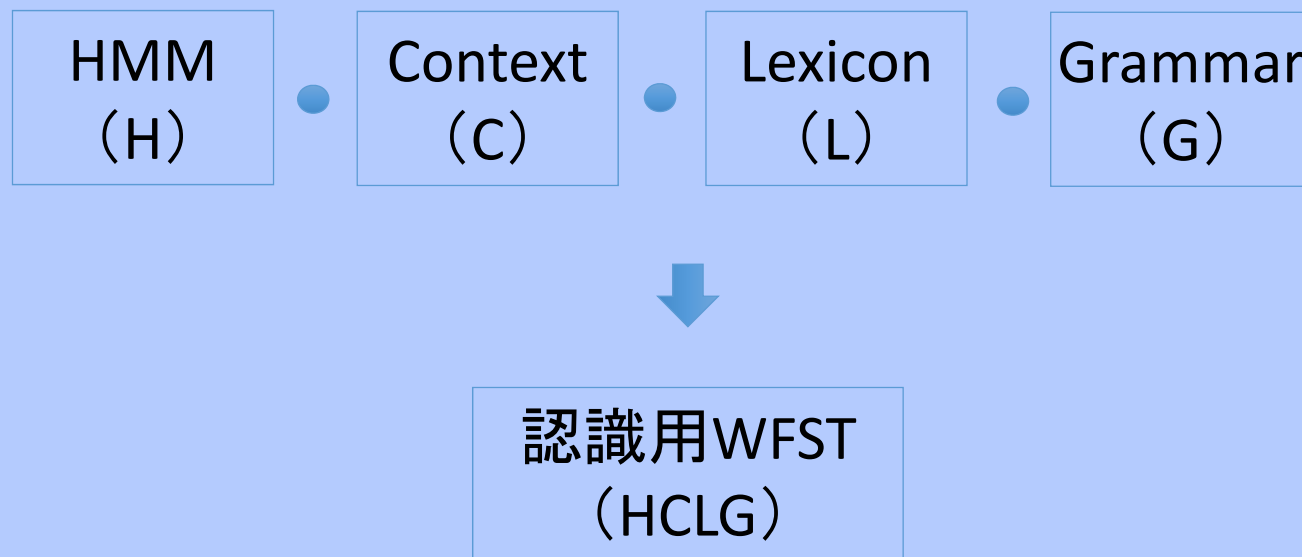
# 混合数の増加/モデル更新

実行例 (01\_synops.sh を参照)

```
$ gmm-est ¥  
  --write-occs=2.occs (十分統計量の出力先) ¥  
  --mix-up=136 (総混合数) ¥  
  1.mdl (モデル) ¥  
  1.acc (統計量) ¥  
  2.mdl (出力先)
```

### iii. 認識用WFSTの作成

- HMMの状態遷移:H、(音素コンテキスト:C)、辞書:L、言語モデル:GをそれぞれWFSTとして表現後、一つのWFST:HCLGに合成



## iv. 評価セットの認識

- 認識にはラティスあり/なしの2種類の方法がある(本実習では主に実行の早いラティスなしを利用)
- ラティスあり: `gmm-latgen-faster`  
(`steps/decode_nolats.sh`)
- ラティスなし: `gmm-decode-faster`  
(`steps/decode.sh`)
- 認識結果の計算には`compute-wer`を用いる  
(品詞を削除するために`local/wer_hyp_filter`を使用)

# ラティスを作成する認識(GMM-HMM)

gmm-latgen-faster ¥

--max-active=7000(デコーダの最大アクティブ) ¥

--beam=11.0(ビーム幅) ¥

--lattice-beam=6.0(ラティスビーム) ¥

--acoustic-scale=0.083333(音響尤度の比率) ¥

--word-symbol-table=words.txt(出力シンボル) ¥

HCLG.fst(FST) ¥

“ark,s,cs:apply-cmvn --utt2spk=ark:utt2spk  
scp:cmvn.scp scp:feats.scp ark:- | add-deltas ark:-  
ark:- |”(特徴量) ¥

“ark:|gzip -c > lat.1.gz”(出力先)

