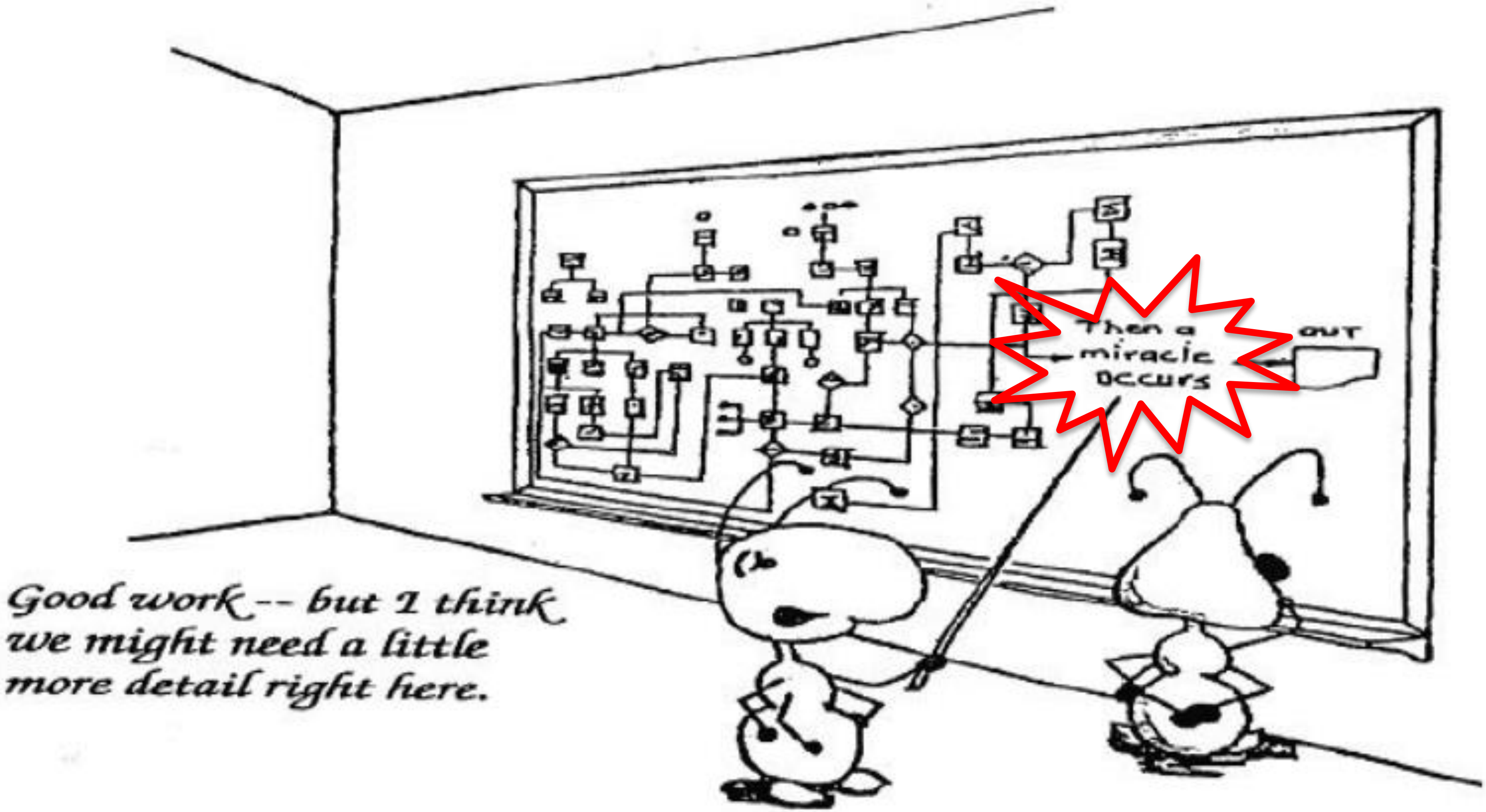
A Few Thoughts on How We May Want to Further Study DNN

Eric Xing
Carnegie Mellon University

Deep Learning is Amazing!!!

What makes it work? Why?



An MLer's View of the World

Loss functions
(likelihood, reconstruction, margin, …)

Structures
(Graphical, group, chain, tree, iid, …)

Constraints
(normality, sparsity, label, prior, KL, sum, …)

