Large Scale Data Analysis Using Deep Learning

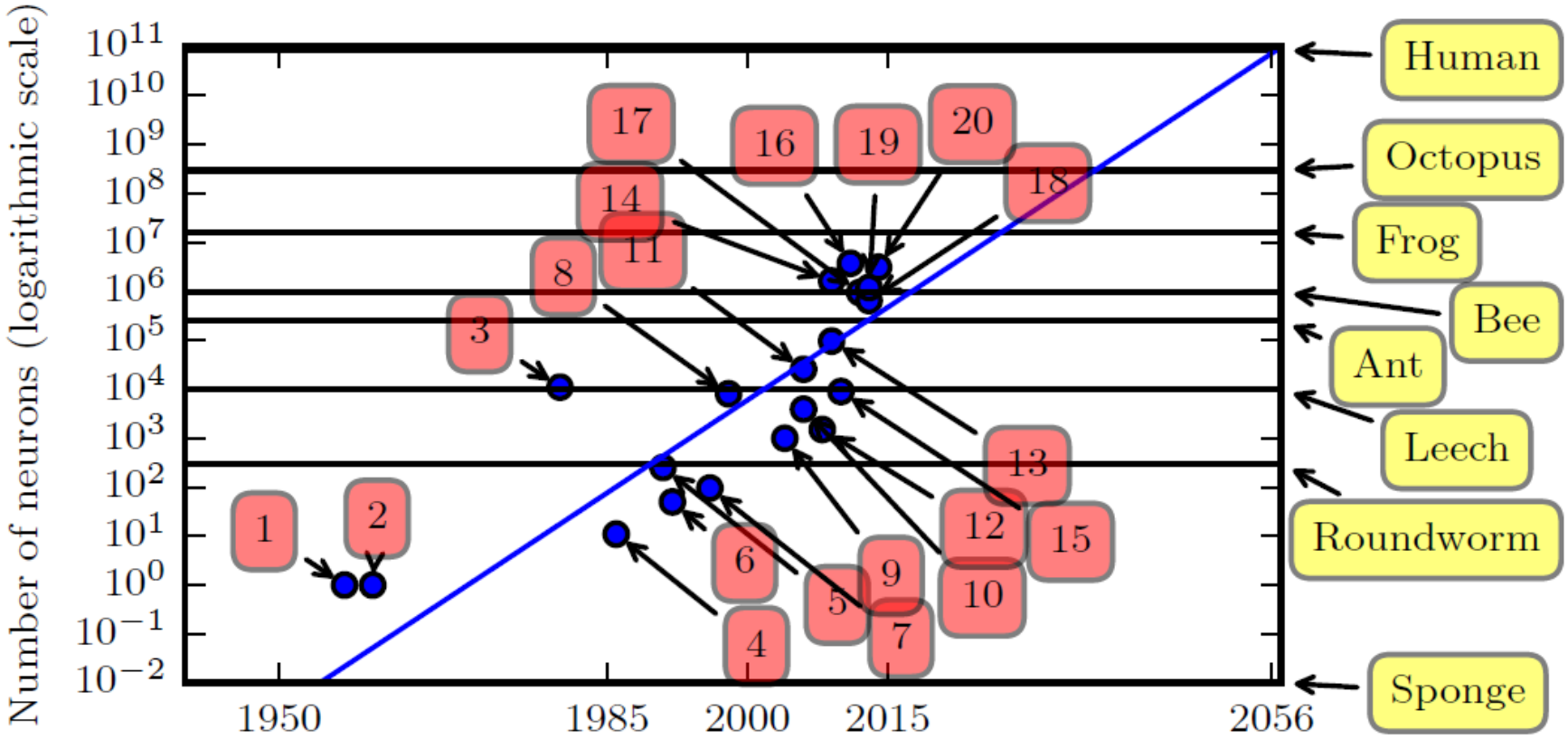
Applications

U Kang
Seoul National University

In This Lecture

- Core technology for large scale deep learning
- Applications
  - Computer Vision
  - Speech
  - Natural Language Processing

