Why DNN Works for Speech and How to Make it More Efficient?

Hui Jiang

Department of Electrical Engineering and Computer Science

Lassonde School of Engineering, York University, CANADA

Joint work with Y. Bao*, J. Pan*, P. Zhou*, S. Zhang*, O. Abdel-Hamid

**** University of Science and Technology of China, Hefei, CHINA***

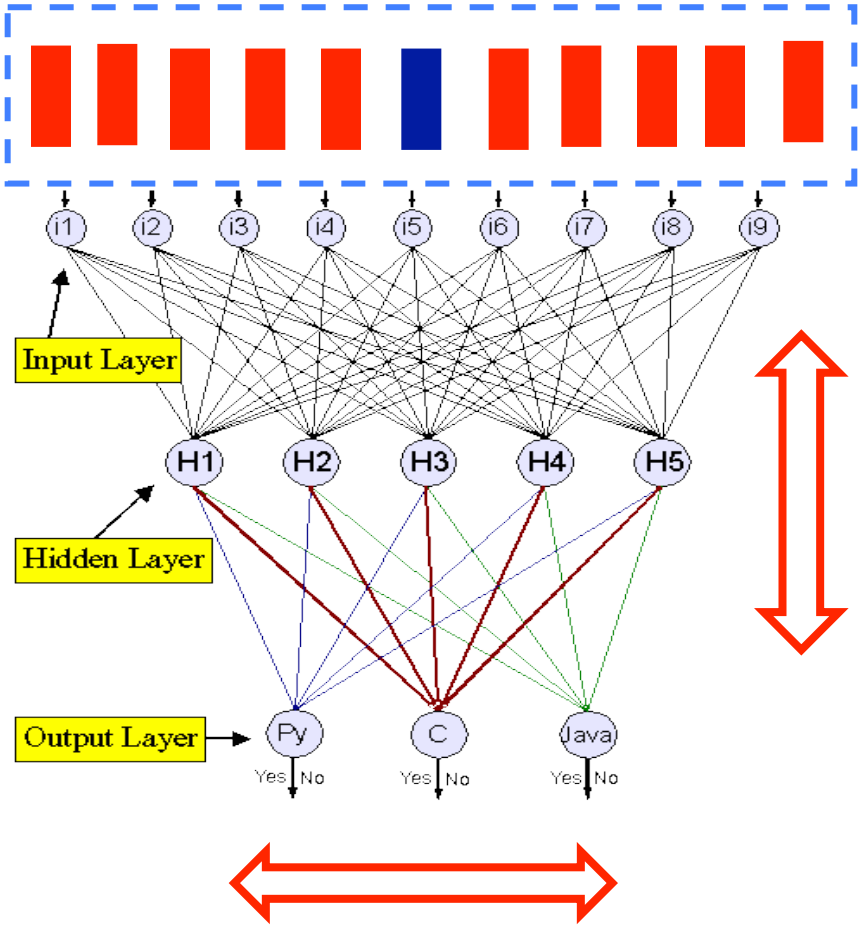
Outline

- Introduction: NN for ASR
- PART I: Why DNN works for ASR
  - DNNs for Bottleneck Features
  - Incoherent Training for DNNs
- PART II: Towards more efficient DNN training
  - DNN with shrinking hidden layers
  - Data-partitioned multi-DNNs
- Summary

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, Yi, et al. 2010)*
- **2011– now: DNN for large-scale ASR**
  - **20-30% rel. gain in Switchboard** *(Seide et al., 2011)*
  - **10-20% rel. gain with sequence training** *(Kingsbury et al., 2012; Su et al., 2013)*
  - **10% rel. gain with CNNs** *(Abdel-Hamid et al., 2012; Sainath et al., 2013)*

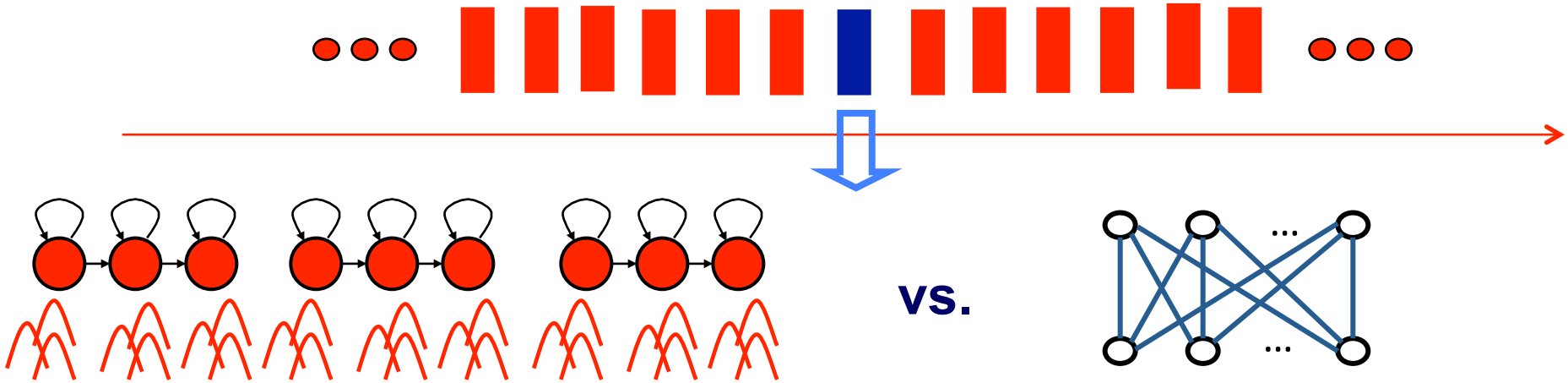
NN for ASR: old and new



- **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 → 2-10 K hours training data**