Rspecifierにパイプライン  
を指定している

Wspecifierにパイプライン  
を指定している

# ラティスを作成しない認識(GMM-HMM)

`gmm-decode-faster` ¥

`--max-active=7000` (デコーダの最大アクティブ) ¥

`--beam=11.0` (ビーム幅) ¥

`--acoustic-scale=0.083333` (音響尤度の比率) ¥

`--word-symbol-table=words.txt` (出力シンボル) ¥

`HCLG.fst` (FST) ¥

`"ark,s,cs:apply-cmvn --utt2spk=ark:utt2spk  
scp:cmvn.scp scp:feats.scp ark:- | add-deltas ark:-  
ark:- |"` (特徴量) ¥

`"ark:|gzip -c > words.1.gz"` (出力先)

# 認識率の計算

```
compute-wer --text --mode=present ¥  
ark:ref.txt ark,p:hyp.txt
```

正解文

認識結果

present: 認識結果に含まれる発話のみを評価  
all: 認識結果が無い場合、結果が空として集計  
strict: 認識結果が無い場合エラー終了

# バイナリ・テキスト変換

- 決定木ファイル

```
$ copy-tree --binary=false tree destfile
```

- GMM-HMM 統計量ファイル

```
$ gmm-sum-accs --binary=false destfile hmm.acc
```

- GMM-HMM モデルファイル

```
$ gmm-copy --binary=false hmm.mdl destfile
```

- ラティスファイル

```
$ lattice-copy ark:foo.lat ark,t:destfile
```

```
$ lattice-copy "ark:gunzip -c foo.lat.gz |" ark,t:destfile
```

# ファイル情報の表示

- GMM-HMMファイル: `gmm-info foo.mdl`  
: `hmm-info foo.mdl`

```
[kalditest@suzukake s5_demo1]$ gmm-info 01_mono/exp/mono/final.mdl
gmm-info 01_mono/exp/mono/final.mdl
number of phones 166
number of pdfs 127
number of transition-ids 1116
number of transition-states 518
feature dimension 39
number of gaussians 1001
```

- 特徴量の次元数: `feat-to-dim scp:feats.scp -`

```
[kalditest@suzukake s5_demo1]$ feat-to-dim
scp:mfcc/raw_mfcc_eval1.1.scp -
feat-to-dim scp:mfcc/raw_mfcc_eval1.1.scp -
13
```

- 決定木ファイル: `tree-info tree`

```
[kalditest@suzukake s5_demo1]$ tree-info 01_mono/exp/mono/tree
tree-info 01_mono/exp/mono/tree
num-pdfs 127
context-width 1
central-position 0
```



# アライメントの確認

```
$ show-alignments phones.txt foo.mdl ark:ali.*
```

```
[kalditest@suzukake s5_demo1]$ show-alignments data/lang_nosp/phones
.txt 01_mono/exp/mono/final.mdl "ark:gunzip -c 01_mono/exp/mono/ali.
*.gz |" | head -n 3
show-alignments data/lang_nosp/phones.txt 01_mono/exp/mono/final.mdl
'ark:gunzip -c 01_mono/exp/mono/ali.*.gz |'
```

```
A01M0007_0000082_0001655 [ 494 493 493 493 493 496 495 495 495 495
495 495 495 495 495 495 495 495 495 495 495 495 495 498 497 497 497
497 ] [ 914 913 916 915 915 915 915 915 915 915 915 915 915 915 915 915
915 915 915 915 915 915 918 ] [ 986 988 990 989 989 ] [ 860 859 859
862 864 ] [ 758 757 757 757 757 757 760 759 762 761 761 ] [ 866 865
865 868 870 ] [ 842 841 844 843 843 843 846 845 ] [ 860 859 862 861
861 861 861 861 861 861 861 861 861 861 861 861 861 861 861 861 861
861 861 861 861 861 861 861 861 861 861 861 861 861 861 861 861
861 861 861 861 861 861 861 864 863 863 863 863 863 863 863 863 ]
A01M0007_0000082_0001655 e:_B
```