Algorithms
MC (MCMC, Importance), Opt (gradient, IP), …

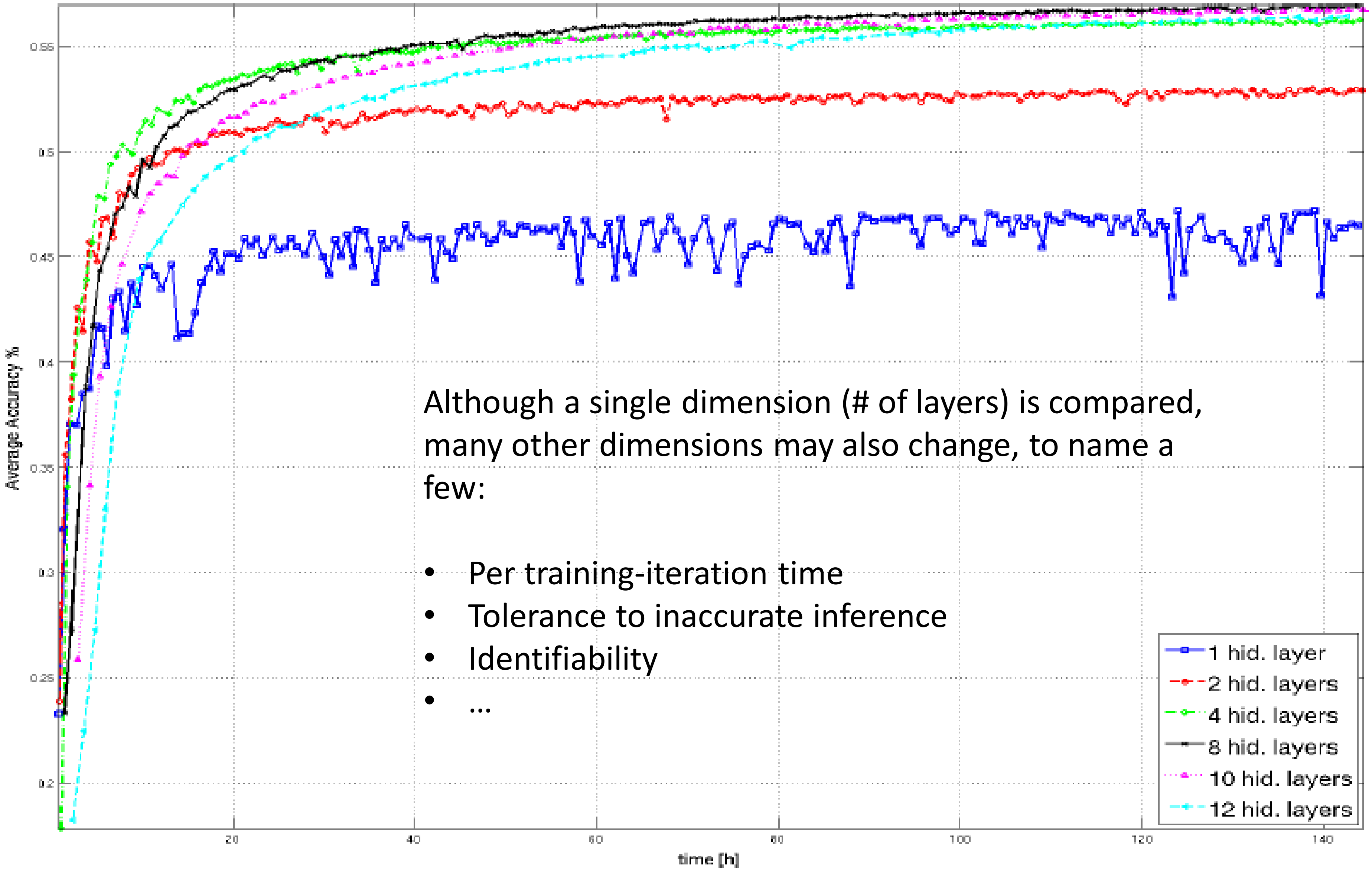
Stopping criteria
Change in objective, change in update …

	DL	ML (e.g., GM)
Empirical goal:	e.g., classification, feature learning	e.g., transfer learning, latent variable inference
Structure:	Graphical	Graphical
Objective:	Something aggregated from local functions	Something aggregated from local functions
Vocabulary:	Neuron, activation/gate function …	Variables, potential function
Algorithm:	A single, unchallenged, inference algorithm -- BP	A major focus of open research, many algorithms, and more to come
Evaluation:	On a black-box score -- end performance	On almost every intermediate quantity
Implementation:	Many untold-tricks	More or less standardized
Experiments:	Massive, real data (GT unknown)	Modest, often simulated data (GT known)

A slippery slope to heuristics

- How to conclusively determine what an improve in performance could come from:
  - Better model (architecture, activation, loss, size)?
  - Better algorithm (more accurate, faster convergence)?
  - Better training data?
- Current research in DL seem to get everything above mixed by evaluating on a black-box “performance score” that is not directly reflecting
  - Correctness of inference
  - Achievability/usefulness of model
  - Variance due to stochasticity

An Example



Although a single dimension (# of layers) is compared, many other dimensions may also change, to name a few:

- Per training-iteration time
- Tolerance to inaccurate inference
- Identifiability
- …

Inference quality

- Training error is the old concept of a classifier with no hidden states, no **inference** is involved, and thus inference accuracy is not an issue
- But a DNN is not just a classifier, some DNNs are not even fully supervised, there are MANY **hidden states**, why their inference quality is not taken seriously?

- In DNN, inference accuracy = visualizing features
  - Study of inference accuracy is badly discouraged
  - Loss/accuracy is not monitored

Inference/Learning Algorithm, and their evaluation

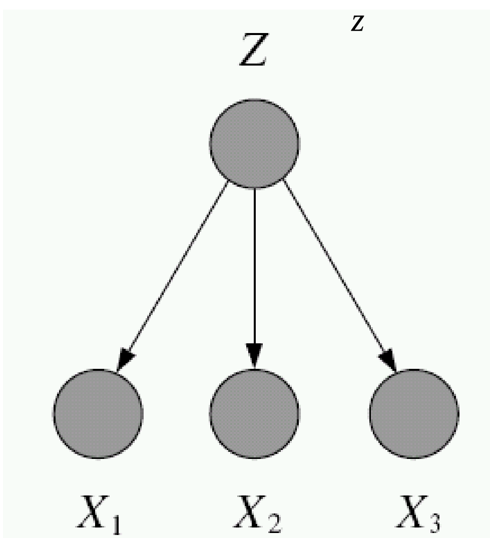
Learning in GM with Hidden Variables

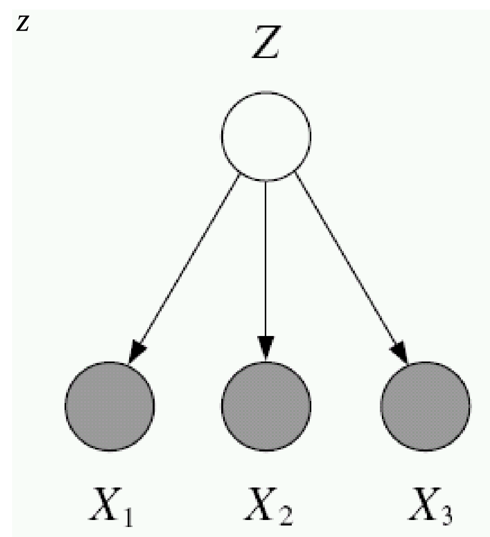
- In fully observed iid settings, the log likelihood decomposes into a sum of local terms (at least for directed models).