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%

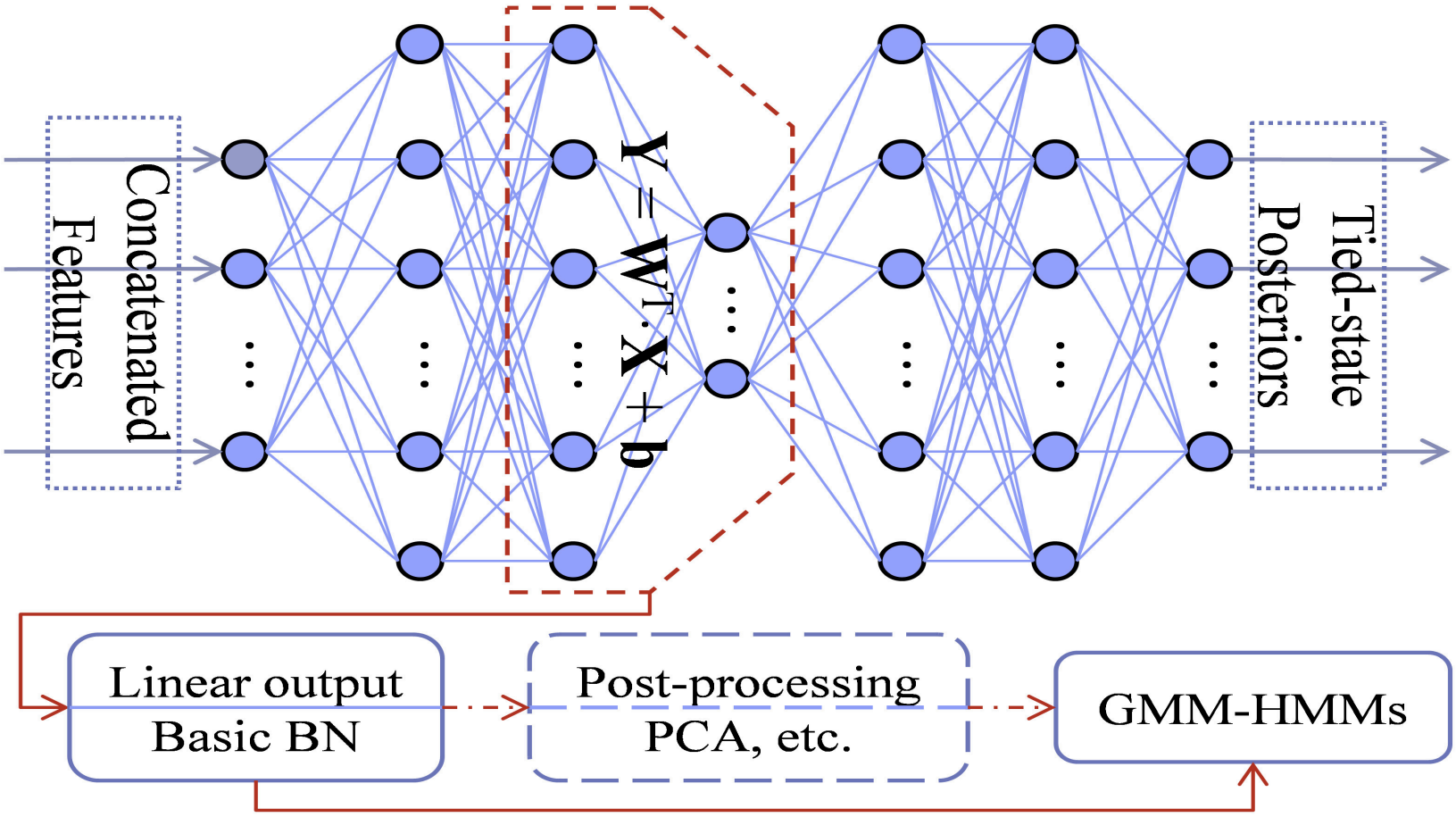
Brief Summary (I)

- 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 large 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

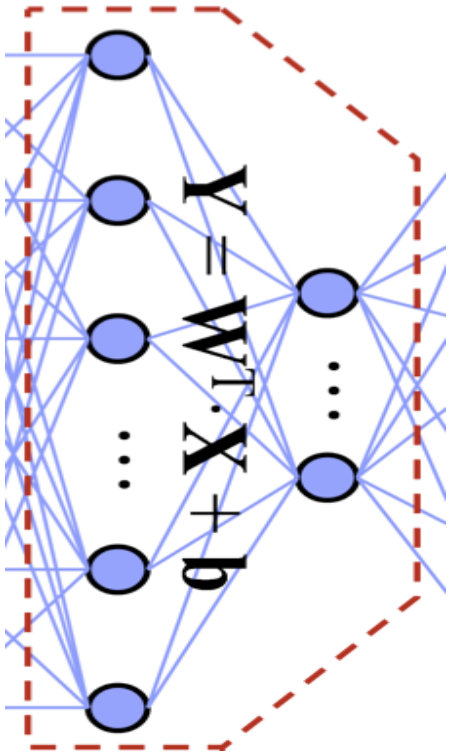
300-hour English Switchboard Task (WER in %)
MLE: maximum likelihood estimation; MPE: discriminative training
ReFA: realigned class labels; DT: sequence training of DNNs
DT+BN: BN features extracted from sequence-trained BN networks

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

- A “better” Idea: embed de-correlation into back-propagation of DNN training.
  - De-correlation by constraining column vectors of weight matrix *W*
  - How to constrain?



Incoherent Training

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

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

- **Intuition: a smaller coherence value indicates all column vectors are more dissimilar.**
- **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:

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

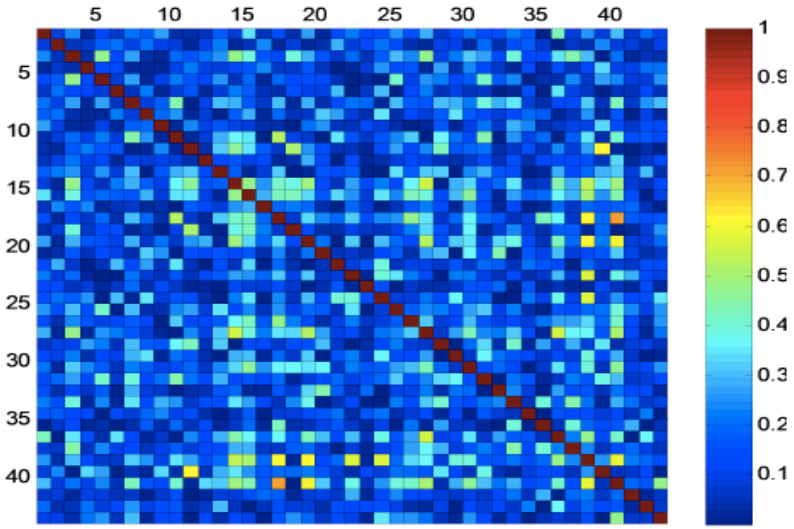
- Derivatives of coherence:

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

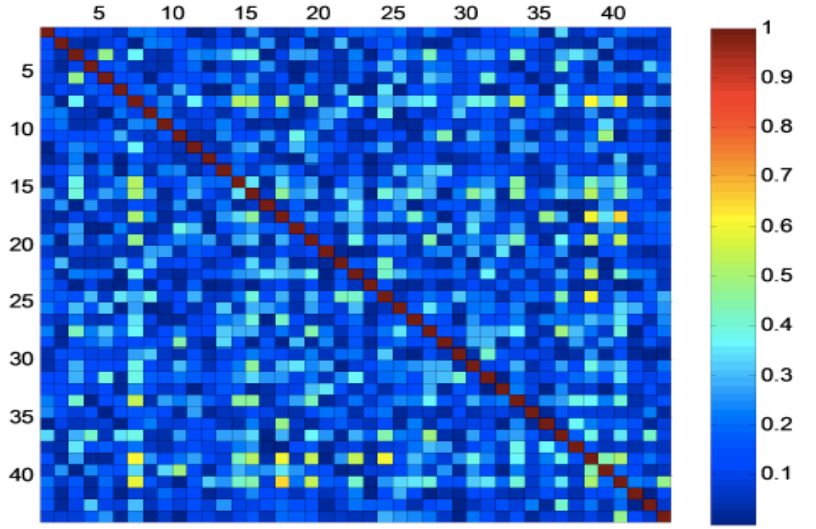
- Back-propagation is still applicable…

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*: $Y = W^T X + b$

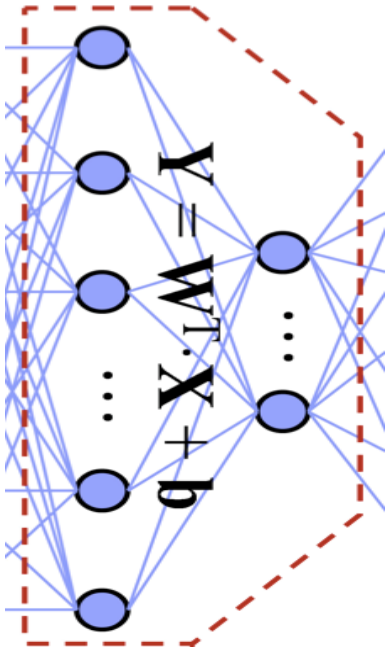
  Covariance matrix of Y: $C_Y = W^T C_X W$

- Directly measure correlation coefficients based on the above covariance matrix:

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

with

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



C_X is estimated from each mini-batch of data

Incoherent Training: Data-driven

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

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

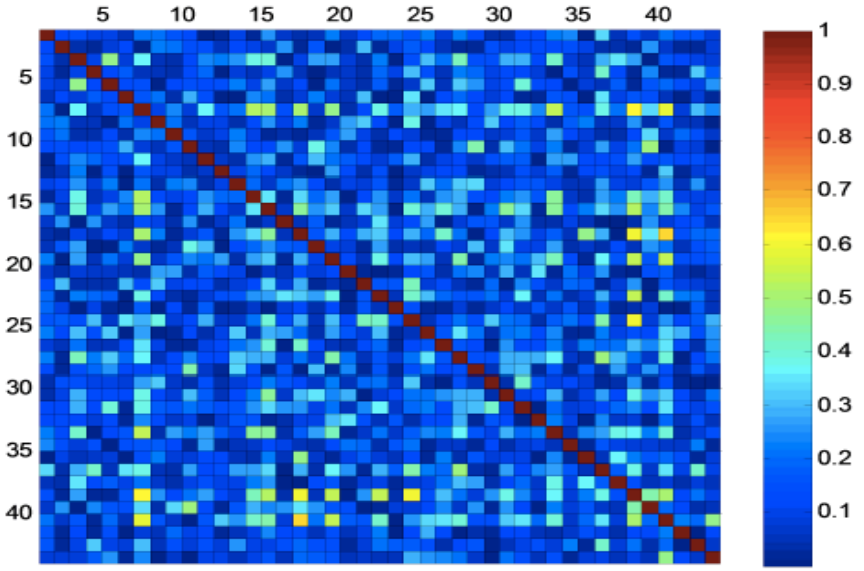
where

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

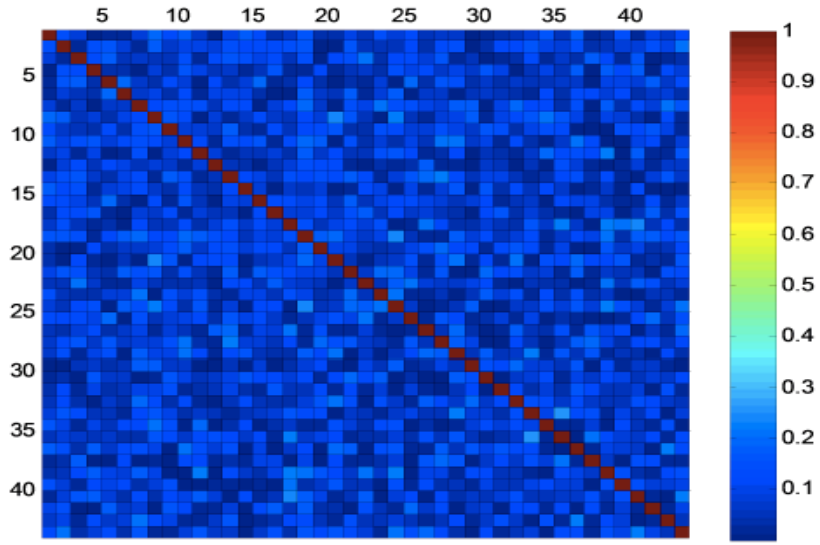
- Back-propagation still applies except Cx is computed from each mini-batch

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%

Brief Summary (II)

- Promising to use DNN as feature extractor for the traditional GMM/HMM framework.
- Beneficial to de-correlate BN using the proposed incoherent training.
- Benefits over hybrid DNN/HMM:
  - Slightly better or similar performance
  - Enjoy other ASR techniques (adaptation, …)
  - Faster training process
  - Faster decoding process

Outline

- Introduction: NN for ASR
- PART I: Why DNN works for ASR
  - DNNs for Bottleneck Features
  - Incoherent Training for DNNs
- PART II: Towards more efficient DNN training
  - DNN with shrinking hidden layers
  - Data-partitioned multi-DNNs
- Summary

Towards Faster DNN Training

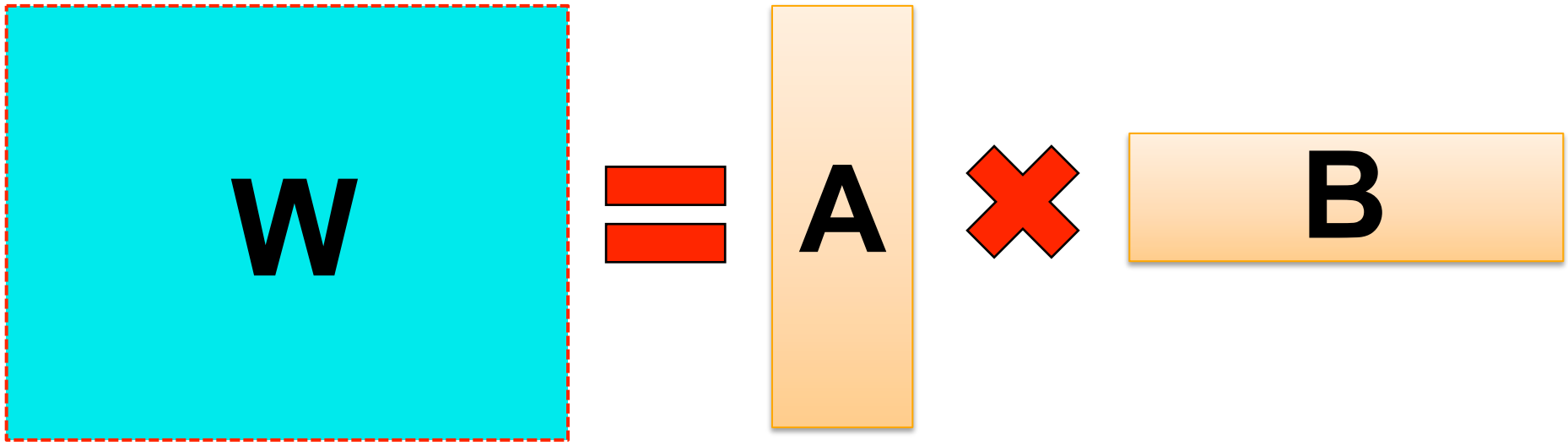
- DNN Training is extremely slow …
  - Taking weeks to months to train large DNNs in ASR
- How to make it more efficient?
  - Simplify DNN model structure …
  - Use training algorithms with faster convergence (than SGD) …
  - Use parallel training methods with many CPUs/GPUs …