q\_I

t\_I

o\_E

k\_B

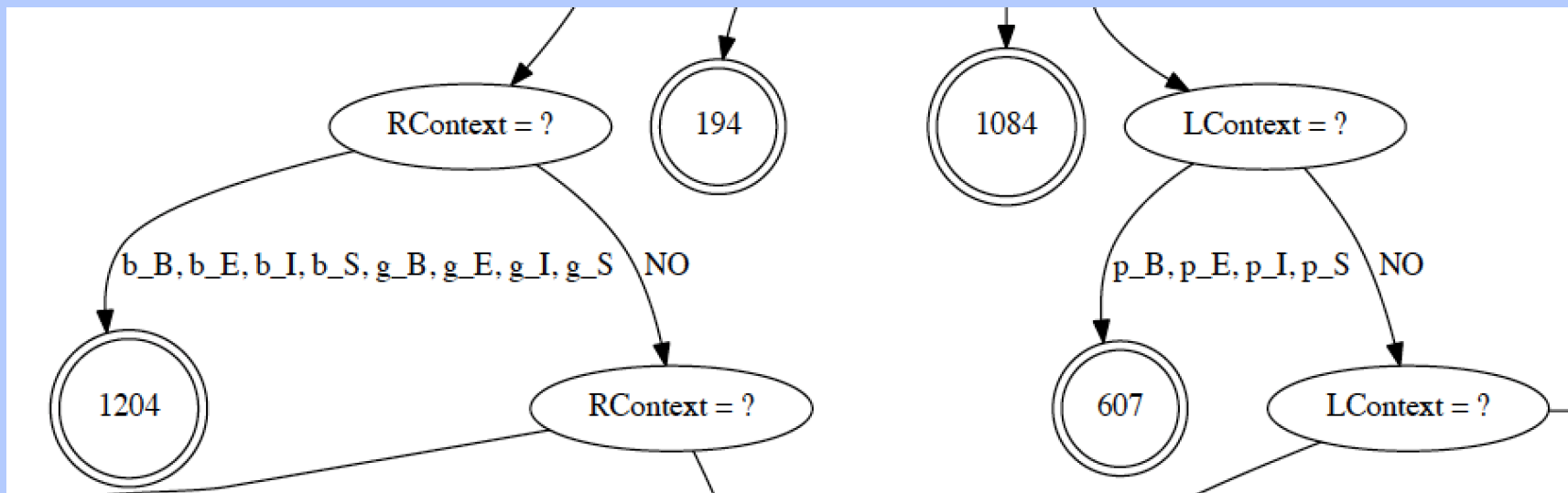
o\_I

n\_I

o\_E

# 状態決定木ファイルのグラフ化

```
$ draw-tree phones.txt tree | ¥  
dot -Tps - Gsize=8,10.5 | ¥  
ps2pdf - tree.pdf
```



\*graphvizのインストールが必要

# トライフォンの学習

# 実行方法

```
$ cd ../02_delta  
$ ./01_run.sh
```

# 処理の流れ

steps/train\_deltas.sh

- i. 決定木の作成とトライフォンの初期化
- ii. トライフォンのEM学習

utils/mkgraph.sh

- iii. 認識用WFSTの作成

steps/decode\_nolat.sh

- iv. 評価セットの認識

# 決定木用統計ファイルの作成

- モノフォンをもとにトライフォン統計量を蓄積

実行例 (01\_synops.sh を参照)

```
$ acc-tree-stats ¥  
  --ci-phones=1:2:3:4:5:6:7:8:9:10(無音のID) ¥  
  final.mdl(モノフォンの最終のモデル) ¥  
  ark:feats_norm(特徴量) ¥  
  "ark:gunzip -c ali.1.gz|" (アライメント) ¥  
  treeacc(出力先)
```

# 決定木用質問リストの自動生成

- 統計量から音素コンテキストの質問を作成

実行例 (01\_synops.sh を参照)

```
$ cluster-phones ¥  
  treeacc(トライフォン統計量) ¥  
  sets.int(全音素のint形式リスト) ¥  
  questions.int(出力先)
```

# 決定木状態クラスタリング

- 決定木によりトライフォン状態をクラスタリング

実行例 (01\_synops.sh を参照)

```
$ build-tree ¥  
  --max-leaves=600(最大リーフノード数) ¥  
  treeacc(トライフォン統計量) ¥  
  roots.int(根の定義ファイル) ¥  
  questions.qst(質問リスト) ¥  
  topo(トポロジー) ¥  
  tree(出力先)
```