$$\ell_c(\theta; D) = \log p(x, z \mid \theta) = \log p(z \mid \theta_z) + \log p(x \mid z, \theta_x)$$

- With latent variables, all the parameters become coupled together via marginalization

$$\ell_c(\theta; D) = \log \sum_z p(x, z \mid \theta) = \log \sum_z p(z \mid \theta_z) p(x \mid z, \theta_x)$$





Gradient Learning for mixture models

- We can learn mixture densities using gradient descent on the log likelihood. The gradients are quite interesting:

$$l(\theta) = \log p(\mathrm{x} \mid \theta) = \log \sum_k \pi_k p_k(\mathrm{x} | \theta_k)$$

$$\frac{\partial l}{\partial \theta_k} = \frac{1}{p(\mathrm{x} \mid \theta)} \sum_k \pi_k \frac{\partial p_k(\mathrm{x} | \theta_k)}{\partial \theta_k}$$

$$= \sum_k \frac{\pi_k}{p(\mathrm{x} \mid \theta)} p_k(\mathrm{x} | \theta_k) \frac{\partial \log p_k(\mathrm{x} | \theta_k)}{\partial \theta_k}$$

$$= \sum_k \pi_k \frac{p_k(\mathrm{x} | \theta_k)}{p(\mathrm{x} \mid \theta)} \frac{\partial \log p_k(\mathrm{x} | \theta_k)}{\partial \theta_k} = \sum_k r_k \frac{\partial l_k}{\partial \theta_k}$$

- In other words, the gradient is aggregated from many other intermediate states
  - Implication: costly iteration, heavy coupling between parameters

Parameter Constraints

- Often we have constraints on the parameters, e.g. $\Sigma_k \pi_k = 1$, Σ being symmetric positive definite (hence $\Sigma_{ii} > 0$).
- We can use constrained optimization, or we can reparameterize in terms of unconstrained values.
  - For normalized weights, use the softmax transform: $\pi_k = \frac{\exp(\gamma_k)}{\Sigma_j \exp(\gamma_j)}$
  - For covariance matrices, use the Cholesky decomposition:

$$\Sigma^{-1} = \mathbf{A}^T \mathbf{A}$$

where A is upper diagonal with positive diagonal:

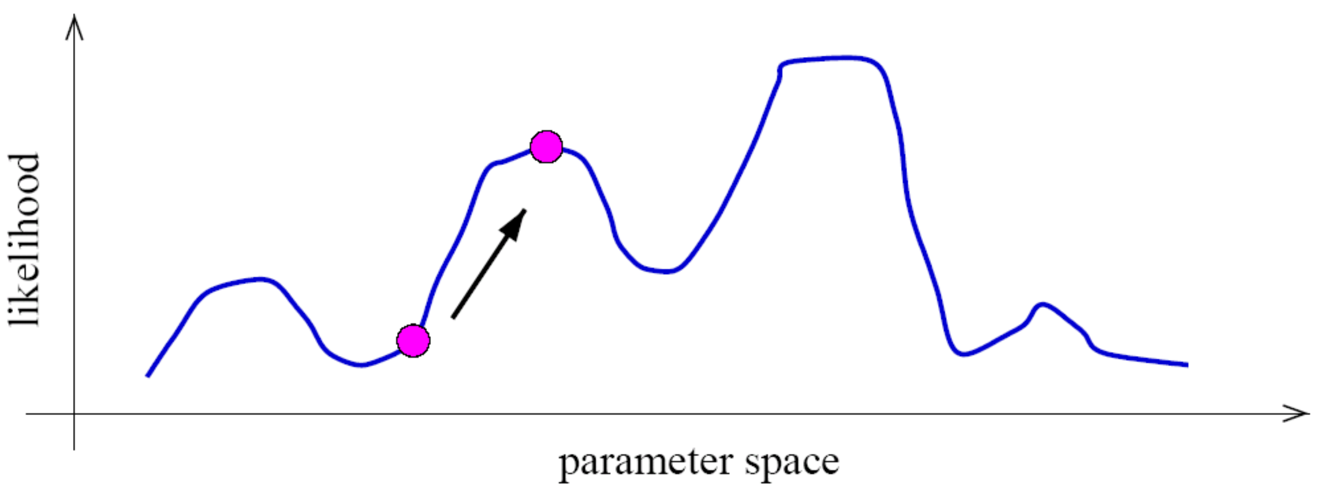
$$\mathbf{A}_{ii} = \exp(\lambda_i) > 0 \quad \mathbf{A}_{ij} = \eta_{ij} \ (j > i) \quad \mathbf{A}_{ij} = 0 \ (j < i)$$

the parameters γ_i, λ_i, $\eta_{ij} \in \mathbb{R}$ are unconstrained.