Large Scale Deep Learning



Fast Implementations

- CPU
  - Exploit fixed point arithmetic in CPU families for a speedup
  - Cache-friendly implementations

- GPU
  - High memory bandwidth
  - No or little cache
    - It can actually be faster to compute the same value twice, rather than compute it once and read it back from memory
  - Inherently multi-threaded, and different threads must be coordinated with each other carefully

Distributed Implementations

- Multi-GPU
- Multi-machine
  - Model parallelism: multiple machines work together on a single data point, with each machine running a different part of the model
  - Data parallelism: each input example is processed by a separate machine
    - Trivial at test time
    - Asynchronous SGD (or lock-free SGD) at train time
      - Several processor cores share the memory representing the parameters
      - Each core reads parameters without a lock, then computes a gradient, then increments the parameters without a lock
      - This reduces the average amount of improvement that each gradient descent step yields, because some of the cores overwrite each other's progress
      - But the increased rate of production steps causes the learning process to be faster overall
      - Parameter server: manages parameters in multiple machines

Model Compression

- In many commercial applications, it is crucial that the time and memory cost of running inference is small
  - It is accepted that the cost of training is large
- Model compression: replace the original, expensive model with a smaller model that requires less memory and runtime to store and evaluate

Model Compression

- Large models often have lower test error
  - Very large model trained with dropout
  - Ensemble of many models
- Want small model for low resource use at test time
- Train a small model to mimic the learned larger model f
  - Generate a training set containing infinitely many examples, by applying f to randomly sampled points x
  - We then train the new, smaller model to match f(x) on these points
  - It is best to sample the new x points from a distribution resembling the actual test inputs; this can be done by corrupting training examples, or by drawing points from a generative model trained on the original training set
  - Obtains better test error than directly training a small model

Dynamic Structure

- Dynamic structure in deep learning systems
  - A strategy for accelerating deep learning systems
  - Many neural networks: determine which subset of them should be run on a given input
  - Individual neural network: determine which subset of features (hidden units) to compute given information from the input; also called conditional computation
  - Major obstacle to using dynamic structure: decreased degree of parallelism that results from the system following different code branches for different input

- Examples
  - Cascade
  - Hard mixture of experts

Dynamic Structure

- Cascade of classifiers
  - Can be applied when the goal is to detect the presence of a rare object
  - To know for sure the object is present, we must use a complex classifier with high capacity
  - However, we can usually use much less computation to reject inputs as not containing the object
  - We can train a sequence of classifiers
  - The first classifier has low capacity, and has high recall; the final classifier has high precision
  - At test time, we run inference by running the classifiers in a sequence, abandoning any example as soon as any one element in the cascade rejects it

Dynamic Structure

- Hard mixture of experts
  - One can use a neural network (called the gater) to select which one out of several expert networks will be used to compute the output, given the current input
  - Mixture of experts: the gater outputs a set of probabilities or weights, one per expert, and the final output is obtained by the weighted combination of the output of the experts
  - Hard mixture of experts: a single expert is chosen by the gater

Specialized Hardware Implementations

- ASICs (Application-Specific Integrated Circuit)
- FPGA (Field Programmable Gated Array)
- Tensor Processing Unit (TPU) by Google

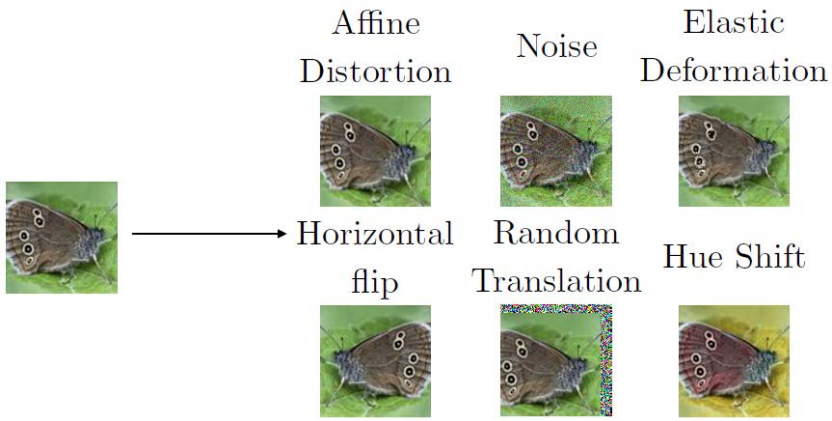
- Algorithms with lower precision
  - It has long been known that it is possible to use less precision in deep network
  - Recent works on low-precision implementation of backprop-based neural nets suggest that 8 and 16 bits of precision can suffice for training deep network
  - More precision is required during training than at inference time
  - Fixed point representation can save bits per number
    - Reduces hardware surface area, power requirements, and computing time