# 状態共有トライフォンモデルの作成

- 決定木に従いトライフォン状態を共有化

実行例 (01\_synops.sh を参照)

```
$ gmm-init-model ¥  
  --write-occs=1.occs (十分統計量の出力先) ¥  
  tree (treeファイル) ¥  
  treeacc (トライフォン統計量) ¥  
  topo (HMMトポロジー定義ファイル) ¥  
  1.mdl (モデル出力先)
```

# トライフォンのEM学習

- 基本的にモノフォンと同様
  - ① 強制アライメントをとる
  - ② 統計量を蓄積する
  - ③ 混合数を増加/モデルを更新する
- ①から③を繰り返して学習を行う

# 特徴量変換とトライフォンの再学習

- スプライシングによりフレームを連結した特徴量を作成
- LDAにより次元圧縮
- MLLTにより特徴量ベクトルの相関を削減
- 新たな特徴量でトライフォンを再学習

## 実行方法

```
$ cd ../03_feats_trans  
$ ./run.sh
```

# 話者適応学習(SAT)

- 話者適応技術(fMLLR)を応用し、話者の特性を正規化した特徴量を作成
- 新たな特徴量でトライフォンを再学習

## 実行方法

```
$ cd ../04_sat  
$ ./run.sh
```

# GMM-HMMの識別学習

- bMMIによる識別学習  
(steps/train\_mmi.sh)
- f-bMMIとbMMIによる識別学習  
(steps/train\_mmi\_fmml.sh)

※識別学習を行うにはラティスが必要  
(GMM-HMMにおけるラティスは  
steps/make\_denlats.shで作成する)

# 実行方法

```
$ cd ../05_mmi  
$ ./run.sh  
$ cd ../06_fmml  
$ ./run.sh
```

※後段のDNN学習はSATモデルを元に行っているので識別学習ステップは関係せず、省略可

## C) DNN-HMMの学習

# 本ステップの概要

- 1.DNN-HMM学習用に予めfMLLRを適用したデータを作成しディスクに保存
- 2.DNNのプレトレーニングとファインチューニング
- 3.系列識別学習を用いたDNNの再学習



# fMLLRを適用したデータの作成

- 何度も使うので予め作成しておく

## 実行方法

```
$ cd ..  
$ ./07_prep_dnn.sh
```

# プレトレーニングとファインチューニング

## 実行方法

```
$ cd 08_dnn  
$ ./01_run.sh
```

# 処理の流れ

steps/nnet/pretrain\_dbn.sh

- i. RBMの積み重ねによるプレトレーニング

steps/nnet/{train.sh,train\_scheduler.sh}

- ii. クロスエントロピーを目的関数とした  
バックプロパゲーションによる  
ファインチューニング

steps/nnet/decode.sh

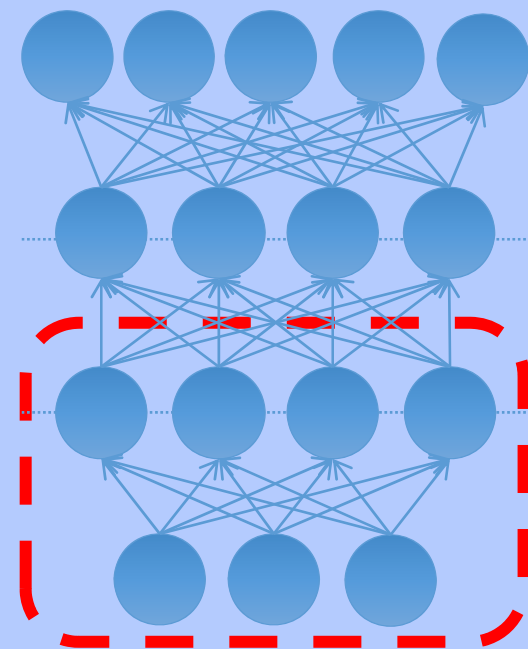
- iii. 評価セットの認識

## i. プレトレーニング

- 入力層の学習
- 隠れ層の学習

# 入力層の学習

- Splice, CMVNをNnet形式にエンコード
- GB-RBMの初期化
- Contrastive Divergence(CD)法による学習
- 学習したRBMをDBN形式に変換