  - Use chain rule to compute $\frac{\partial \boldsymbol{\ell}}{\partial \pi}, \frac{\partial \boldsymbol{\ell}}{\partial \mathbf{A}}$.

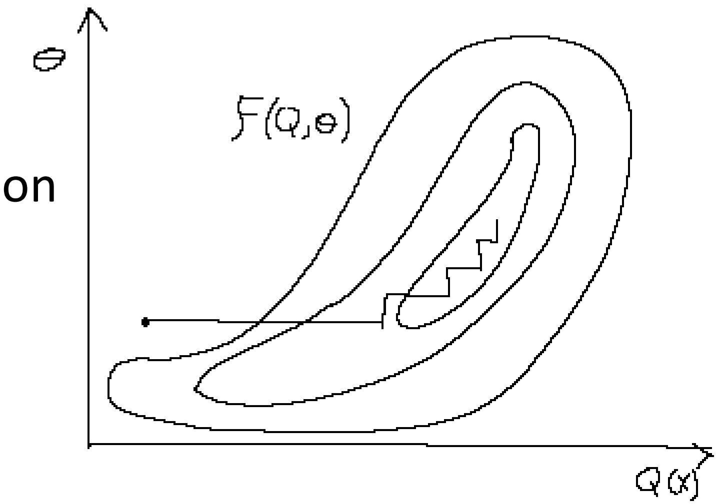
Identifiability

- A mixture model induces a multi-modal likelihood.
- Hence gradient ascent can only find a local maximum.
- Mixture models are unidentifiable, since we can always switch the hidden labels without affecting the likelihood.
- Hence we should be careful in trying to interpret the "meaning" of latent variables.



Then Alternative Approaches Were Proposed

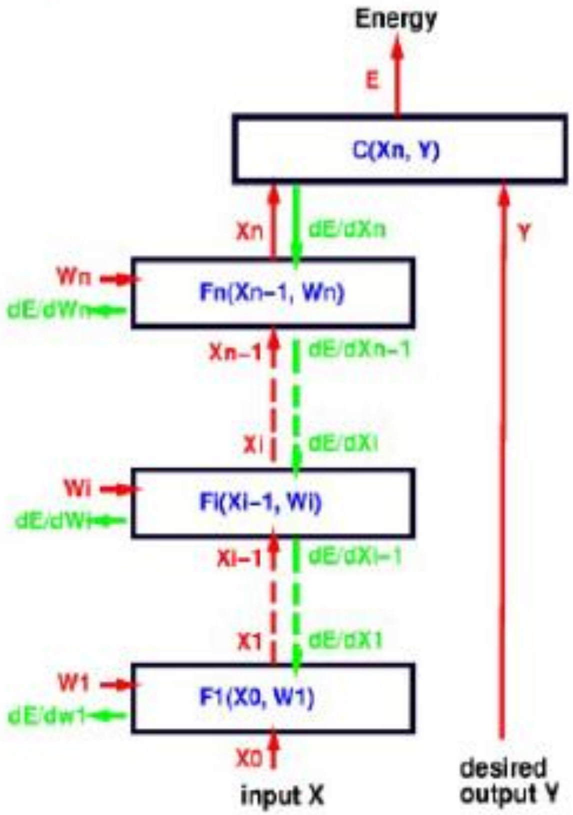
- The EM algorithm
  - M: a convex problem
  - E: approximate constrained optimization
    - Mean field
    - BP/LBP
    - Marginal polytope



- Spectrum algorithm:
  - redefine intermediate states, convexify the original problem

Learning a DNN

To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

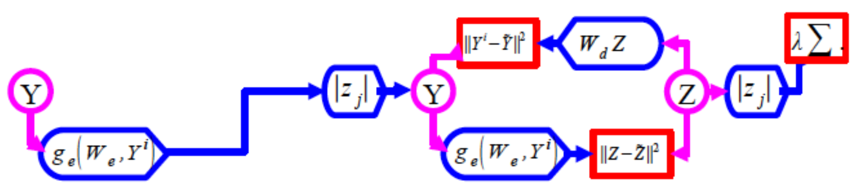
Learning a DNN

- In a nutshell, sequentially, and recursively apply:

$$w_{j,i}^{t+1} = w_{j,i}^{t} - \eta_t \delta_j z_i$$

$$\delta_i = h^{'}(a_i) \sum_j \delta_j w_{j,i}$$

- Things can getting hairy when locally defined losses are introduced, e.g., auto-encoder, which breaks a loss-driven global optimization formulation



- Depending on starting point, BP converge or diverge with probability 1
  - A serious problem in Large-Scale DNN

Backprop in Practice

Y LeCun
MA Ranzato

- Use ReLU non-linearities (tanh and logistic are falling out of favor)
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
  - But it's best to turn it on after a couple of epochs
- Use “dropout” for regularization
  - Hinton et al 2012 http://arxiv.org/abs/1207.0580
- Lots more in [LeCun et al. “Efficient Backprop” 1998]
- Lots, lots more in “Neural Networks, Tricks of the Trade” (2012 edition) edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)

DL

Utility of the network

- A vehicle for synthesize complex decision hypothesis
  - stage-wise projection and aggregation
- A vehicle for organizing computing operations
  - stage-wise update of latent states
- A vehicle for designing processing steps/computing modules
  - Layer-wise parallization
- No obvious utility in evaluating DL algorithms

Utility of the Loss Function

- Global loss? Well it is non-convex anyway, why bother ?