Simplify DNN: Exploring Sparseness

- **Sparse DNNs** *(Yu et al, 2012)*: **zeroing 80% of DNN weights leads to no performance loss.**
  - **Smaller model footprint but no gain in speed**

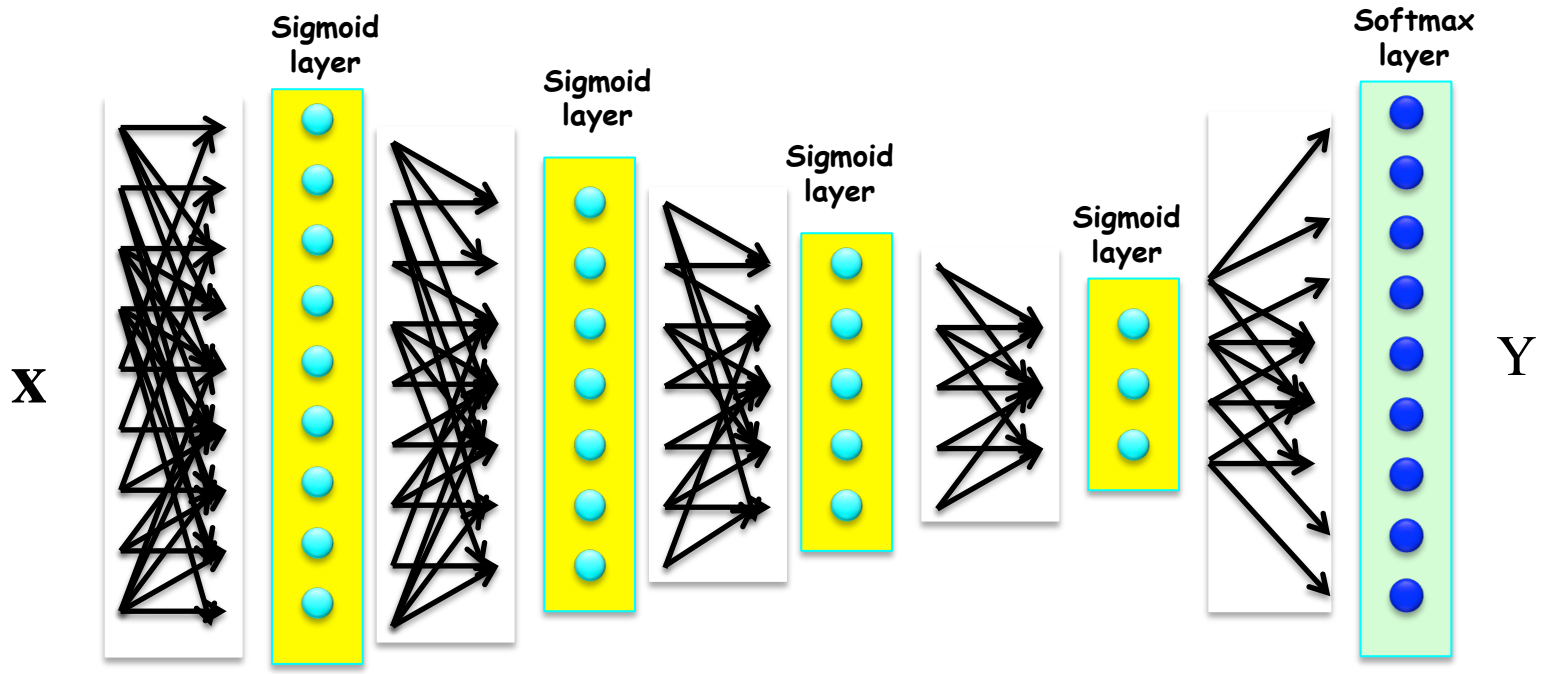
- **How to explore sparseness for speed:**
  - **DNN weight matrix low-rank factorization** *(Sainath et al, 2013 and Xue et al, 2013)*
  - **DNN with shrinking hidden layers** *(to be submitted to ICASSP'14)*

Weight Matrix Factorization



- **IBM** *(Sainath et al, 2013)*: **30-50% smaller model size, 30-50% speedup.**
- **Microsoft** *(Xue et al, 2013)*: **80% smaller model size, >50% speedup.**
- **Our investigation shows 50% speedup in training, 40% in testing.**

DNN: Shrinking Hidden Layers



- Significantly reduce number of weights, particularly the output layer.
- Lead to faster matrix multiplication.

Experiments: Shrinking DNNs

- Switchboard task: 300-hour training data, Hub5e00 test set
- Cross-entropy training (10 epochs of BP; minibatch 1024)

Baseline DNN (6 hidden layer): 429-2048*6-8991

Shrinking DNN I (sDNN-I): 429-2048*2+1024*2+512*2-8991

Shrinking DNN II (sDNN-II): 429-2048-1792-1536-1280-1024-768-8991

	WER	DNN Size	Training time* (speedup)
DNN	16.3%	41M	10h
DNN (+pre-train)	16.1%	41M	≈ 20h (x0.5)
sDNN-I	16.7%	13M (32%)	4.5h (x2.2)
sDNN-II	16.5%	19M (48%)	5.5h (x1.8)

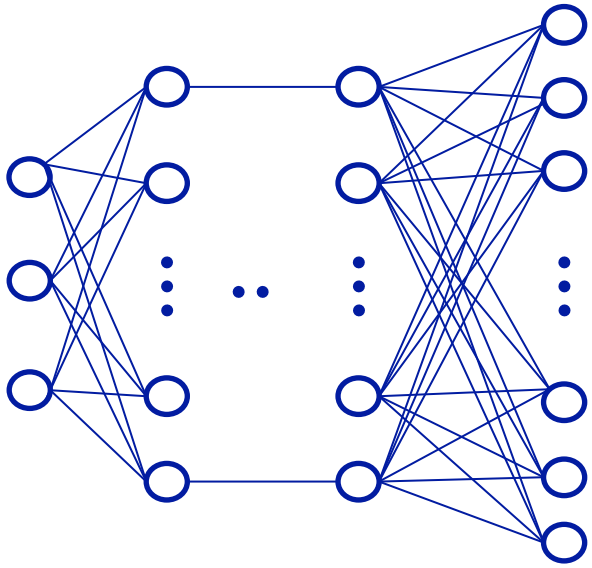
* Average training time (plus pre-training) per epoch using one GTX670