# Splice,CMVNをNnet形式にエンコード

実行例 (01\_1synop\_pre-train.sh を参照)

```
$ nnet-forward ¥  
tr_splice5-1.nnet(spliceの定義) ¥  
ark:"copy-feats scp:train.scp ark:- |"(特徴量) ¥  
ark:-(標準出力) | ¥  
compute-cmvn-stats ¥  
ark:-(標準入力)-(標準出力) | ¥  
cmvn-to-nnet -(標準入力)-(標準出力) | ¥  
nnet-concat ¥  
--binary=false(出力形式の指定) ¥  
tr_splice5-1.nnet(spliceの定義)-(標準入力) ¥  
tr_splice5-1_cmvn-g.nnet(出力先)
```

# GB-RBMの初期化

## 実行例 (01\_1synop\_pre-train.sh を参照)

```
$ echo "<NnetProto>  
  <Rbm> <InputDim> 440 <OutputDim> 256 <VisibleType> gauss  
<HiddenType> bern <ParamStddev> 0.1  
  </NnetProto>  
" > 1.rbm.proto  
$ nnet-initialize ¥  
  1.rbm.proto(入力層の形式) ¥  
  1.rbm.init(出力先)
```

# CD法によるパラメタ推定

実行例 (01\_1synop\_pre-train.sh を参照)

```
$ rbm-train-cd1-frmshuff ¥  
--learn-rate=0.01 (学習係数) ¥  
--l2-penalty=0.0002 (L2正則化係数) ¥  
--num-iters=2 (エポック数) ¥  
--feature-transform=tr_splice5-1_cmvn-g.nnet ¥ (特徴量変換)  
1.rbm.init (入力RBM) ¥  
ark:"copy-feats scp:train.scp ark:- |" (特徴量) ¥  
1.rbm (更新したRBMの出力先)
```



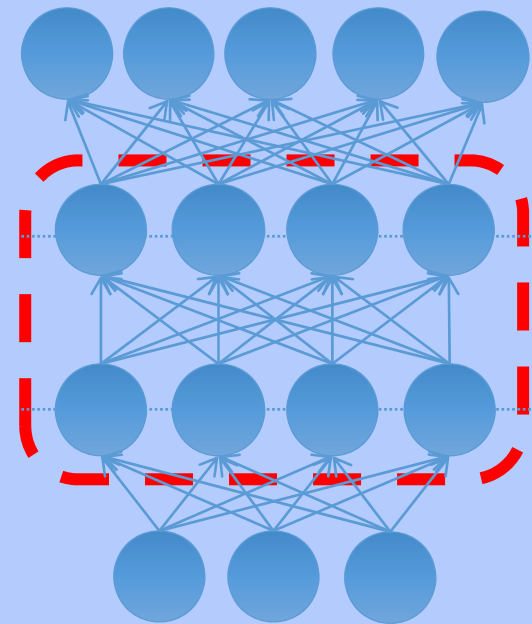
# 学習したRBMをDBNに変換

実行例 (01\_1synop\_pre-train.sh を参照)

```
$ rbm-convert-to-nnet ¥  
  --binary=true(出力形式) ¥  
  1.rbm(入力RBM) ¥  
  1.dbn(DBN出力先)
```

# 隠れ層の学習

- ① 前段出力の正規化係数の計算
- ② BB-RBMの初期化
- ③ CD法による学習
- ④ 学習したRBMをDBNに変換し、連結



指定した階層数になるまで①から④を繰り返す

# 前段出力の正規化係数の計算

実行例 (01\_2synop\_pre-train.sh を参照)

```
$ nnet-forward ¥  
"nnet-concat tr_splice5-1_cmvn-g.nnet 1.dbn - |" ¥  
(入力層の情報)  
ark:"copy-feats scp:train.scp ark:- |" (特徴量) ¥  
ark:- (標準出力) | ¥  
compute-cmvn-stats ark:- (標準入力) - (標準出力) | ¥  
cmvn-to-nnet - (標準入力) $depth.cmvn (出力先)
```

# BB-RBMの初期化

実行例 (01\_2synop\_pre-train.sh を参照)

```
$ echo "<NnetProto>  
  <Rbm> <InputDim> 256 <OutputDim> 256 <VisibleType> bern  
<HiddenType> bern <ParamStddev> 0.1  
<VisibleBiasCmvnFilename> $depth.cmvn  
  </NnetProto>  
" > depth.rbm.proto  
  
$ nnet-initialize ¥  
  depth.rbm.proto(中間層の情報) ¥  
  depth.rbm.init(出力先)
```