Computer Vision (CV)

- One of the most active research areas for deep learning applications
- Most deep learning for CV is used for object recognition or detection of some form
  - Reporting which object is present in an image
  - Annotating an image with bounding boxes around each object
  - Transcribing a sequence of symbols from an image
  - Labeling each pixel in an image with the identity of the object it belongs to

Computer Vision (CV)

- Preprocessing for CV
  - CV usually requires very little preprocessing
  - Standardizing pixel values
    - The only strictly necessary preprocessing is to standardize pixel values to be in [0, 1] or [-1, 1]
  - Scaling images
    - Many CV architectures require images of a standard size, so images must be cropped or scaled to fit that size
    - But, some convolutional models automatically scales images
  - Dataset augmentation
    - Reduce generalization errors



Speech Recognition

- 1980s ~ 2009
  - State of the art: GMM-HMM model
  - GMM (Gaussian mixture model): models the association between acoustic features and phonemes
  - HMM: models the sequences of phonemes

- 2009~
  - Use neural network to replace GMMs for the task of associating acoustic features to phonemes
  - Ongoing task: building an end-to-end deep learning speech recognition system that completely removes the HMM

Natural Language Processing

- The use of human languages by a computer
  - E.g., Machine translation, dialogue system, etc.
- Many NLP applications are based on language models that define a probability distribution over sequences of words
- Because the total number of possible words is so large, word-based language models must operate on an extremely high-dimensional and sparse discrete space; several strategies have been developed to make efficient models
- Models
  - N-gram
  - Neural language model

N-gram Model

- N-grams language model
  - N-gram: a sequence of n tokens (typically, words)
  - Models based on n-grams define the conditional probability of the n-th token given the preceding n-1 tokens
    - Assumes that the probability depends only on the n-1 tokens
  - The model uses products of these conditional distributions to define the probability distribution over longer sequences

$$P(x_1, \ldots, x_\tau) = P(x_1, \ldots, x_{n-1}) \prod_{t=n}^{\tau} P(x_t \mid x_{t-n+1}, \ldots, x_{t-1})$$

  - Special names: unigram (n=1), bigram (n=2), trigram (n=3)
  - Training n-gram models is straightforward because the maximum likelihood estimate can be computed simply by counting

N-gram Model

- Trigram example: compute the probability of the sentence “THE DOG RAN AWAY”
  - P(THE DOG RAN AWAY) = P(THE DOG RAN) P(AWAY | THE DOG RAN)
    = P(THE DOG RAN) P(AWAY| DOG RAN)
    = P(THE DOG RAN) P(DOG RAN AWAY) / P(DOG RAN)

- Problem of maximum likelihood for n-gram: sparseness
  - Smoothing: add non-zero probability mass to all of the possible symbol values
  - Mixture model containing higher-order and lower-order n-gram models, with the high-order models providing more capacity and the lower-order models being more likely to avoid counts of zero
    - Back-off methods: look-up the lower-order n-grams if the frequency of the context $x_{t-1}, \ldots, x_{t-n+1}$ is too small to use the higher-order model

