Why Deep Neural Network Works for Speech Recognition?

Hui Jiang

Department of Electrical Engineering and Computer Science

Lassonde School of Engineering, York University, CANADA

Joint work with Y. Bao, J. Pan, O. Abdel-Hamid

Outline

LASSONDE SCHOOL OF ENGINEERING | YORK UNIVERSITÉ UNIVERSITY

- Introduction
  - Automatic Speech Recognition (ASR)
  - Deep Neural Network (DNN)
- DNN/HMM for Speech
- Bottleneck Features
- Incoherent Training
- Conclusions

Introduction: ASR History

- ASR formulation:
  - GMM/HMM + n-gram + Viterbi search
- Technical advances (incremental) over past 10 years:
  - Adaptation (speaker/environment): 5% *rel.* gain
  - Discriminative Training: 5-10% *rel.* gain
  - Feature normalization: 5% *rel.* gain
  - ROVER: 5% *rel.* gain
- More and more data ➔ better and better accuracy
  - read speech (>90%), telephony speech (>70%)
  - meeting/voicemail recording (<60%)

Acoustic Modeling: Optimization

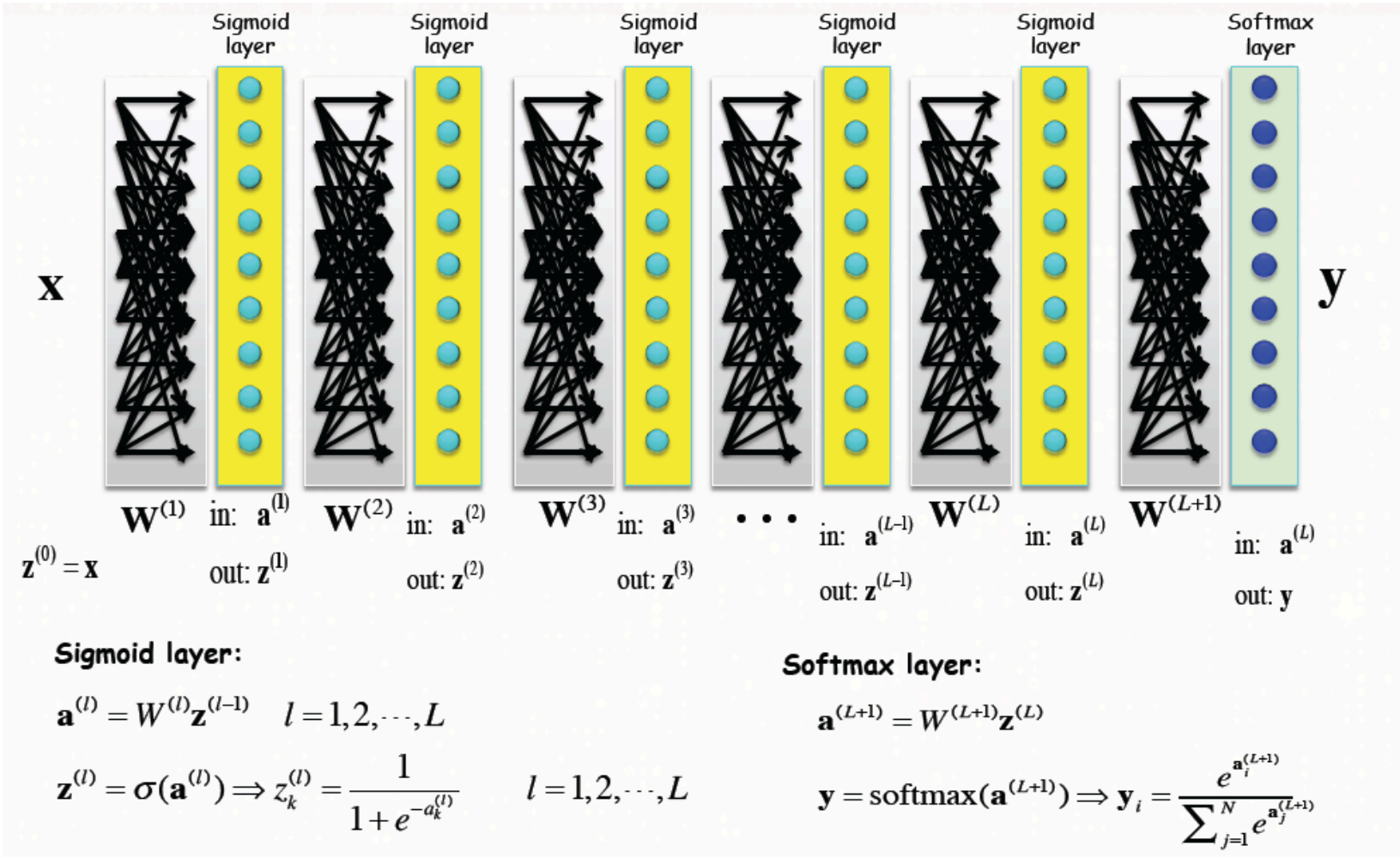
- Acoustic modeling → large-scale optimization
  - 2000+ hour data → GMMs/HMM

    billions of samples → 10+ million free parameters
- Training Methods
  - Maximum Likelihood Estimation (MLE)
  - Discriminative Training (DT)
- Engineering Issues
  - Efficiency: feasible with 100-1000 of CPUs
  - Reliability: robust estimation of all parameters

Neural Network for ASR

- **1990s: MLP for ASR** *(Bourlard and Morgan, 1994)*
  - **NN/HMM hybrid model (worse than GMM/HMM)**
- **2000s: TANDEM** *(Hermansky, Ellis, et al., 2000)*
  - **Use MLP as Feature Extraction (5-10% rel. gain)**
- **2006: DNN for small tasks** *(Hinton et al., 2006)*
  - **RBM-based pre-training for DNN**
- **2010: DNN for small-scale ASR** *(Mohamed, Yu, et al. 2010)*
- **2011: DNN for large-scale ASR**
  - **Over 30% rel. gain in Switchboard** *(Seide et al., 2011)*

Deep Neural Network (DNN)



Sigmoid layer:

$$\mathbf{a}^{(l)} = W^{(l)}\mathbf{z}^{(l-1)} \qquad l = 1, 2, \cdots, L$$

$$\mathbf{z}^{(l)} = \sigma(\mathbf{a}^{(l)}) \Rightarrow z_k^{(l)} = \frac{1}{1+e^{-a_k^{(l)}}} \qquad l = 1, 2, \cdots, L$$

Softmax layer:

$$\mathbf{a}^{(L+1)} = W^{(L+1)}\mathbf{z}^{(L)}$$

$$\mathbf{y} = \text{softmax}(\mathbf{a}^{(L+1)}) \Rightarrow \mathbf{y}_i = \frac{e^{\mathbf{a}_i^{(L+1)}}}{\sum_{j=1}^{N} e^{\mathbf{a}_j^{(L+1)}}}$$

DNN Training (I)