# 学習したRBMをDBNに変換・結合

実行例 (01\_2synop\_pre-train.sh を参照)

```
$ rbm-convert-to-nnet ¥  
  --binary=true(出力形式) ¥  
  depth.rbm(RBM) -(標準出力) | ¥  
  nnet-concat (depth-1).dbn(1層下までのRBM) ¥  
  -(標準入力) ¥  
  depth.dbn(出力先)
```

## ii. ファインチューニング

- PDF-IDを用いたアライメントの作成
- PDFの出現回数をカウント(事前確率推定)  
(ベイズの定理を用いてDNN出力をHMM状態出力確率に変換するため)
- 出力層の作成
- 初期DNNの構成
- アライメントからDNN出力側データの作成
- ミニバッチを用いたバックプロパゲーション

# PDF-IDを用いたアライメント作成

実行例 (01\_3synop\_fine-tuning.shを参照)

```
$ ali-to-pdf ¥  
  final.mdl(モデル) ¥  
  “ark:gunzip -c ali.*.gz |”(アライメント) ¥  
  ark:pdf_ali(出力先)
```

# PDFの出現回数のカウント

実行例 (01\_3synop\_fine-tuning.shを参照)

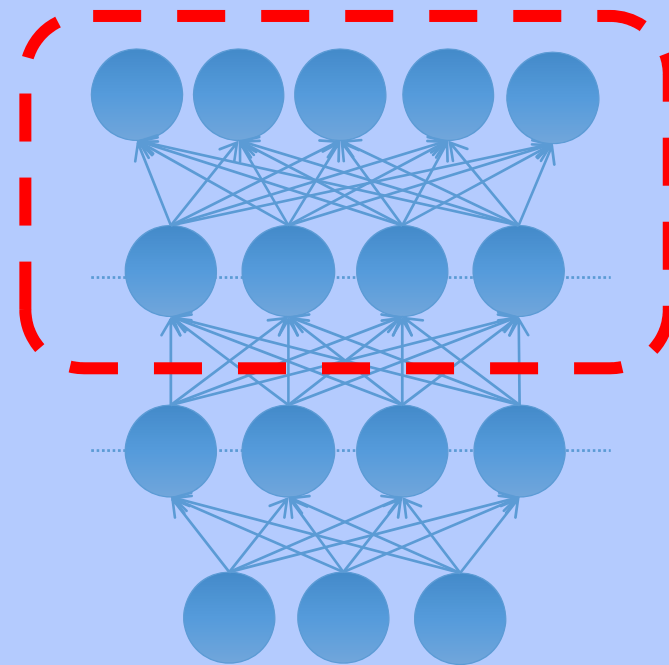
```
$ analyze-counts ¥  
  --binary=false(出力形式) ¥  
  ark:pdf_ali(PDF-IDを用いたアライメント) ¥  
  ali_train_pdf.counts(出力形式)
```



# 出力層の作成

実行例 (01\_3synop\_fine-tuning.shを参照)

```
$ utils/nnet/make_nnet_proto.py ¥  
  num_fea(入力側の素子数) ¥  
  num_tgt(出力層の数) ¥  
  0(隠れ層の数: 0=output) ¥  
  1(隠れ層の素子数: 1->ダミー) ¥  
  > nnet.proto(出力先)  
$ nnet-initialize ¥  
  nnet.proto(出力層の情報) ¥  
  nnet.init(出力先)
```



# 初期DNNの構成

実行例 (01\_3synop\_fine-tuning.shを参照)

```
$ nnet-concat ¥  
  dbn(中間層のDBN) ¥  
  nnet.init(出力層の情報) ¥  
  nnet_depth.dbn_dnn.init(Nnet出力先)
```

# DNN出力側データの作成

実行例 (01\_3synop\_fine-tuning.shを参照)

```
$ ali-to-post ¥  
ark:pdf_ali(PDF-IDを用いたアライメント) ¥  
ark:pdf_post(出力先)
```