GM

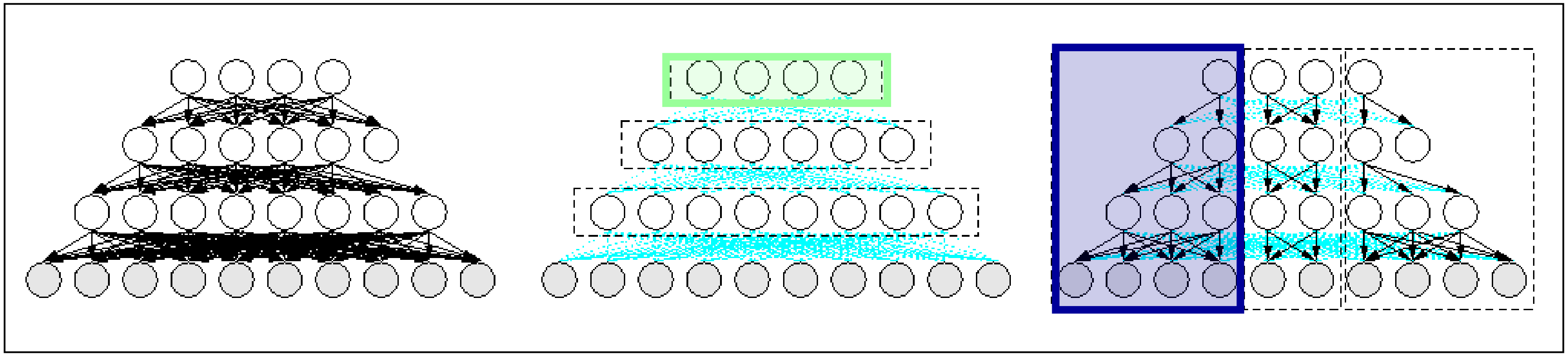
- A vehicle for synthesize a global loss function from local structure
  - potential function, feature function
- A vehicle for designing sound and efficient inference algorithm
  - Sum-product, mean-field
- A vehicle to inspire approximation and penalization
  - Structured MF, Tree-approx
- Vehicle for monitoring theoretical and empirical behavior and accuracy of inference

- A major measure of quality of algorithm and model

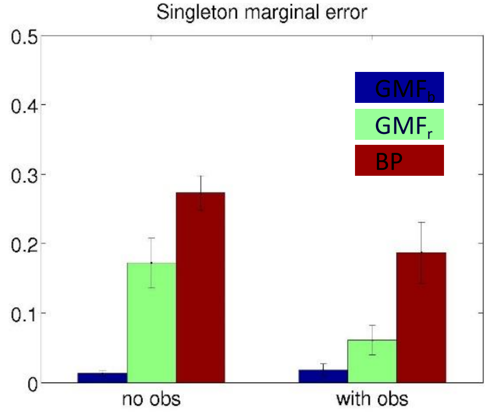
An Old Study of DL as GM Learning

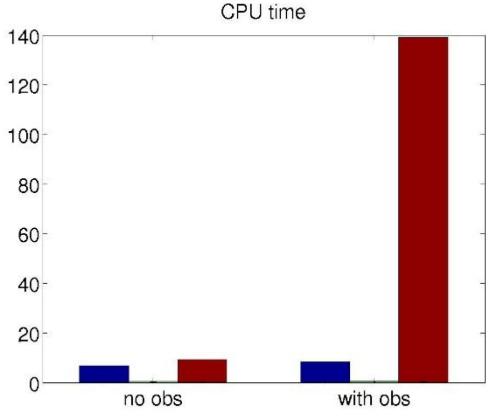
[Xing, Russell, Jordan, UAI 2013]