Word Embeddings in Neural Language Models

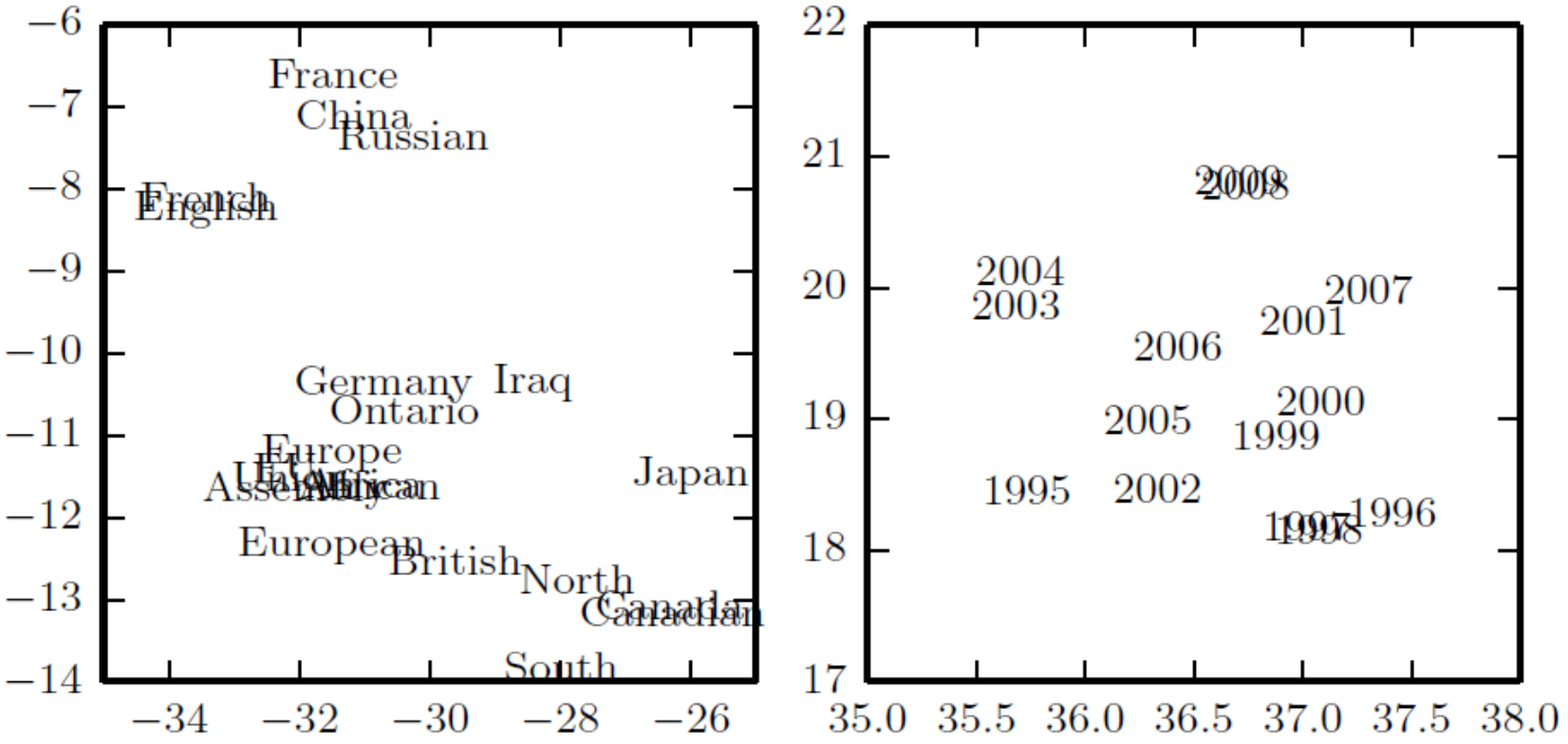
- Neural language models
  - Learns distributed representation of words (also known as word embeddings)
  - In the embedding space, words that frequently appear in similar contexts (or any pair of words sharing some features learned by the model) are close to each other. This often results in words with similar meanings being neighbors
  - Skipgram model (word2vec)
    - Given a sequence of training words $w_1, w_2, \ldots, w_T$,

      Maximize $\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$

      where $p(w_O|w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{|V|}\exp({v'_w}^T v_{w_I})}$