# バックプロパゲーション

実行例 (01\_3synop\_fine-tuning.shを参照)

```
$ nnet-train-frmsbuff ¥  
  --learn-rate=0.008 (学習率) ¥  
  --momentum=1.0e-05 (モーメントム) ¥  
  --minibatch-size=256 (ミニバッチサイズ) ¥  
  --randomizer-size=32768 (シャッフルのバッファサイズ) ¥  
  --feature-transform=feature_transform (入力特徴量の変換) ¥  
  --randomizer-seed=777 (seed) ¥  
  "ark:copy-feats scp:train.scp ark:- |" (特徴量) ¥  
  ark:pdf_post (DNN事後確率) ¥  
  in.nnet (DNN) ¥  
  out.nnet (出力先)
```

# 学習率の扱いについて

steps/nnet/train\_scheduler.sh で用いられている戦略

- クロスバリデーションセットの前エポックと現在のクロスエントロピーの値の相対改善率が規定値を下回る場合学習率を半減
- クロスバリデーションセットの相対改善率が規定値を下回ったら学習終了

### iii. 評価セットの認識

- ラティスあり: `latgen-faster-mapped`
  - 対応するラッパースクリプト: `steps/nnet/decode.sh`
- ラティスなし: `decode-faster-mapped`
  - 対応するラッパースクリプト: `local/nnet/decode_dnn_nolats.sh`  
(本ラッパースクリプトは公開版Kaldi内には無く、  
今実習のために作成したもの)

# ラティスを作成する認識(DNN-HMM)

```
$ nnet-forward ¥  
--no-softmax=true ¥  
--feature-transform=final.feature_transform(特徴量変換) ¥  
--class-frame-counts=prior_counts(PDFのフレームカウント) ¥  
final.nnet(モデル) ¥  
"ark,s,cs:copy-feats scp:feats.scp ark:- |"(特徴量) ¥  
ark:-(標準出力) | ¥  
latgen-faster-mapped ¥  
--beam=13.0(ビーム) ¥  
--lattice-beam=8.0(ラティスビーム) ¥  
--acoustic-scale=0.0909(音響尤度の比率) ¥  
--word-symbol-table=words.txt(出力シンボル) ¥  
final.mdl(モデル) ¥  
HCLG.fst ark:-(FST) ¥  
"ark:|gzip -c > lat.1.gz"(出力先)
```

# ラティスを作成しない認識(DNN-HMM)

```
$ nnet-forward ¥  
  --no-softmax=true ¥  
  --feature-transform=final.feature_transform (特徴量変換) ¥  
  --class-frame-counts=prior_counts (PDFのフレームカウント) ¥  
  final.nnet (モデル) ¥  
  “ark,s,cs:copy-feats scp:feats.scp ark:- |” (特徴量) ¥  
  ark:- (標準出力) | ¥  
  decode-faster-mapped ¥  
  --beam=13.0 (ビーム) ¥  
  --acoustic-scale=0.0909 (音響尤度の比率) ¥  
  --word-symbol-table=words.txt (出力シンボル) ¥  
  final.mdl (モデル) ¥  
  HCLG.fst ark:- (FST) ¥  
  “ark:|gzip -c > words.1.gz” (出力先)
```



# DBN/Nnetのバイナリ・テキスト変換

```
$ nnet-copy --binary=false foo.(dbn/nnet) destfile
```

```
[kalditest@suzukake s5_demo1]$ final_nnet=08_dnn/exp/dnn5b_pretrain-dbn_dnn/final.nnet
[kalditest@suzukake s5_demo1]$ nnet-copy --binary=false $final_nnet - | head -n 4
nnet-copy --binary=false 08_dnn/exp/dnn5b_pretrain-dbn_dnn/final.nnet -
<Nnet>
<AffineTransform> 256 440
<LearnRateCoef> 1 <BiasLearnRateCoef> 1 <MaxNorm> 0 [
  -0.1844334 -0.4705751 -0.2005902 0.2116556 0.6403909 -0.2382788 0.4106917 0.156999 0.
5 0.322658 -0.1328638 0.001161144 0.2475637 -0.194409 0.07545146 0.1103025 0.3214659 -0
31631 0.05354135 -0.1910693 0.1584137 0.1259145 -0.08965535 -0.01454324 0.04012248 -0.0
.1242767 0.1292929 0.5704796 -0.1947335 0.3559495 0.1815075 0.04229722 0.07666813 -0.01
353781 0.1875764 -0.07298911 0.001389893 0.0709499 0.2371386 -0.3631808 0.188143 0.0690
```