Given a training set $X=\{\mathbf{x}_t, l_t,\ t=1,2\cdots T\}$, optimize the objective function:

$$Q(\mathbf{W})=\sum_{t=1}^{T}Q_t(\mathbf{W})=\sum_{t=1}^{T}\sum_{i=1}^{N}[-\delta(l_t-i)\cdot\ln \mathbf{y}_i(\mathbf{x}_t,\mathbf{W})]$$

Define error signals in each layer $e_{tk}^{(l)}=\frac{\partial}{\partial a_k^{(l)}}Q_t(W)=\frac{-1}{y_{l_t}(\mathbf{x}_t,\mathbf{W})}\frac{\partial y_{l_t}(\mathbf{x}_t,\mathbf{W})}{\partial a_k^{(l)}}$

Softmax layer l=L+1

$$e_{tk}^{(L+1)}=\frac{-1}{y_{l_t}(\mathbf{x}_t,\mathbf{W})}\frac{\partial y_{l_t}(\mathbf{x}_t,\mathbf{W})}{\partial a_k^{(L+1)}}=y_k-\delta(l_t-k)$$

Sigmoid layer l=1,2,…,L

$$e_{tk}^{(l)}=\frac{\partial Q_t(W)}{\partial a_k^{(l)}}=\sum_{j=1}^{N}\frac{\partial Q_t(W)}{\partial a_j^{(l+1)}}\frac{\partial a_j^{(l+1)}}{\partial a_k^{(l)}}=\sum_{j=1}^{N}e_{tj}^{(l+1)}\frac{\partial a_j^{(l+1)}}{\partial a_k^{(l)}}=\sum_{j=1}^{N}e_{tj}^{(l+1)}\cdot z_k^{(l)}\cdot(1-z_k^{(l)})\cdot W_{kj}^{(l+1)}$$

$$=z_k^{(l)}\cdot(1-z_k^{(l)})\cdot\sum_{j=1}^{N}e_{tj}^{(l+1)}W_{jk}^{(l+1)}$$

DNN Training (II)

Given a training set $\mathrm{X}=\{\mathbf{x}_t, l_t,\ t=1,2\cdots T\}$

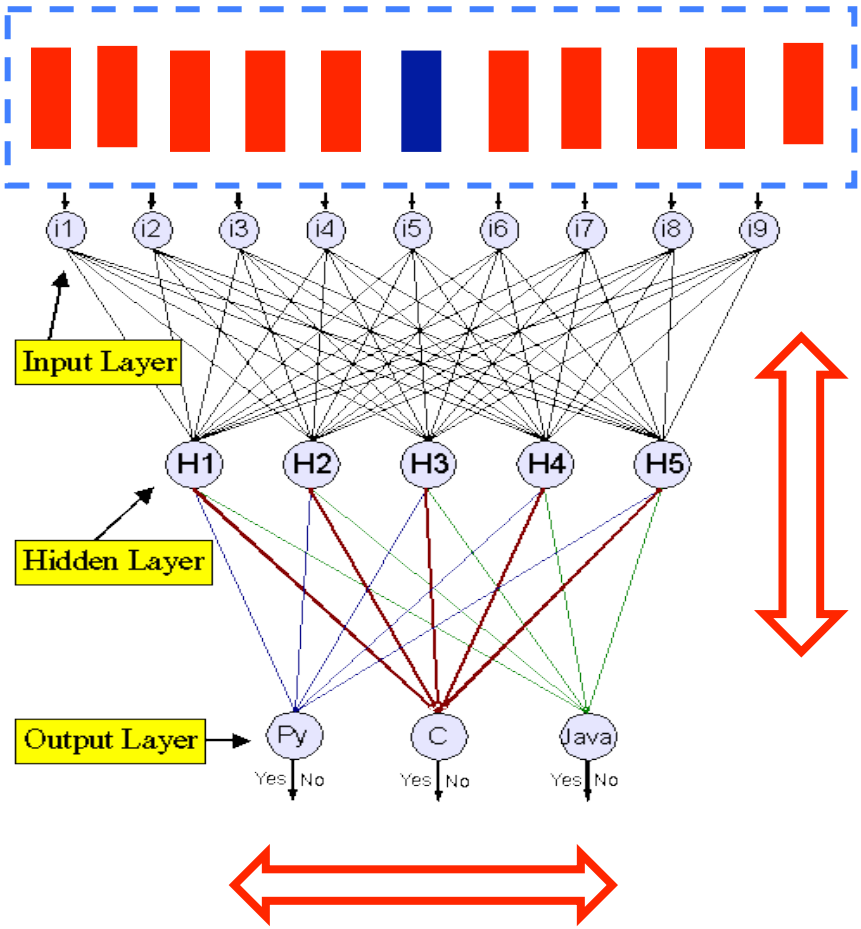
$$Q(\mathbf{W}) = -\sum_{t=1}^{T} Q_t(\mathbf{W}) = -\sum_{t=1}^{T}\sum_{i=1}^{N} \delta(l_t - i)\cdot \ln \mathbf{y}_i(\mathbf{x}_t, \mathbf{W})$$

$$\frac{\partial}{\partial \mathbf{w}_k^{(l)}} Q_t(\mathbf{W}) = \frac{\partial Q_t(\mathbf{W})}{\partial a_k^{(l)}} \frac{\partial a_k^{(l)}}{\partial \mathbf{w}_k^{(l)}} = e_{tk}^{(l)} \cdot \mathbf{z}_t^{(l-1)}$$

Use stochastic gradient Descent (SGD) to update weight vectors:

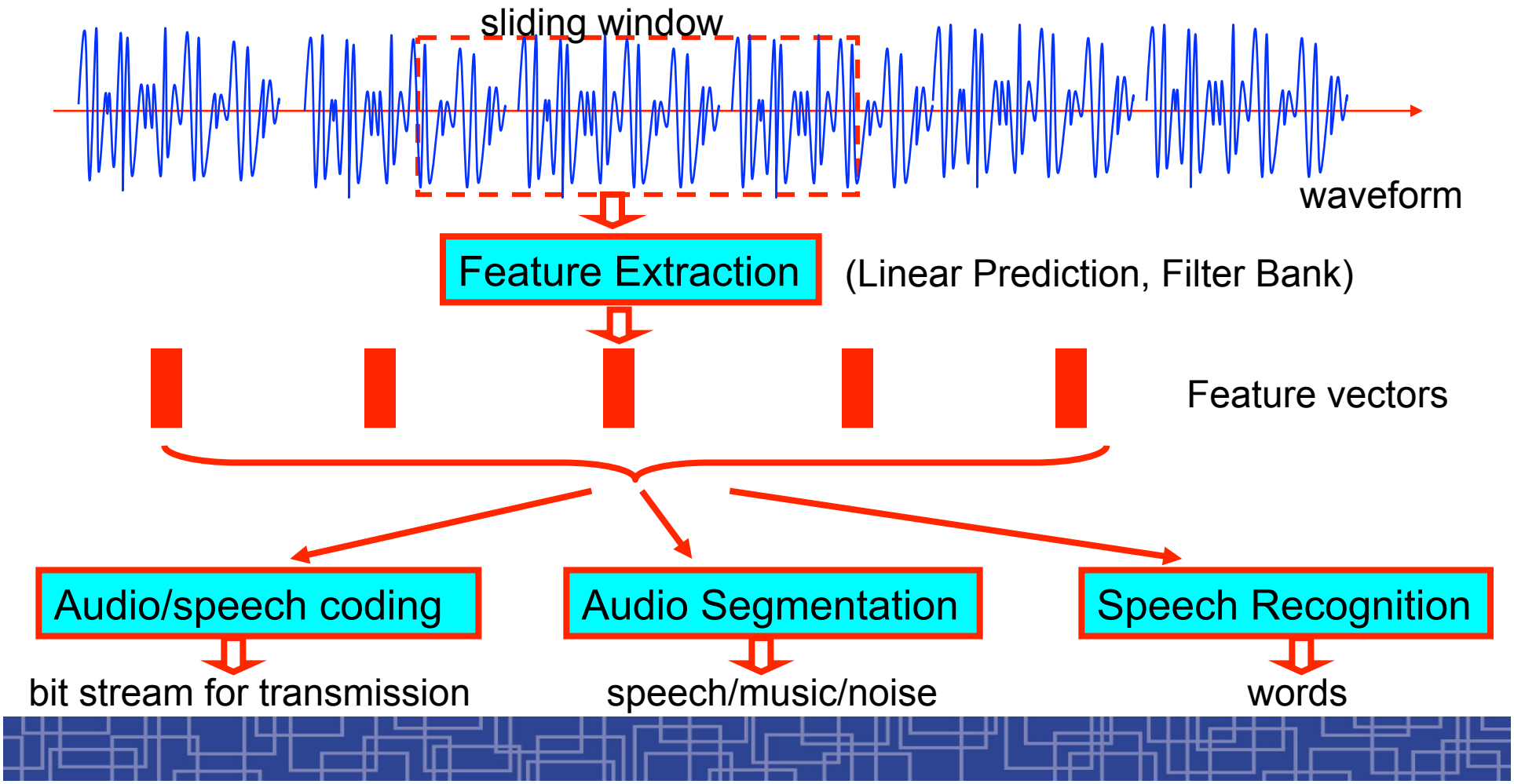
$$\mathbf{w}_k^{(l)} \leftarrow \mathbf{w}_k^{(l)} - \eta \frac{\partial}{\partial \mathbf{w}_k^{(l)}} Q_t(\mathbf{W}) = \mathbf{w}_k^{(l)} - \eta \cdot e_{tk}^{(l)} \cdot \mathbf{z}_t^{(l-1)}$$

NN for ASR: old and new

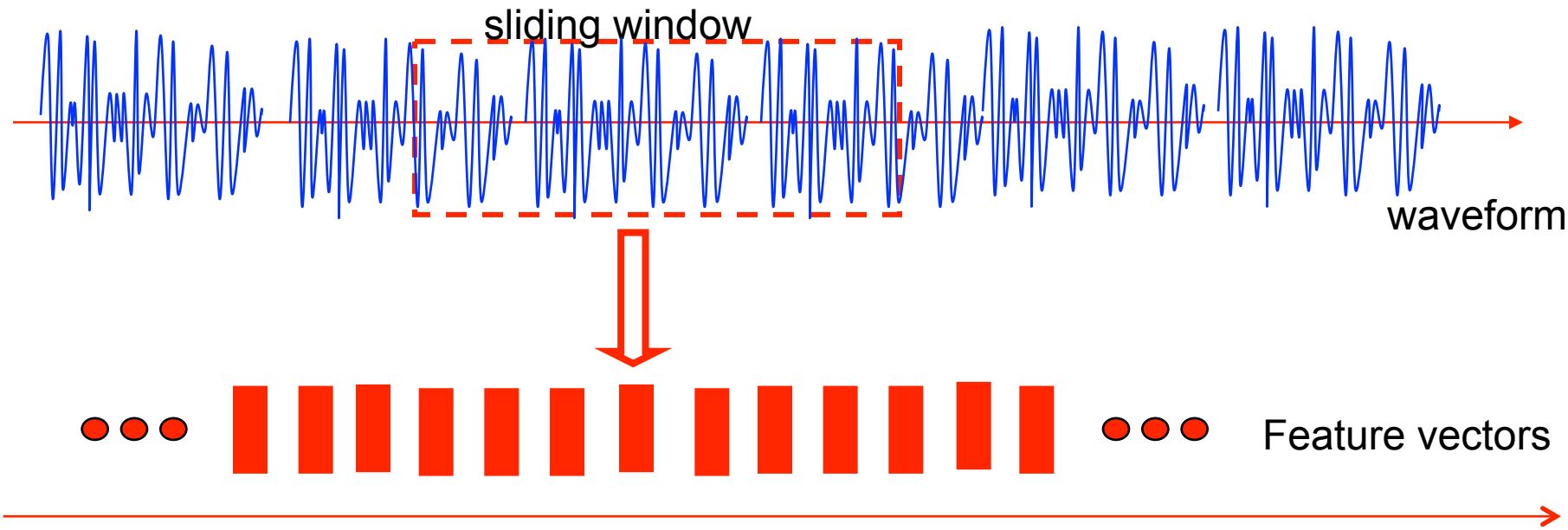


- Deeper network
  - more hidden layers
  - (1 → 6-7 layers)
- Wider network
  - More hidden nodes
  - More output nodes
  - (100 → 5-10 K)
- More data
  - 10-20 hours → 300 to 10k+ hours of training data