Word Embeddings in Neural Language Models

- Word2vec



High-Dimensional Output Layers for Large Vocabularies

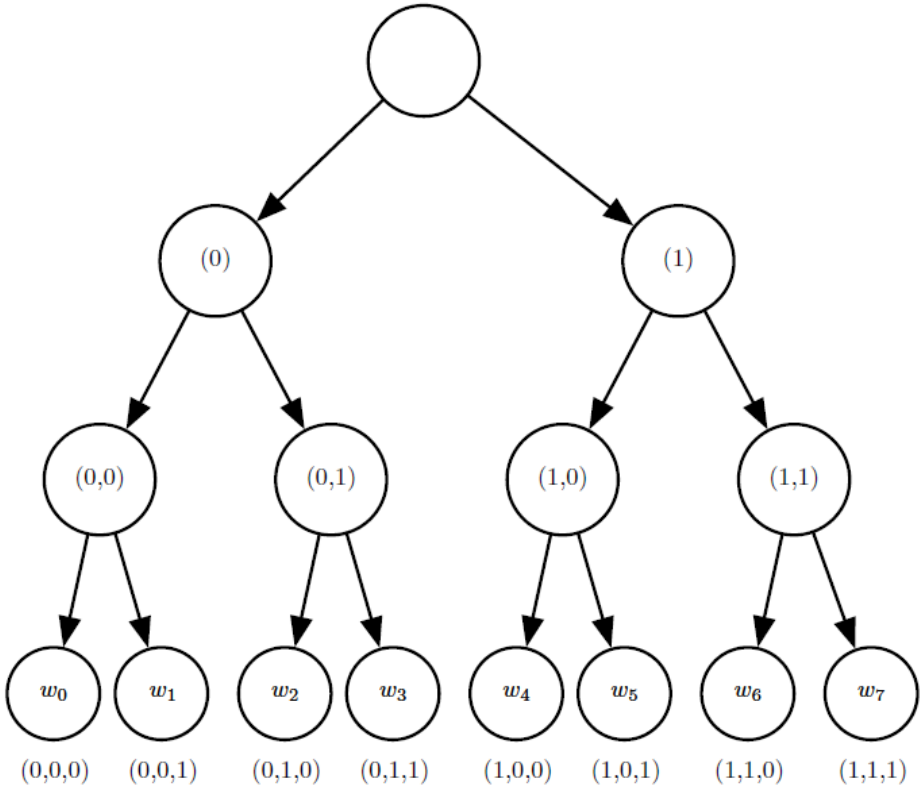
- In many natural language applications, we want our model to produce words as the fundamental unit of the output
- The vocabulary V often contains hundreds of thousands of words
- Assume h is the top hidden layer used to predict the output probabilities y
  - $a_i = b_i + \sum_j W_{ij} h_j$
  - $y_i = \frac{e^{a_i}}{\sum_{i'=1}^{|V|} e^{a_{i'}}}$
- To compute y_i, we need to compute |V| terms in the denominator!

High-Dimensional Output Layers for Large Vocabularies

- Solutions of the high-dimensional output problem
  - Hierarchical softmax
  - Negative sampling