Parallel Training of DNNs

- **If can't make training even faster, why not parallelize it?**
- **Stochastic gradient descent (SGD) is hard to parallelize while second-order optimization (HF) is much higher in complexity.**
- **GPU helps but still not enough: taking weeks to run SGD to train large DNNs for ASR.**
- **GOAL: Parallel Training of DNN using multiple GPUs**
  - **Parallelized (or asynchronous) SGD not optimal for GPUs.**
  - **Pipelined BP** *(Chen et al, 2012)***: data traffic among GPUs.**

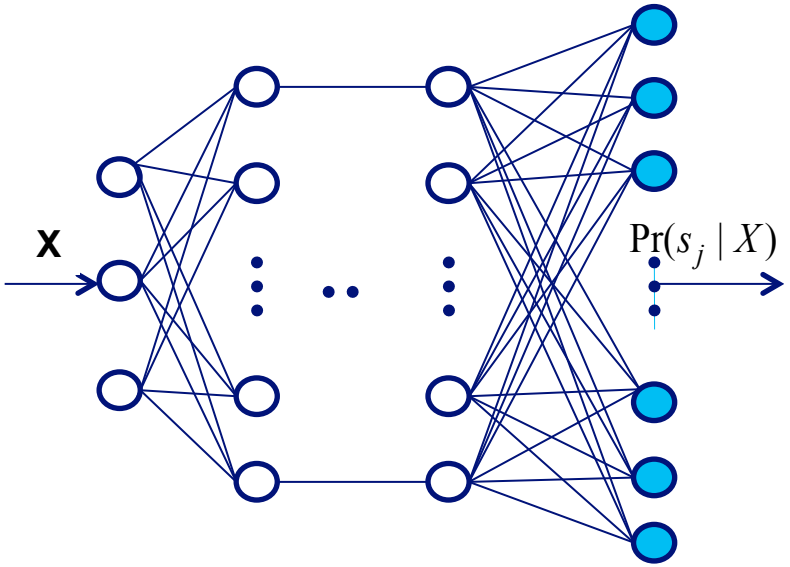
Data-Partitioned Multi-DNNs

Traditional DNN

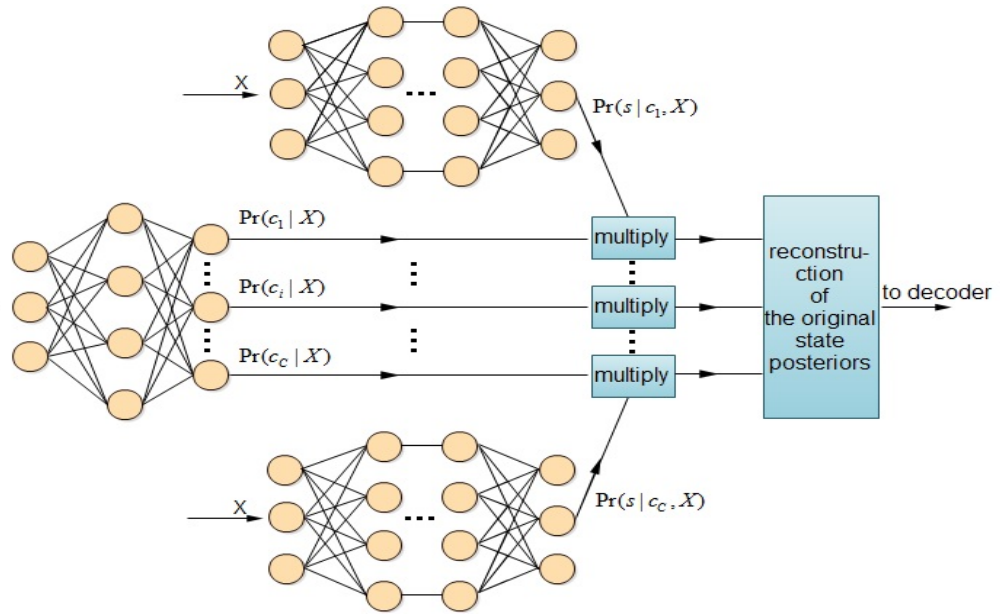


Data-Partitioned Multi-DNNs

Traditional DNN



Multi-DNNs



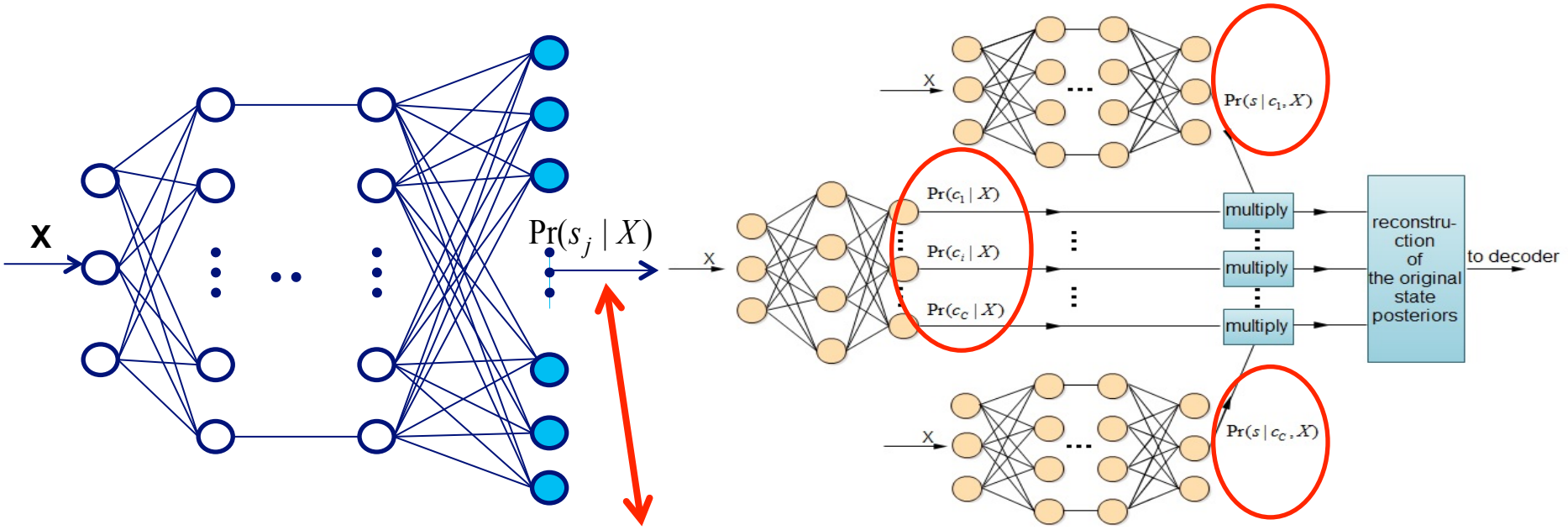
Data Partition

- Unsupervisedly cluster training data into several subsets that have disjoint class labels
- Train DNN on each subset to distinguish labels within each cluster
- Train another top-level DNN to distinguish different clusters