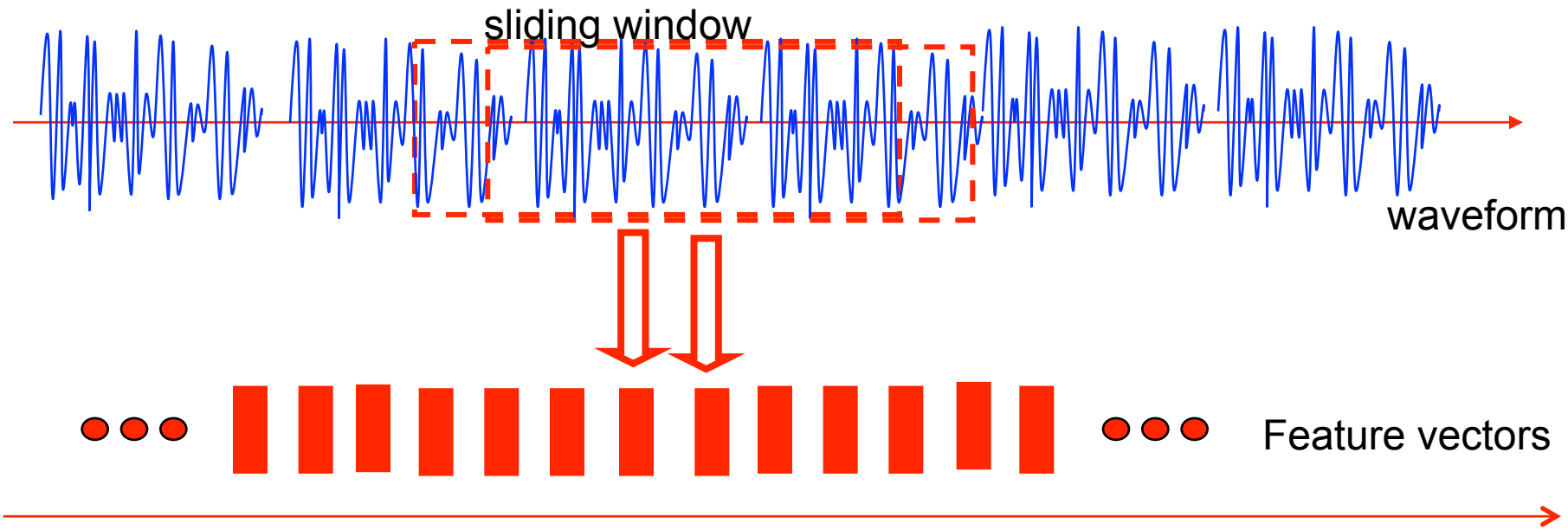
ASR Frontend



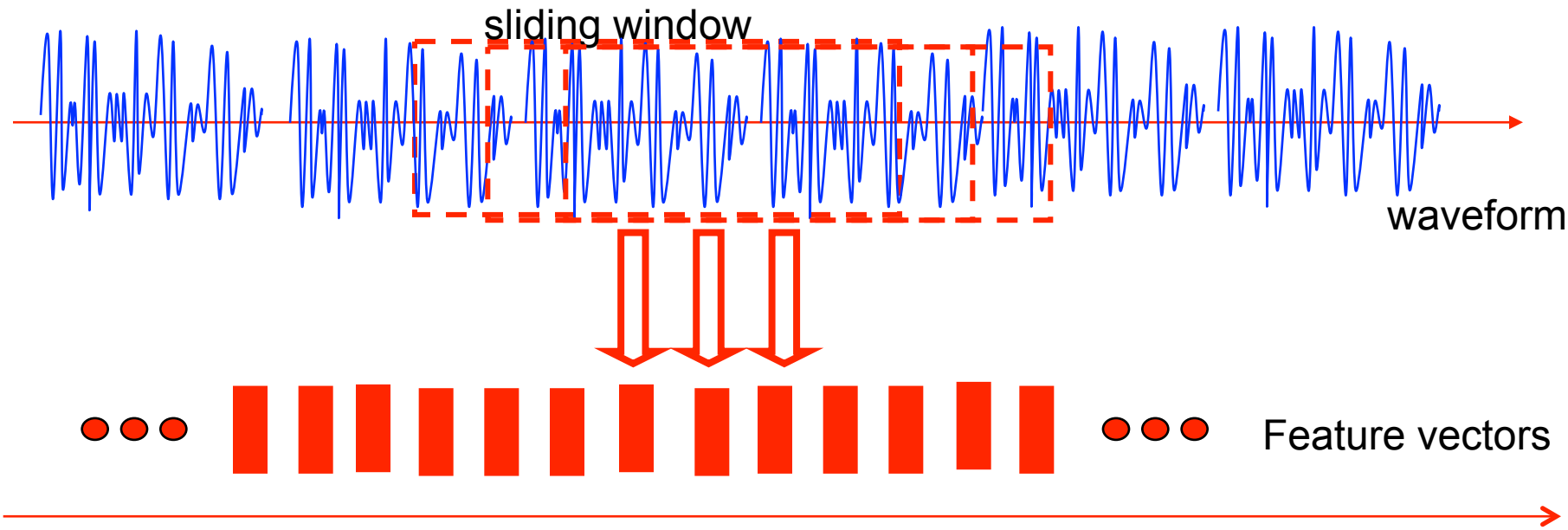
Short-time Analysis



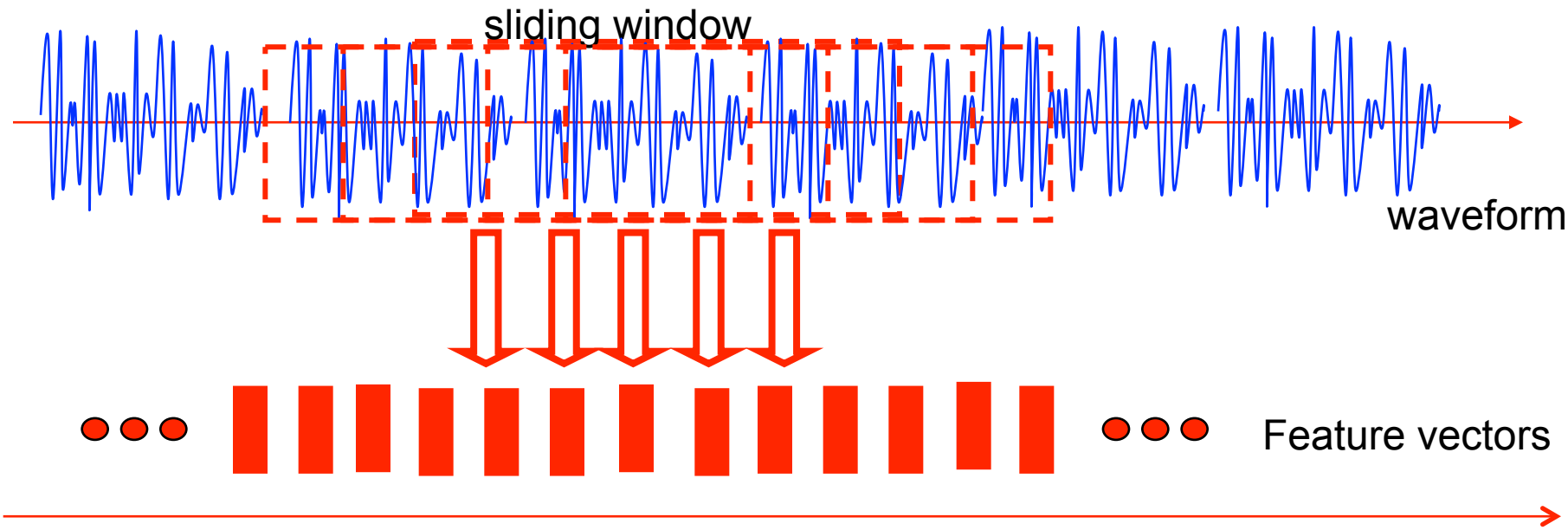
Short-time Analysis



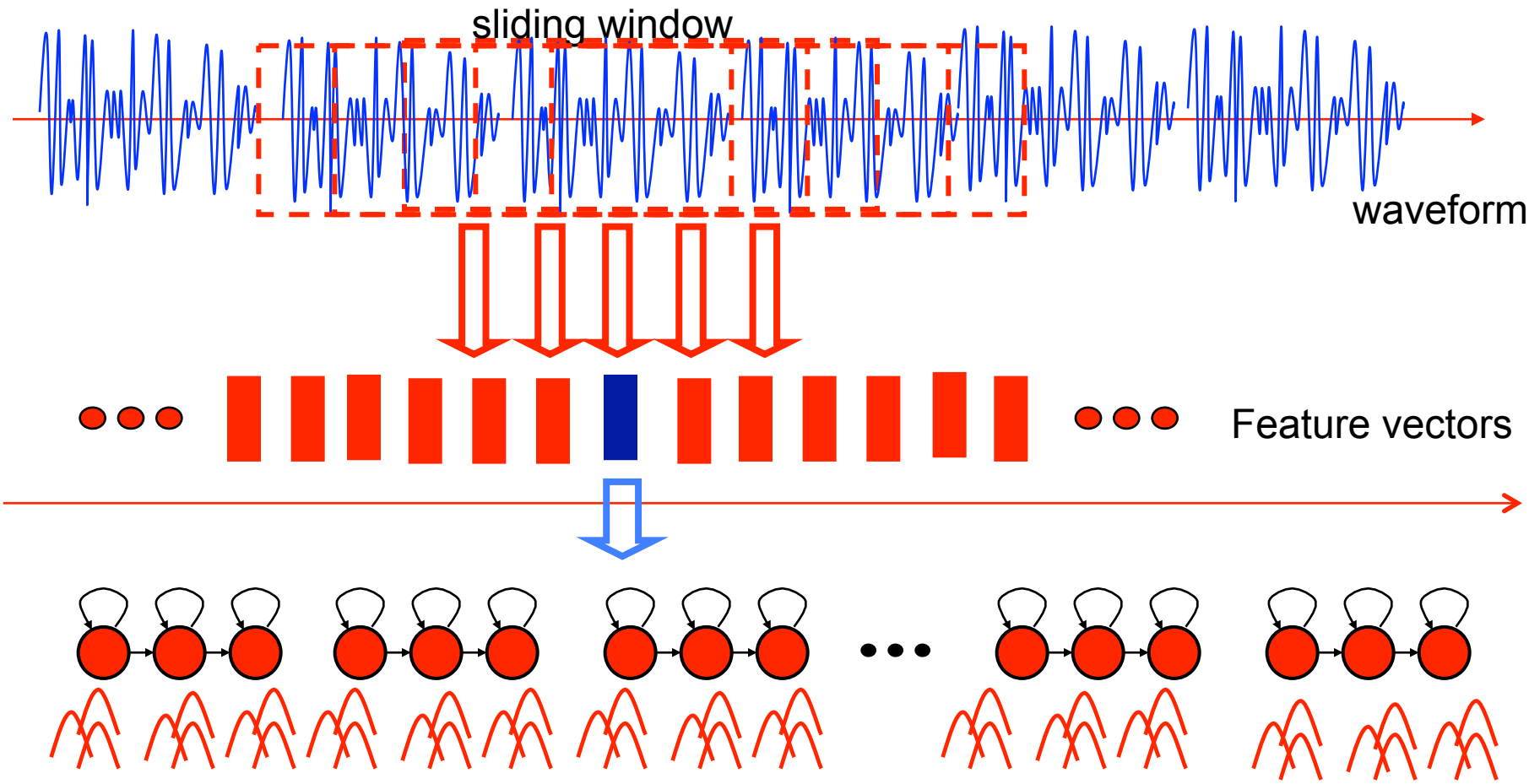
Short-time Analysis



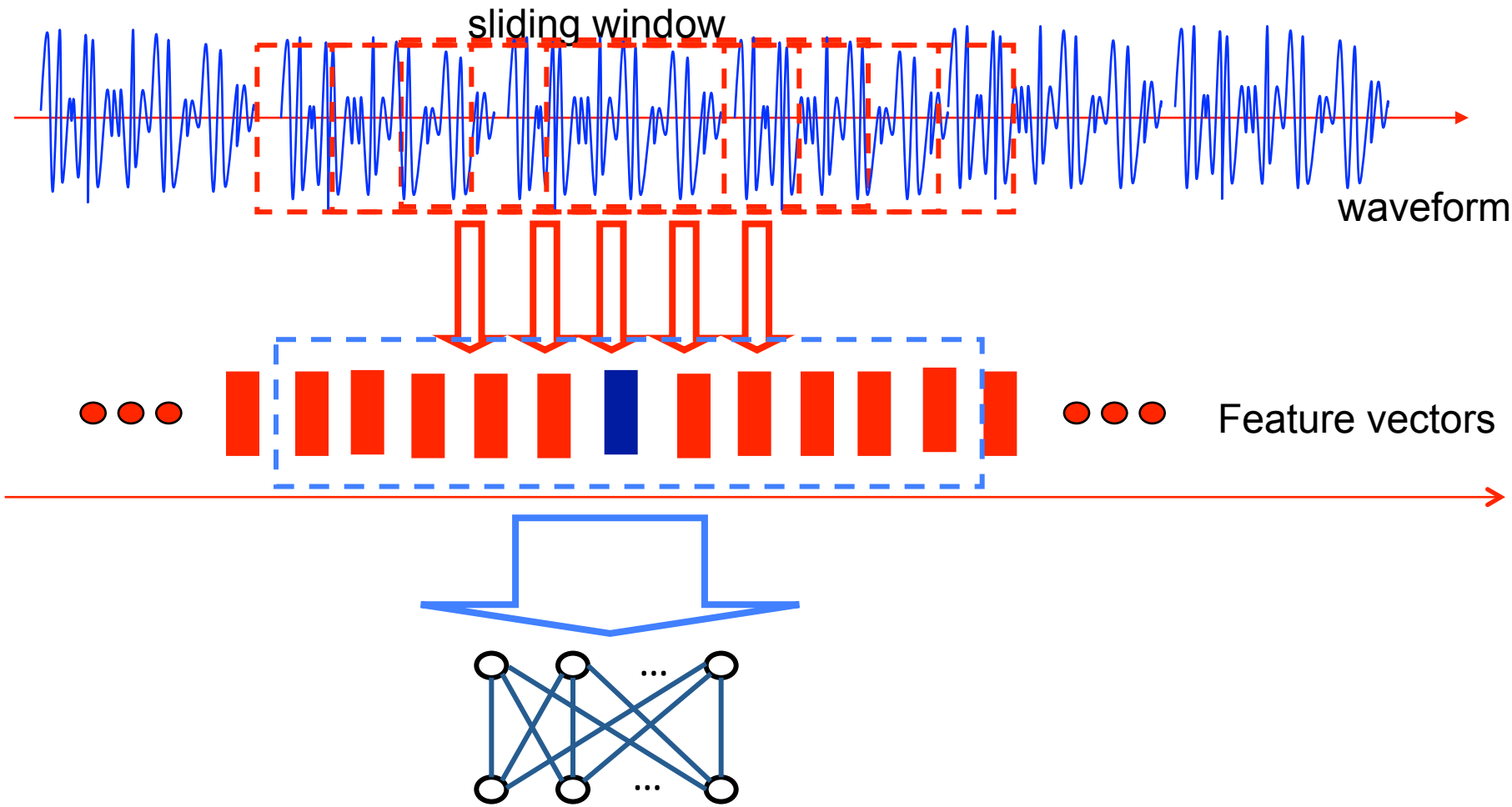
Short-time Analysis



ASR Frontend: GMM/HMM

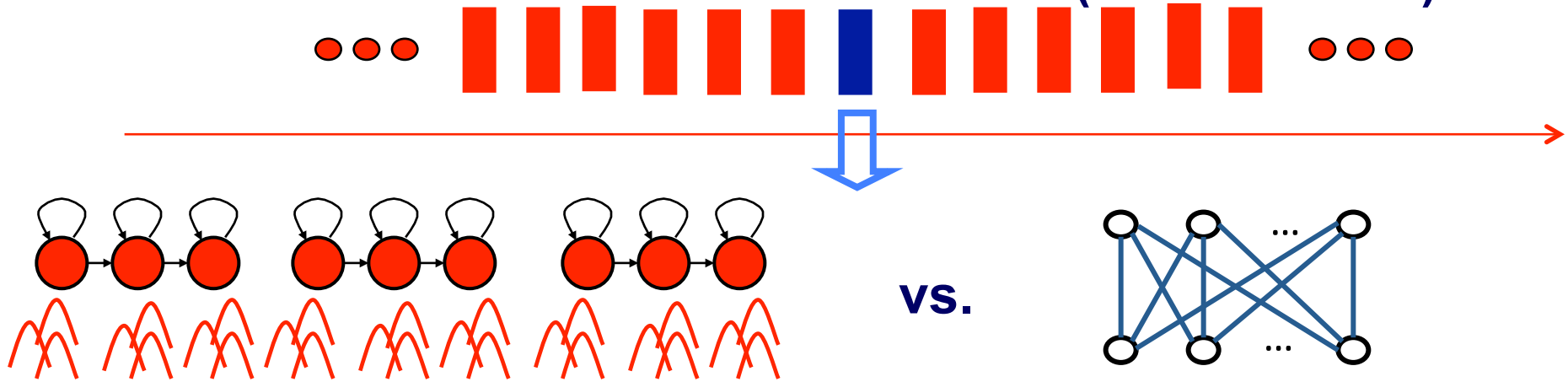


ASR Frontend: NN/HMM



GMMs/HMM vs. DNN/HMM

- Different acoustic models
  - GMMs vs. DNN
- Different feature vectors