A sigmoid belief network, and mean-field partitions



Study focused on only inference/learning accuracy, speed, and partition



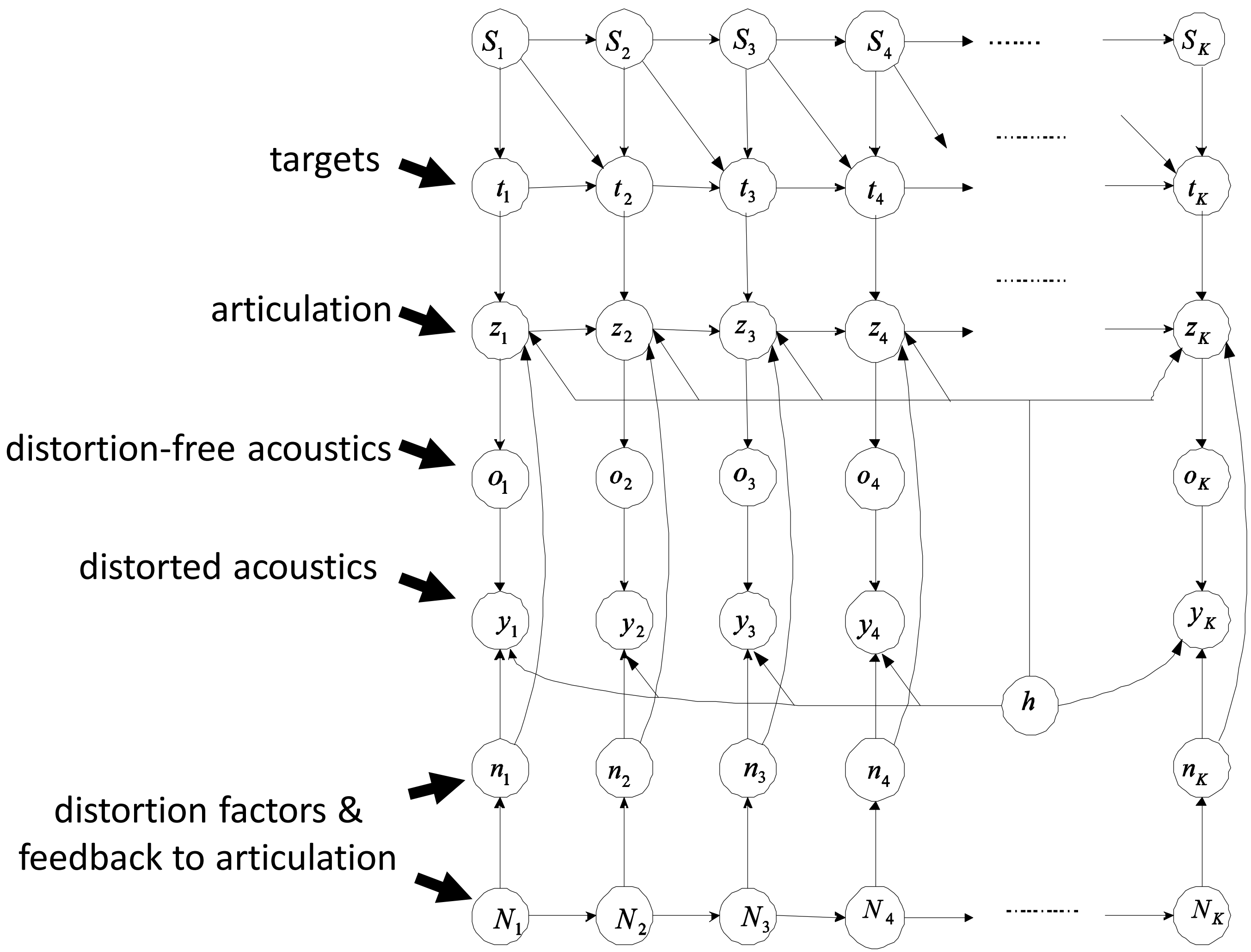


Now we can ask, with a correctly learned DN, is it doing will on the desired task?

Why A Graphical Model formulation of DL might be fruitful

- Modular design: easy to incorporate knowledge and interpret, easy to integrate feature learning with high level tasks, easy to built on existing (partial) solutions
- Defines an explicit and natural learning objective
- Guilds strategies for inference, parallelization, evaluation, and theoretical analysis
- A clear path to further upgrade:
  - structured prediction
  - Integration of multiple data modality
  - Modeling complex: time series, missing data, online data …
- Big DL on distributed architectures, where things can get messy everywhere due to incorrect parallel computations

Easy to incorporate knowledge and interpret



Slides Courtesy:
Li Deng

Easy to integrate feature learning with high level tasks

Hidden Markov Model

+

Gaussian Mixture Model

Jointly trained, but shallow

Hidden Markov Model

+

Deep Neural Network

Deep, but separately trained

Hidden Markov Model

+

Deep Graphical Models

Jointly trained and deep

Mathematics 101 for ML

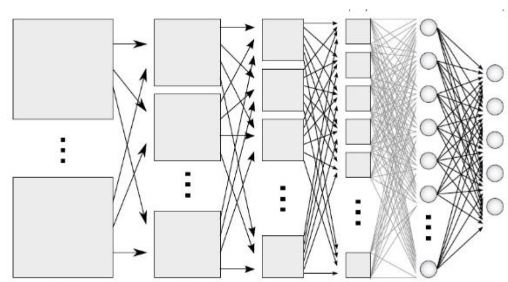
$$\arg\max_{\vec{\theta}} \equiv \mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \; ; \; \vec{\theta}) + \Omega(\vec{\theta})$$

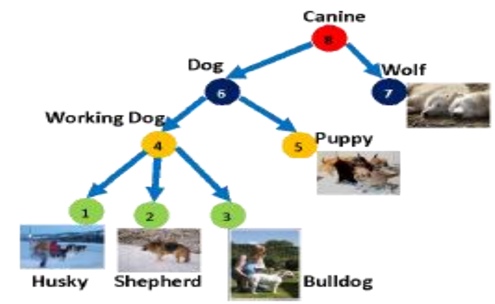
Model Data Parameter

$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

This computation needs to be parallelized!