Multi-DNNs: Merging Probabilities

Traditional DNN

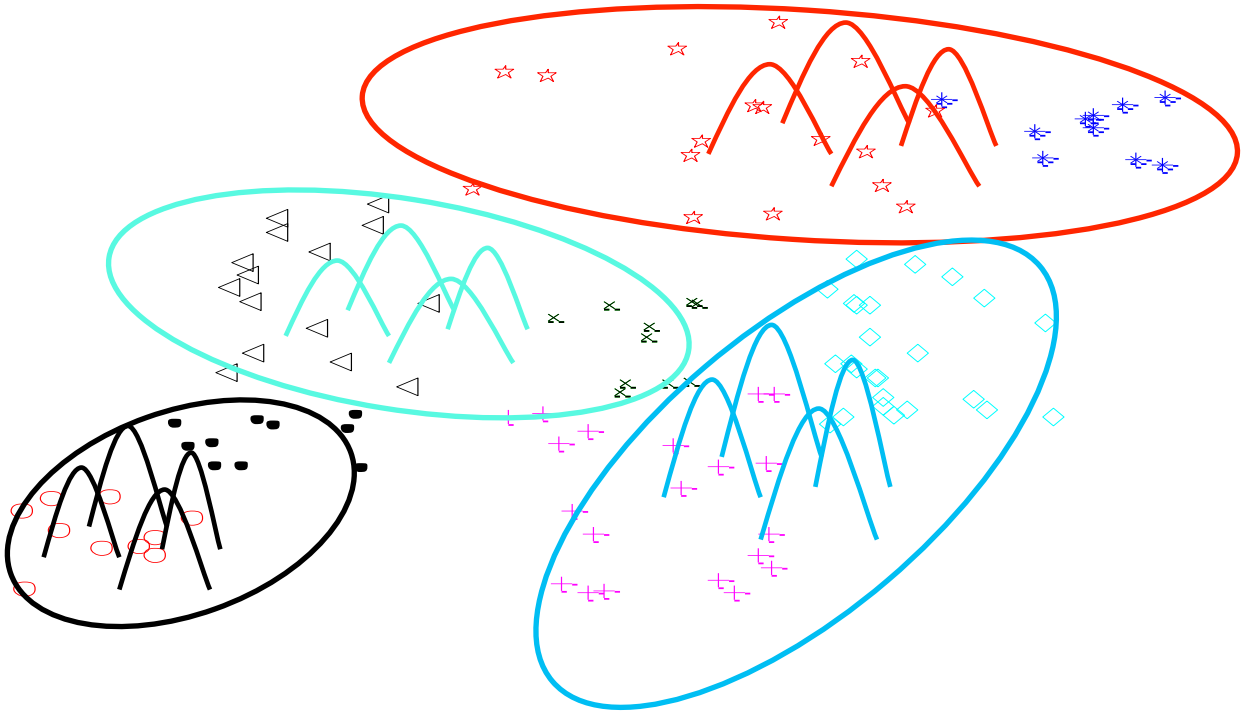
Multi-DNNs



$$\Pr(s_j \mid X) = \Pr(c_i \mid X) \Pr(s_j \mid c_i, X) \quad (with\ s_j \in c_i).$$

Data-driven Clustering

- Iterative GMM based bottom-up clustering of data from all classes (like normal speaker clustering):



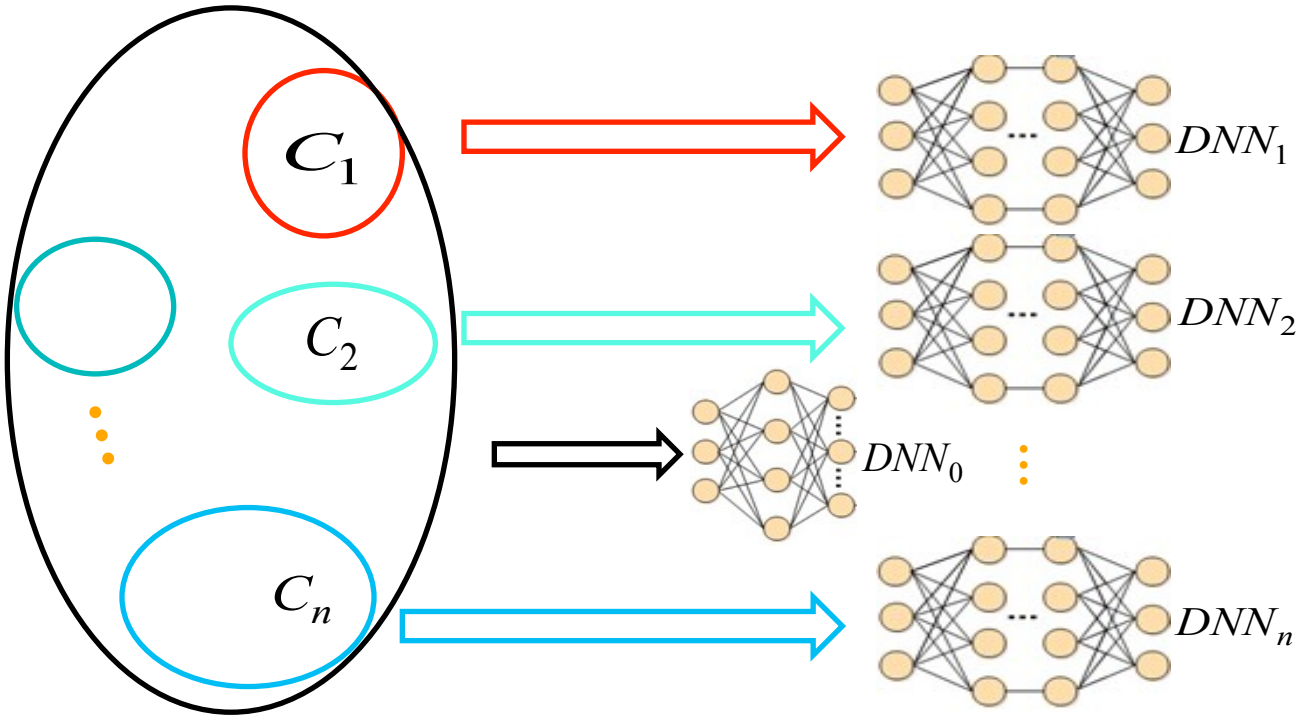
Model Parallelization

- **Advantage of Multi-DNNs:**
  - **Easy to parallelize**

$$e_k = \frac{\partial E^n}{\partial y_K}\frac{\partial y_K}{\partial a_k} = \begin{cases} 0, & k \notin C \\ y_m - t_m, & k \in C \end{cases}$$