  - 1 frame vs. concatenated frames (11-15 frames)



Experiment (I): GMMs/HMM vs. DNN/HMM

- In-house 70-hour Mandarin ASR task;
- GMM: 4000 tied HMM states, 30 Gaussians per state
- DNN: pre-trained; 1024 nodes per layer; 1-6 hidden layers

Numbers in word error rates (%)
NN-1: 1 hidden layer; DNN-6: 6 hidden layers
MPE-GMM: discriminatively trained GMM/HMM

Context window	NN-1	DNN-6	MPE-GMM
1	18.0	17.0	16.7
Context window	3	5	7
DNN-6	14.2	13.7	13.5
Context window	9	11	13
DNN-6	13.5	13.4	13.6

Experiment (II): GMMs/HMM vs. DNN/HMM

- 300-hour Switchboard task, Hub5e01 test set
- GMM: 8991 tied HMM states, 40 Gaussian per state
- DNN: pre-trained; 2048 nodes per layer; 1-5 hidden layers

Word error rates (WER) in Hub01 test set (%)
NN-1: 1 hidden layer; DNN-3/5: 3/5 hidden layers
MPE-GMM: discriminatively trained GMM/HMM

context window	MPE GMM	NN-1	DNN-3	DNN-5
1	32.8%	35.4%	34.8%	33.1%
11	n/a	31.4%	25.6%	23.7%

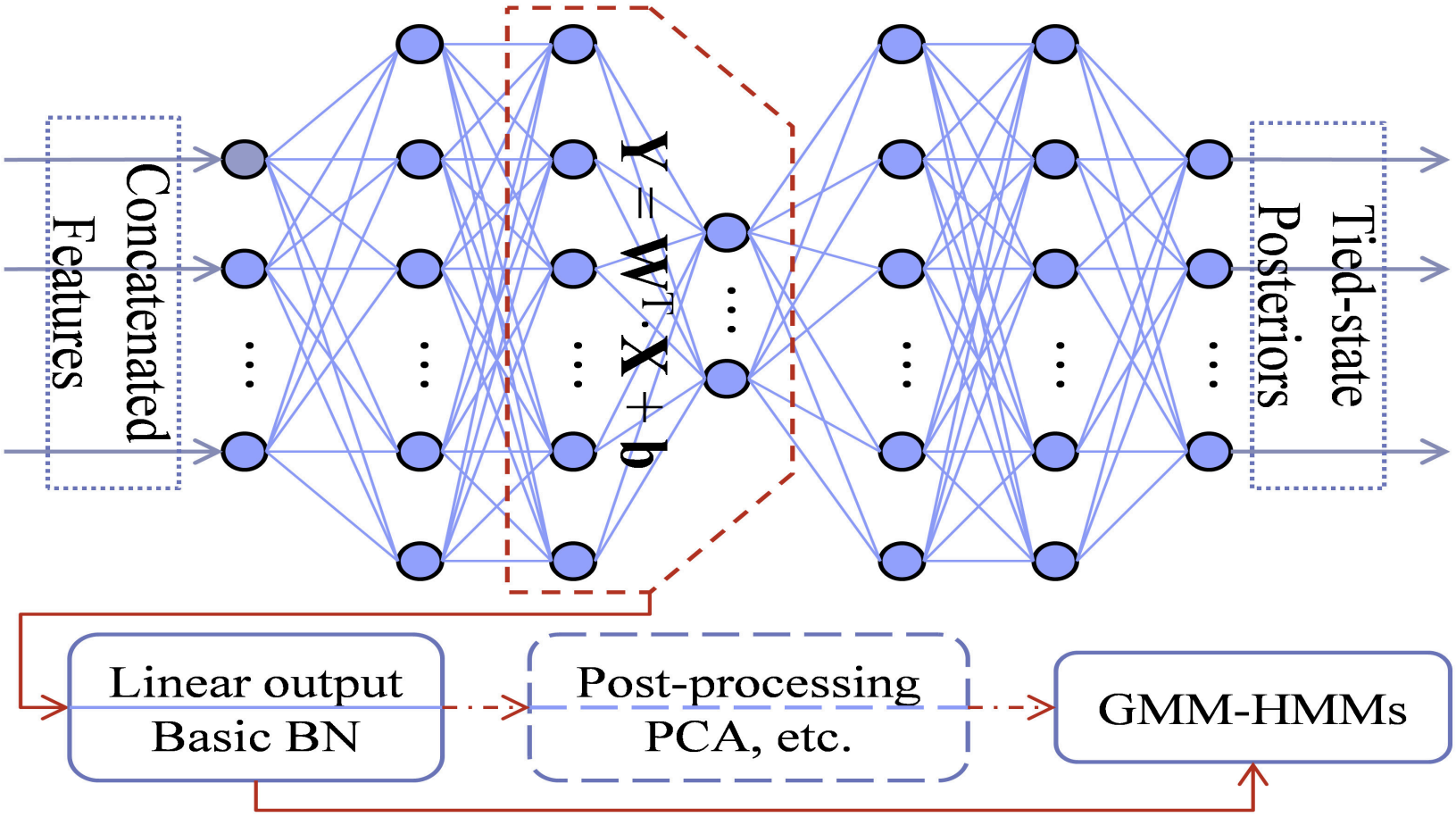
Conclusions (I)