Toward Big ML

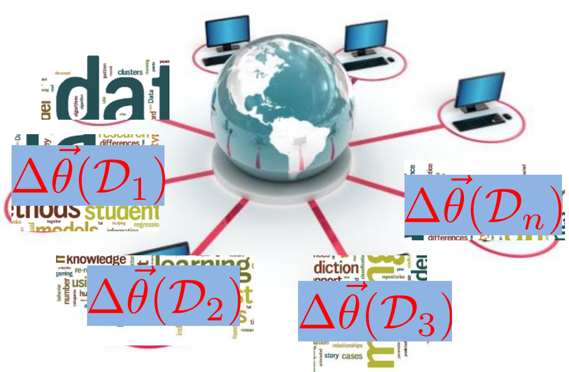


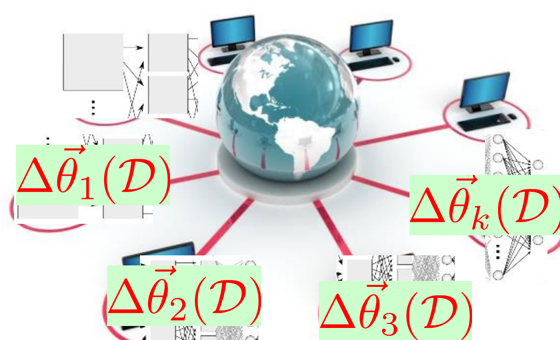


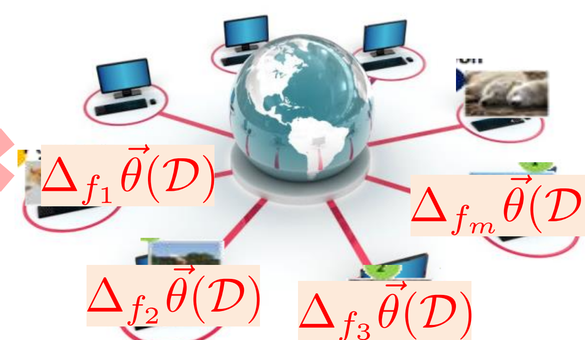
Data Parallel

Model Parallel

Task Parallel







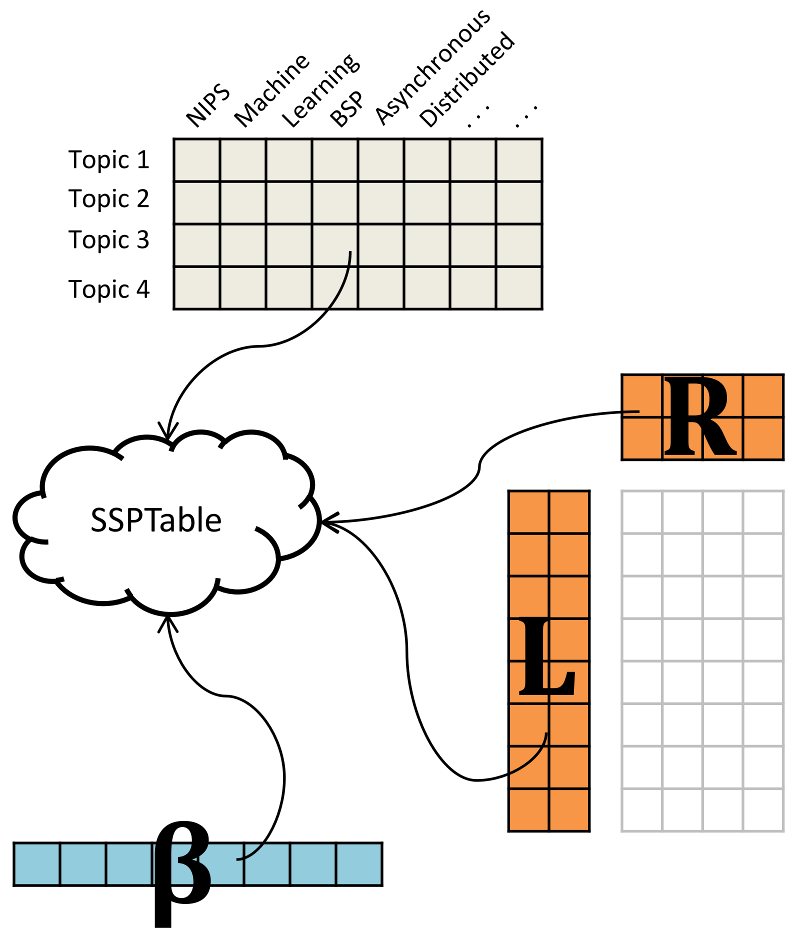
$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$

$\vec{\theta} \equiv [\vec{\theta}_1^{\mathsf{T}}, \vec{\theta}_2^{\mathsf{T}}, \ldots, \vec{\theta}_k^{\mathsf{T}}\}^{\mathsf{T}}$

$f \equiv \{f_1, f_2, \ldots, f_m\}$

Data-Parallel DNN using Petuum Parameter Server

- Just put global parameters in SSPTable:
- **DNN (SGD)**
  - **The weight table**
- **Topic Modeling (MCMC)**
  - Topic-word table
- **Matrix Factorization (SGD)**
  - Factor matrices L, R
- **Lasso Regression (CD)**
  - Coefficients β
- SSPTable supports generic classes of algorithms
  - With these models as examples