  - **Each DNN is learned only from one subset**

Parallel Training Scheme



ZERO communication data across GPUs

Experiment: Multi-DNNs

- Switchboard training data (300 hour, 8991 classes)
- Test sets：Hub01 and Hub00
- GMM-based bottom-up clustering method to group training data into 4 subsets to train 4 DNNs in parallel

	C1	C2	C3	C4
Num. of states	2450	3661	20	2860
Data %	45.0%	25.3%	9.3%	20.4%

Experiments: Multi-DNNs

Baseline DNN: 4-6 hidden layers of 2048 nodes
Multi-DNNs: DNN1-4 has 6 hidden layers of 2048 nodes
DNN0 has 3 hidden layers of 2048 nodes

	Hidden layer	4	5	6
baseline DNN	WER	24.4%	23.7%	23.6%
	Time (hr)	13.0h	13.5h	15.1h
Multi-DNNs (3 GPUs)	WER	24.5%	24.2%	23.8%
	Time (hr)	3.7h	4.3h	4.9h
	Speed-up	x3.5	x3.1	x3.1

Switchboard ASR: WER (in %) on Hub5e01 set and training time per epoch using 3 GPUs

Experiments: Multi-DNNs

Baseline DNN: 4-6 hidden layers of 2048 nodes
Multi-DNNs: DNN1-4 has 6 hidden layers of 2048 nodes
DNN0 has 3 hidden layers of 2048 nodes

	Hidden layer	4	5	6
baseline DNN	WER	17.0%	16.7%	16.2%
	Time (hr)	13.0h	13.5h	15.1h
Multi-DNNs (3 GPUs)	WER	17.0%	16.9%	16.7%
	Time (hr)	3.7	4.3	4.9
	Speed-up	x3.5	x3.1	x3.1

Switchboard ASR: WER (in %) on Hub5e00 set and training time per epoch using 3 GPUs

Experiments: Smaller Multi-DNNs

Baseline DNN: 4-6 hidden layers of 2048 nodes
Multi-DNNs: DNN1-4 has 6 hidden layers of 1024 nodes
DNN0 has 3 hidden layers of 1024 nodes

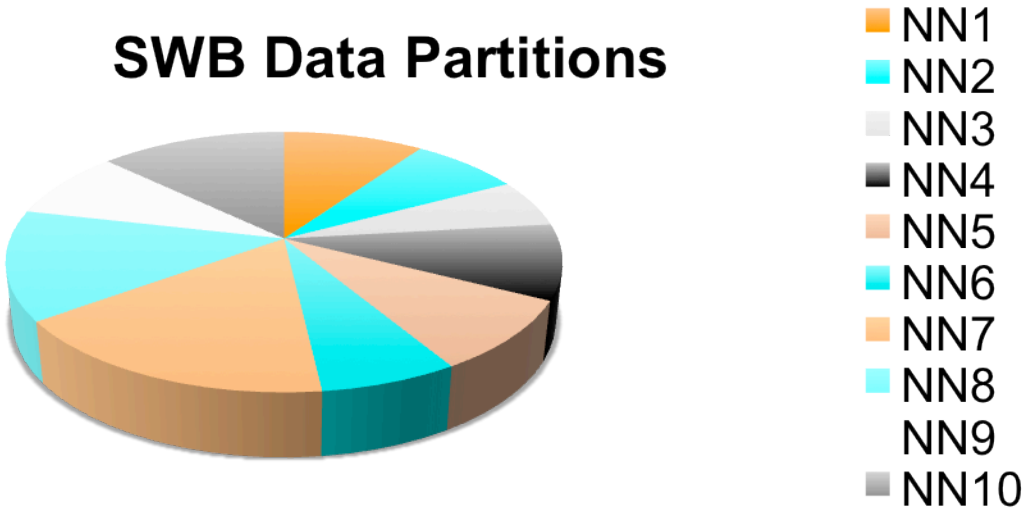
	Hidden layer	4	5	6	ReFA
baseline DNN	WER	17.0%	16.7%	16.2%	15.9%
	Time (hr)	13.0h	13.5h	15.1h	
Multi-DNNs (3 GPUs)	WER	17.8%	17.6%	17.4%	16.4%
	Time (hr)	1.8h	2.1h	2.3h	
	Speed-up	x7.0	x6.6	x6.5	

Switchboard ASR: WER (in %) on Hub5e00 set and training time per epoch using 3 GPUs

More Clusters for better Speedup

- Clustering SWB training data (300-hr, 8991 classes) to 10 clusters

	NN0	DNN1	DNN2	DNN3	DNN4	DNN5	DNN6	DNN7	DNN8	DNN9	DNN10
%	100%	9.7%	7.6%	5.9%	9.3%	8.9%	6.8%	16.8%	13.6%	8.5%	12.9%
class	10	1258	1162	993	1232	1042	1127	40	357	926	854



More Clusters for Faster Speed

- Baseline: single DNN with 2048 nodes per hidden layer
- Multi-DNNs: 1200 hidden nodes per layer for DNN1-10; NN0 is 4*2048

Hidden Layers		3	4	5	6
Baseline	WER	17.8%	17.0%	16.9%	16.2%
	Time (h)	11.0 h	13.0 h	13.5 h	15.0h
Multi-DNN (10 GPUs)	WER	17.7%	17.7%	17.4%	17.4%
	Time (h)	0.6h	0.7h	0.8h	0.9h
	speedup	x18.5	x18.9	x16.6	x16.3

Switchboard ASR: WER (in %) on Hub5e00 set
and training time per epoch using 10 GPUs

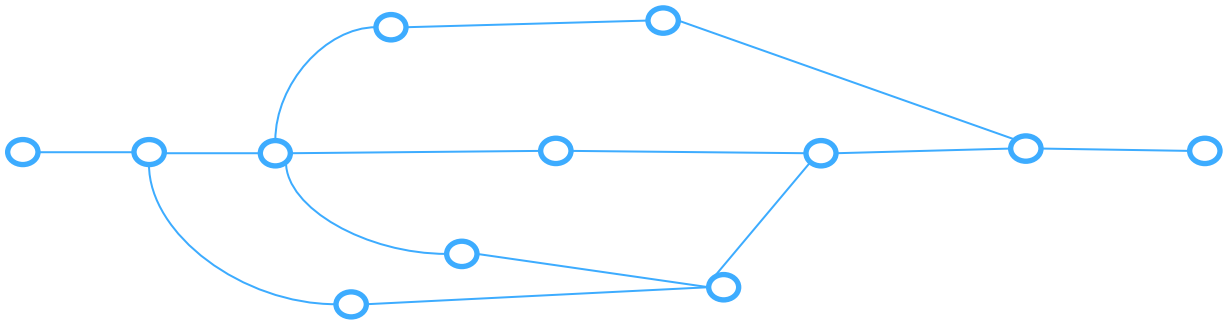
Sequence Training of Multi-DNNs

- SGD-based sequence training of DNNs using GPUs
  - H-Criterion for smoothing MMI *(Su et al., 2013)*
  - Implementing BP/SGD and lattice computation in GPU(s)