# DNNファイルの概要表示

```
$ nnet-info foo.nnet
```

```
[kalditest@suzukake s5_demo1]$ nnet-info 08_dnn/exp/dnn5b_pretrain-dbn_dnn/final.nnet | head -n 8
nnet-info 08_dnn/exp/dnn5b_pretrain-dbn_dnn/final.nnet
LOG (nnet-info:main():nnet-info.cc:57) Printed info about 08_dnn/exp/dnn5b_pretrain-dbn_dnn/final.
nnet
num-components 8
input-dim 440
output-dim 493
number-of-parameters 0.371181 millions
component 1 : <AffineTransform>, input-dim 440, output-dim 256,
  linearity ( min -1.3777, max 1.95341, mean -0.00120677, variance 0.0187669, skewness 0.0543704,
kurtosis 1.80027 )
  bias ( min -2.67613, max 1.336, mean -0.540819, variance 0.449719, skewness 0.0774225, kurtosis
0.348014 )
component 2 : <Sigmoid>, input-dim 256, output-dim 256,
```

# 系列識別学習を用いたDNNの再学習

- 系列識別基準を用いたDNNパラメタの推定
  - 計算量は多いが、フレームベースのファインチューニングから更に認識率向上が期待できる

## 実行方法

```
$ cd ../09_dnn_s  
$ ./run.sh
```

\*ラティスの生成が必要

steps/nnet/make\_denlats.sh

\*学習スクリプト本体(MPE基準)

steps/nnet/train\_mpe.sh

# Appendix

# バイナリ・テキスト形式の変換

- Kaldiでは多くのデータファイルはデフォルトではバイナリ形式で扱われる
- 入力ファイルがバイナリ形式かテキスト形式かは自動判定される
- Wspecifierを用いた出力の際にオプション(t)で出力形式を指定できる

# バイナリ→テキスト変換コマンドリスト

- 特徴量ファイル(ark形式):  
`copy-feats ark:foo.ark ark,t:destfile`
- 特徴量ファイル(scpで読み込み元ファイルを指定):  
`copy-feats scp:foo.scp ark,t:destfile`
- ラティスファイル:  
`lattice-copy ark:foo.lat ark,t:destfile`
- ラティスファイル(圧縮形式):  
`lattice-copy "ark:gunzip -c foo.lat.gz |" ark,t:destfile`
- 単語リスト:  
`copy-int-vector ark:words.bin ark,t:destfile`
- GMM-HMMモデルファイル:  
`gmm-copy --binary=false hmm.mdl destfile`

# バイナリ→テキスト変換コマンドリスト

- GMM-HMM統計量ファイル:  
`gmm-sum-accs --binary=false destfile hmm.acc`
- Diagonal GMMモデルファイル:  
`gmm-global-copy --binary=false gmm.mdl destfile`
- Diagonal GMM統計量ファイル:  
`gmm-global-sum-accs --binary=false destfile foo.acc`
- DNNファイル(dbnまたはnnetファイル):  
`nnet-copy --binary=false foo.(dbn/nnet) destfile`
- アライメントファイル:  
`ali-to-pdf final.mdl "ark:gunzip -c ali.*.gz |" ark,t:destfile`

# バイナリ→テキスト変換コマンドリスト

- 決定木ファイル:

`copy-tree --binary=false tree destfile`

- 整数形式(テキスト)の認識結果を単語に置換:

`utils/int2sym.pl -f 2: word.txt int.hyp`



# ファイル情報の表示

- GMM-HMMファイル:  
gmm-info foo.mdl または、hmm-info foo.mdl
- GMMファイル:  
gmm-global-info foo.dubm
- DNNファイル:  
nnet-info foo.nnet
- 音響モデルの情報:  
am-info foo.mdl
- 特徴量の次元数:  
feat-to-dim scp:feats.scp
- 決定木ファイル:  
tree-info tree

# その他確認用コマンド

- アライメントの確認:

```
show-alignments phons.txt foo.mdl ark:ali.*
```

- 状態決定木ファイルの質問リストおよびリーフノードのPDF作成:

```
draw-tree phones.txt tree | dot -Tps -Gsize=8,10.5 | ¥  
ps2pdf - ~/tree.pdf
```