Hierarchical Softmax

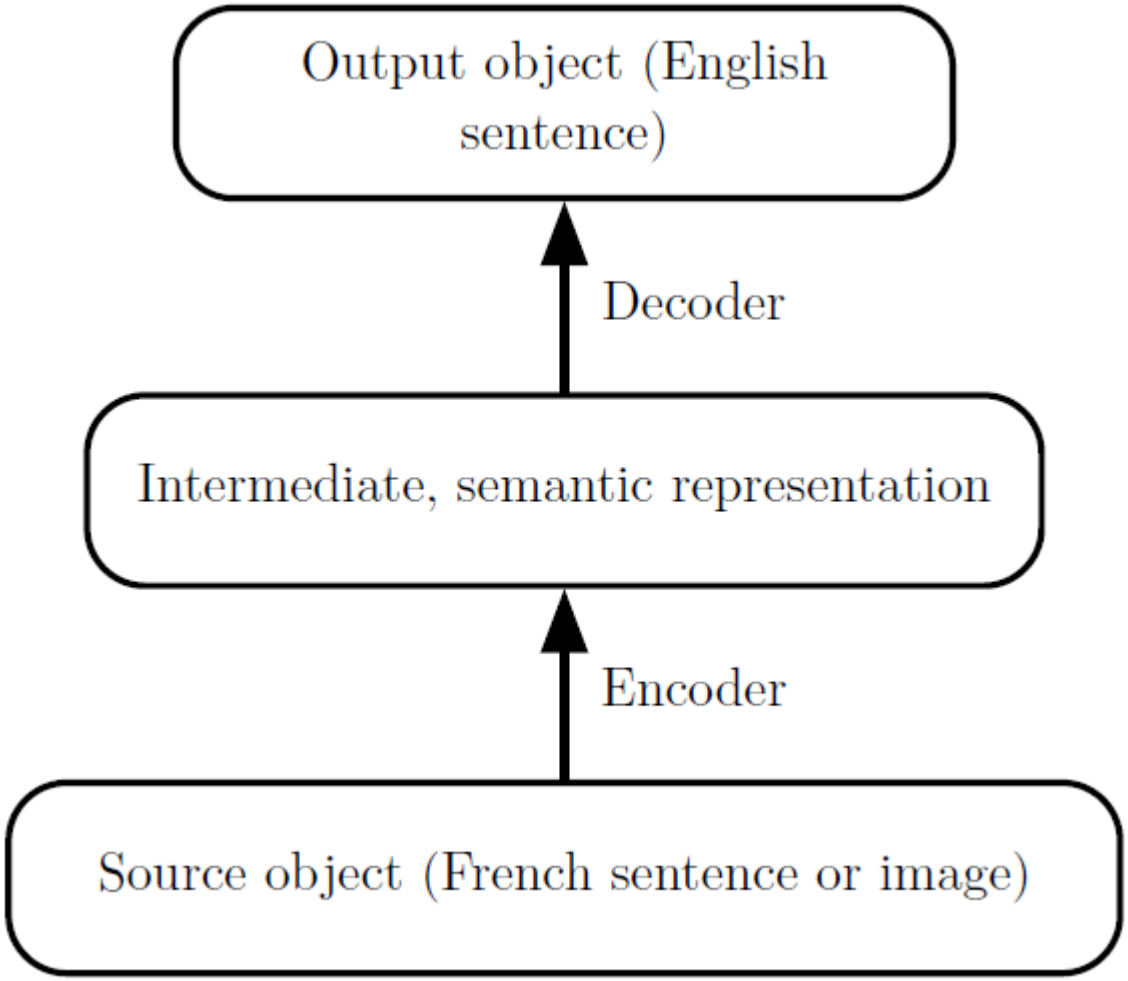
- Decompose probabilities hierarchically
- Instead of running a computations |V| times, do it only log |V| times
- $P(y = w_4) = P(b_0 = 1, b_1 = 0, b_2 = 0)$
  $= P(b_0 = 1)P(b_1 = 0|b_0 = 1)P(b_2 = 0|b_0 = 1, b_1 = 0)$



Negative Sampling

- In word2vec, our goal is to maximize $p(w_O|w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{|V|} \exp({v'_w}^T v_{w_I})}$
- Evaluating the denominator requires too much computation
- Negative sampling
  - Maximize a bit different, but related objective
  - New objective: $\log \sigma({v'_{w_O}}^T v_{w_I}) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^T v_{w_I}) \right]$
  - Intuition: maximize ${v'_{w_O}}^T v_{w_I}$ for similar words w_O and w_I, but minimize ${v'_{w_i}}^T v_{w_I}$ for dissimilar words w_i and w_I

Neural Machine Translation



What you need to know

- Core technology for large scale deep learning
  - Fast implementation
  - Distributed implementation
  - Model compression
  - Dynamic structure
  - Specialized hardware
- Applications
  - Computer Vision: light preprocessing needed
  - Speech
  - Natural Language Processing: efficient method for high-dimensional output

Questions?