- The gain of DNN/HMM hybrid is almost entirely attributed to the concatenated frames.
  - The concatenated features contain almost all additional information resulting in the gain.
  - But they are highly correlated.
- DNN is powerful to leverage highly correlated features.

What's next

- How about GMM/HMM?
- Hard to explore highly correlated features in GMMs.
  - Requires dimensional reduction for de-correlation.
- Linear dimensional reduction (PCA, LDA, KLT, …)
  - Failed to compete with DNN.
- Nonlinear dimensional reduction
  - Using NN/DNN *(Hinton et al.)*, a.k.a. bottleneck features
  - Manifold learning, LLE, MDS, SNE, …?

Bottleneck (BN) Feature



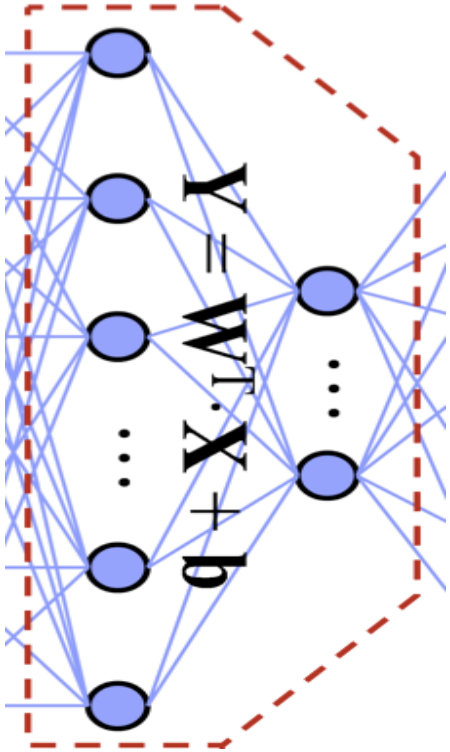
Experiments: BN vs. DNN

300-hour English Switchboard Task (WER in %)
MLE: maximum likelihood estimation; MPE: discriminative training
ReFA: realigned class labels; DT: sequence training of DNNs
DT+BN: BN features extracted from sequence-trained BN networks

Acoustic models (Features)	Hub5e01		Hub5e00	
	MLE	MPE	MLE	MPE
GMM-HMMs (PLP)	35.4%	32.8%	28.7%	24.7%
GMM-HMMs (BN)	26.0%	23.2%	17.9%	15.9%
DNN (11 PLPs)	23.7%		16.7%	
DNN (ReFA+DT)	--		14.0%	
GMM-HMMs (DT+BN)	--		16.6%	14.6%

Incoherent Training

- Bottleneck (BN) works but:
  - BN hurts DNN performance a little
  - Increasing BN ➔ correlation up
- Can we do better?
- The Idea: embedding de-correlation into back-propagation of DNN training.
  - De-correlation by constraining columns of weight matrix *W*
  - How to constrain?



Incoherent Training

- **Define coherence of DNN weight matrix *W* as:**

$$G_W = \max_{i,j} g_{ij} = \max_{i,j} \frac{\left| w_i \cdot w_j \right|}{\left\| w_i \right\| \left\| w_j \right\|}$$

- **A matrix with smaller coherence indicates all of its column vectors are less similar.**
- **Approximate coherence using soft-max:**

$$G_W = \log \left(\frac{1}{M} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \exp\{\beta \cdot g_{ij}\} \right)^{\frac{1}{\beta}}$$

Incoherent Training

- All DNN weight matrices are optimized by minimizing a regularized objective function:

$$F^{(new)} = F^{(old)} + \alpha \cdot \max_{W} G_W$$

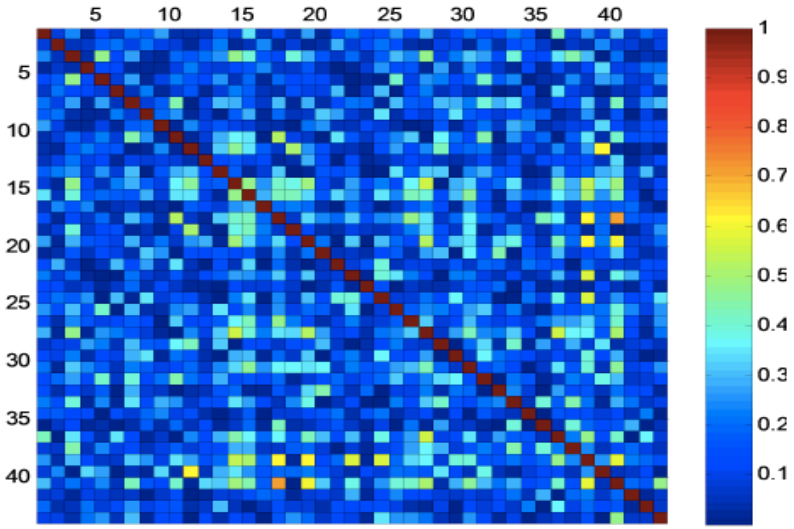
- Derivatives of coherence:

$$\frac{\partial G_W}{\partial w_k} = \sum_{j=1}^{N} \gamma_{kj} g_{kj} \left[\frac{\mathbf{w}_j}{\mathbf{w}_k \cdot \mathbf{w}_j} - \frac{\mathbf{w}_k}{\mathbf{w}_k \cdot \mathbf{w}_k} \right]$$