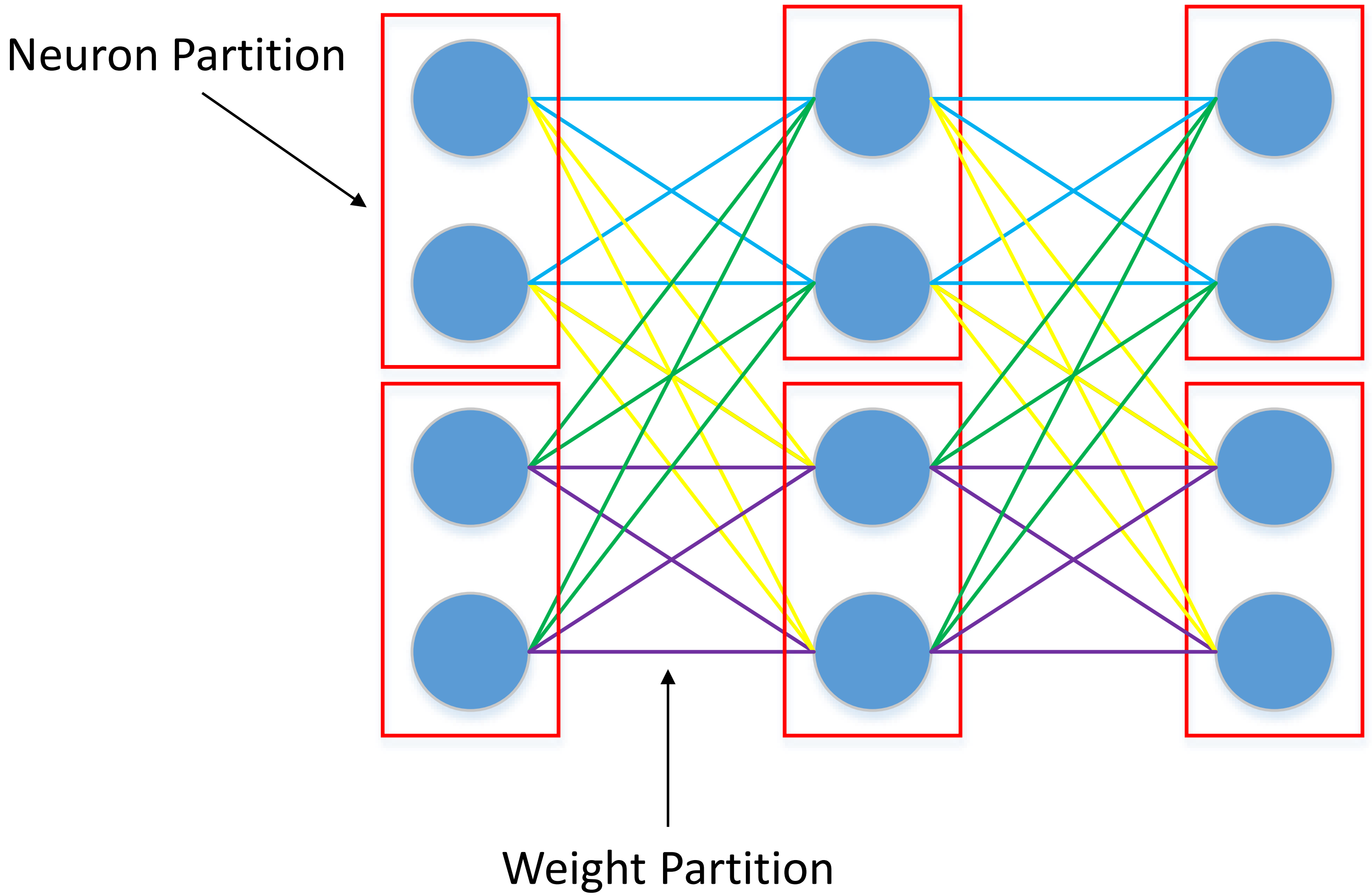


Theorem: Multilayer convergence of SSP based distributed DNNs to optima

- If the undistributed BP updates of a multi-layer DNN lead to weights w_t, and the distributed BP updates under SSP lead to weights $\tilde{w}_t$, then $\tilde{w}_t$ converges in probability to w_t, i.e . $(\tilde{w}_t \xrightarrow{P} w_t)$

Consequently $(\tilde{w}_t^* \xrightarrow{P} w^*)$

Model-Parallel DNN using Petuum Scheduler



Theorem: Multilayer convergence of model distributed DNNs to optima

- If the undistributed BP updates of a multi-layer DNN lead to weights w_t and the distributed BP updates in model distributed setting lead to weights $\tilde{w}_t$, then $\tilde{w}_t$ converges in probability to w_t, i.e . $(\tilde{w}_t \xrightarrow{P} w_t)$ .Consequently

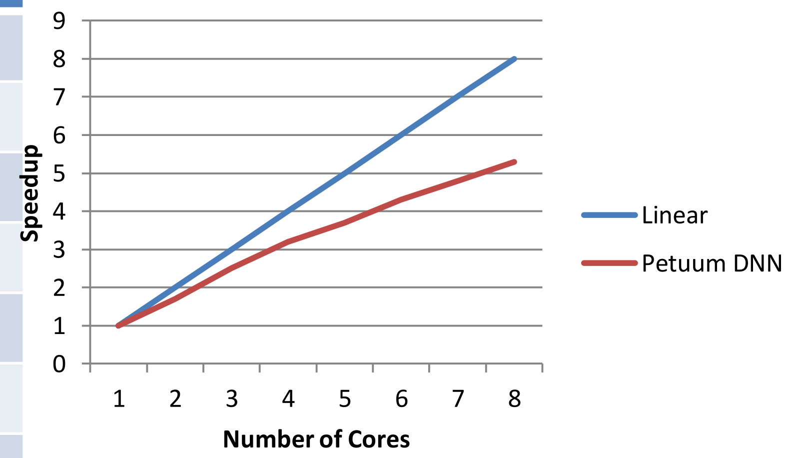
$$(\tilde{w}_t^* \xrightarrow{P} w^*)$$

- In case of model distributed DNN we divided the DNN vertically such that a single layer is distributed across processors

Distributed DNN: (preliminary)

- Application: phoneme classification in speech recognition.
- Dataset: TIMIT dataset with 1M samples.
- Network configuration: input layer with 440 units, output layer with 1993 units, six hidden layers with 2048 units in each layer

Methods	PER
Conditional Random Field [1]	34.8%
Large-Margin GMM [2]	33%
CD-HMM [3]	27.3%
Recurrent Neural Nets [4]	26.1%
Deep Belief Network [5]	23.0%
Petuum DNN (Data Partition)	24.95%
Petuum DNN (Model Partition)	25.12%



Conclusion

- In GM: lots of efforts are directed to improving inference accuracy and convergence speed
  - An advanced tutorial would survey dozen's of inference algorithms/theories, but few use cases on empirical tasks
- In DL: most effort is directed to comparing different architectures and gate functions (based on empirical performance on a downstream task)
  - An advanced tutorial typically consist of a list of all designs of nets, many use cases, but a single name of algorithm: back prop of SGD
- The two fields are similar at the beginning (energy, structure, etc.), and soon diverge to their own signature pipelines
- A convergence might be necessary and fruitful