- For each mini-batch (utterances and word graphs)
  - ① DNN forward pass (parallel in 1 vs. N GPUs)
  - ② States occupancy for all arcs (parallel in 1 vs. N GPUs)
  - ③ Process lattices for arc posteriori probs (CPU vs. 1 GPU)
  - ④ Sum state statistics for all arcs (parallel in 1 vs. N GPUs)
  - ⑤ DNN back-propagation pass (parallel in 1 vs. N GPUs)

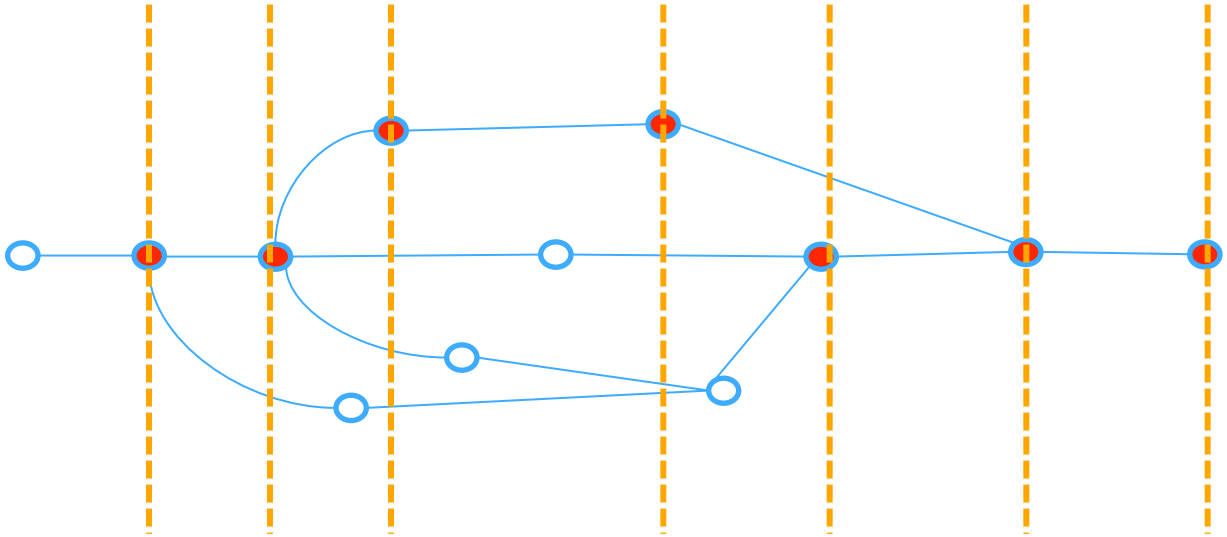
Process word graphs with GPU

- Sort all arcs based on starting time



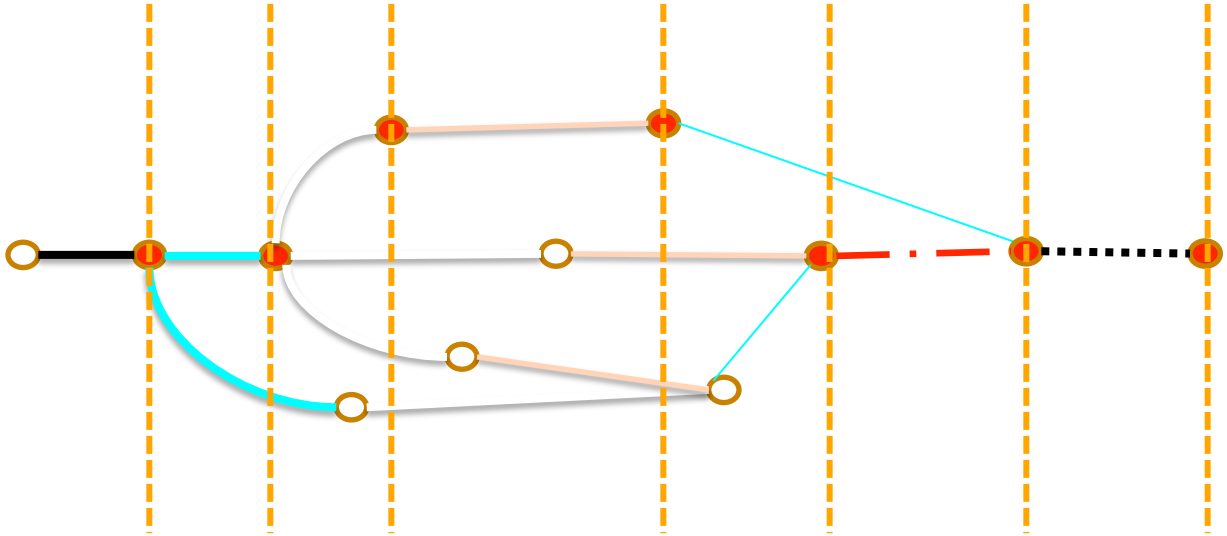
Process word graphs with GPU

- Find splitting nodes in the sorted list based on max starting time and min ending time of arcs.



Process word graphs with GPU

- Split all arcs into subsets for different CUDA launches.
- In each launch, arcs do forward-backward in parallel.



Experiment: Sequence Training of 4-cluster Multi-DNNs

Baseline DNN: 6 hidden layers of 2048 nodes
Multi-DNNs: DNN1-4 has 6 hidden layers of 1024 or 1200 nodes
DNN0 has 3 hidden layers of 1200 nodes

	CE (FA)	DT	Speedup (3 GPUs)
Baseline DNN	15.9%	14.2%	--
Multi-DNNs 6x1024	16.4%	15.4%	x4.1**
Multi-DNNs 6x1200	16.1%	15.2%*	x3.6**

CE (FA): 10 epochs of CE training using realigned labels
DT: CE(FA) plus one iteration of sequence training
* Mismatched lattices; ** based on simulation estimation

Final Remarks

- DNN PAIN: extremely time-consuming to train DNNs.
- Critical to expedite DNN training for big data sets.
- DNN training can be largely accelerated via:
  - Simplify model structure by exploring sparseness
  - Employ parallel training using multiple GPUs