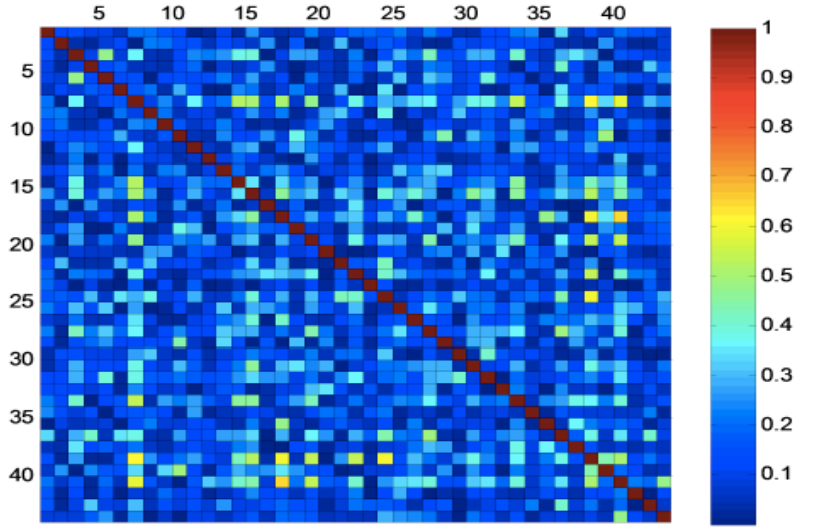
- Back-propagation is still applicable…

Incoherent Training: De-correlation

Applying incoherent training to one weight matrix in BN



(a) Baseline BN



(b) Weight-matrix Incoherent BN

Incoherent Training: Data-driven

- If only applying to one weight matrix *W*: $Y = W^T X + b$
- Covariance matrix of Y: $C_Y = W^T C_X W$
- Directly measure correlation coefficients based on the above covariance matrix:

$$G_W = \max_{i \neq j} g_{ij}$$

with $$g_{ij} = \frac{\left|(\mathbf{w}_i)^\top \mathbf{C_X} \mathbf{w}_j\right|}{\sqrt{(\mathbf{w}_i)^\top \mathbf{C_X} \mathbf{w}_i} \cdot \sqrt{(\mathbf{w}_j)^\top \mathbf{C_X} \mathbf{w}_j}}$$

Cx is estimate from one mini-batch each time

Incoherent Training: Data-driven

- After *soft-max*, Derivatives can be computed as:

where

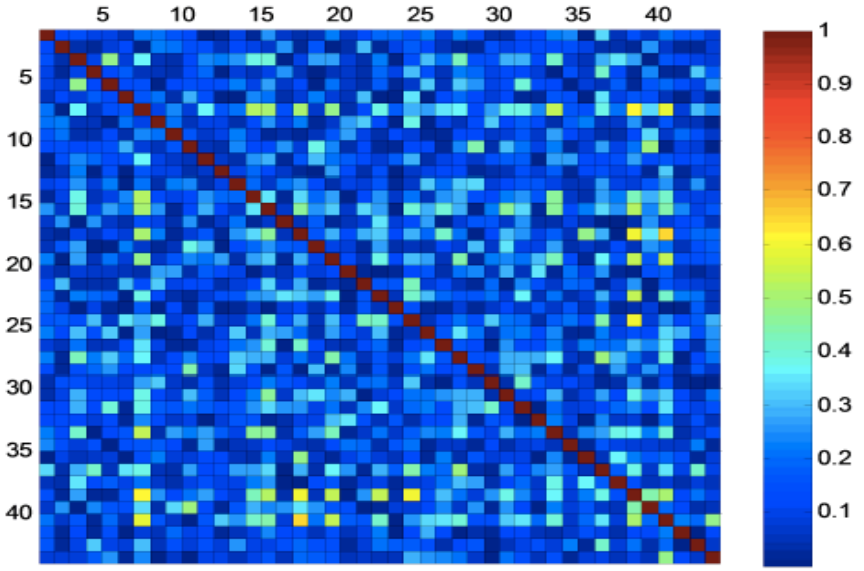
$$\frac{\partial G_W}{\partial w_k} = \frac{\sum_{j \neq k}^{N} \left[\exp\{\beta \cdot g_{kj}\} \cdot \frac{\partial g_{kj}}{\partial \mathbf{w}_k} \right]}{\sum_{i=1}^{N} \sum_{j=i+1}^{N} \exp\{\beta \cdot g_{ij}\}}$$

$$\frac{\partial g_{kj}}{\partial \mathbf{w}_k} = g_{kj} \left[\frac{\mathbf{C_X w}_j}{(\mathbf{w}_k)^\top \mathbf{C_X w}_j} - \frac{\mathbf{C_X w}_k}{(\mathbf{w}_k)^\top \mathbf{C_X w}_k} \right]$$

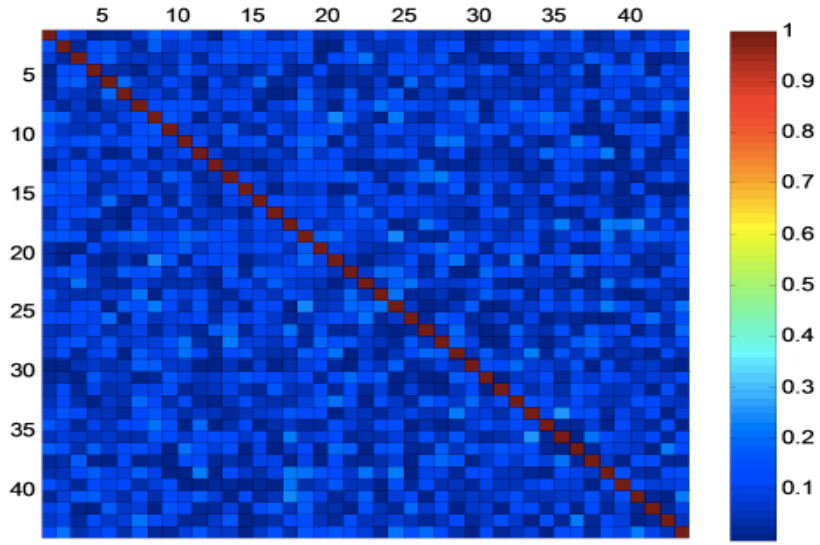
- Back-propagation still applies except Cx is computed for each mini-batch

Incoherent Training: Data-driven De-correlation

When applying Incoherent Training to one weight matrix



(b) Weight-matrix Incoherent BN



(c) Mini-batch-data Incoherent BN

Experiment: Incoherent Training

300-hour English Switchboard Task, WER (%) on Hub5e01
DNN: 6 hidden layers of 2048 nodes
BN: extracted from 5 hidden layers of bottleneck DNN

BN Feature / models	MLE	MPE
DNN	23.7%	
Baseline BN	26.0%	23.2%
Weight-Matrix Incoherent BN	25.7%	--
Mini-batch-data Incoherent BN	25.6%	22.8%

Conclusions (II)

- Possible to compete with DNN under the traditional GMMs/HMM framework.
- Promising to use bottleneck features learned from the proposed incoherent training.
- Benefits over DNN/HMM:
  - Slightly better performance
  - Enjoy other ASR techniques (adaptation, …)
  - Faster training process
  - Faster decoding process

Future Work

- Apply incoherent training to general DNN learning.
- Consider other nonlinear dimensional reduction methods for concatenated features.

Other Ongoing DNN Projects

- Convolutional neural network (CNN) for ASR
- Rapid adaptation of DNNs
- Parallel training of DNNs