Variational Learning and Variational Inference

Ian Goodfellow

What is variational learning?

- Some models are hard to train because it is hard to compute the probability distribution over the hidden units given the visible units
- Instead of computing that distribution, we can compute a simpler one
- Finding the best simple distribution is a *calculus of variations* problem

Comparison to other ideas

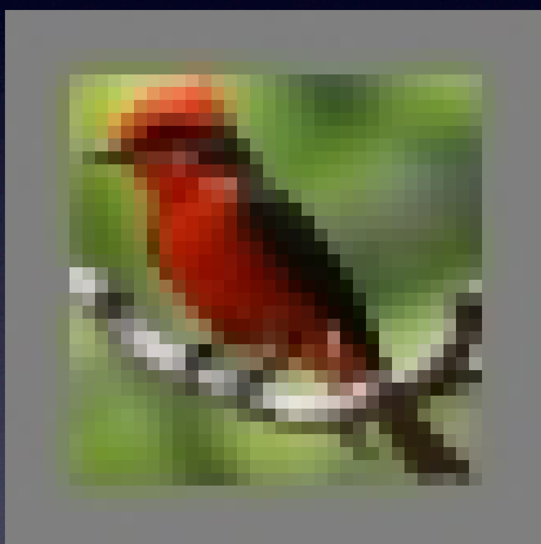
- RBMs are hard to train because the partition function is hard to compute. That's a different kind of difficulty.
- Some approximate learning techniques give you a stochastic but unbiased estimate of the gradient.
- With variational learning, there's no stochasticity, but there is bias. You optimize a different objective function, exactly.

Example: Binary Sparse Coding

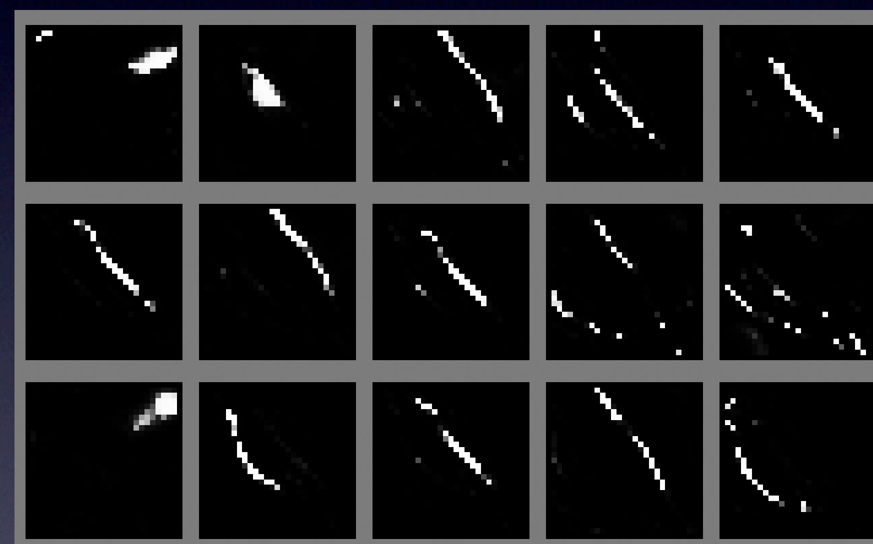
- Let's make a simple model of images
- Suppose we have an image, vector v
- Suppose we think images are made by adding up a bunch of edges, the columns of W
- Suppose we choose each edge independently to include in the image

Example

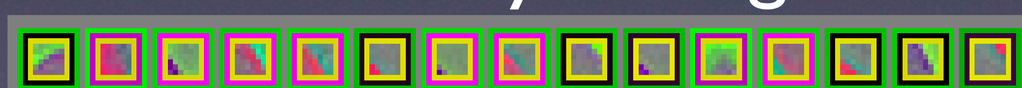
Input image



Map of where to add each



Dictionary of edges



Probabilistic definition

If h_i is 1, the edge is included in the image.
Choose the edges independently of each other:
$p(h_i = 1) = \sigma(b_i)$

We can add up the edges with a matrix multiply:

$$Wh$$

To get a smooth distribution over images v, we add some Gaussian noise:

$$p(v \mid h) = \mathcal{N}(v \mid Wh, I)$$

Energy-Based Model

By multiplying all of the $p(h_i)$ and $p(v \mid h)$ together we get:

$$p(h, v) = \Pi_i \frac{\exp(b_i h_i)}{1 + \exp(b_i)} \Pi_j \sqrt{\frac{1}{2\pi}} \exp\left(-\frac{1}{2}\|v - Wh\|_2^2\right)$$

$$E(v, h) = -b^T h + \frac{1}{2}\|v - Wh\|_2^2$$

$$Z = \Pi_i \sigma(-b_i) \Pi_j \sqrt{2\pi}$$

Maximum likelihood

To train the model, we want to maximize the log likelihood.

We do this by following the derivatives of $\log p(v)$:

$$\frac{d}{d\theta} \log p(v) = \frac{d}{d\theta} \log \sum_h p(h, v)$$

$$= \frac{d}{d\theta} \log \sum_h \frac{1}{Z} \exp\left(-E(h, v)\right)$$

$$= \frac{d}{d\theta} \log \frac{1}{Z} \sum_h \exp\left(-E(h, v)\right)$$

$$= \frac{d}{d\theta} \left[\log \sum_h \exp\left(-E(h, v)\right) - \log Z \right]$$

This is exactly the same as training an RBM, but with a new $E(h, v)$ and Z.

Negative phase

- The negative phase was the only hard part of training the RBM.
- For the RBM, Z is intractable, and we approximate the gradients of log Z by sampling
- For binary sparse coding, it is easy!
- Z is tractable, and so are the derivatives of log Z

Negative phase

$$Z = \Pi_i \sigma(-b_i) \Pi_j \sqrt{2\pi}$$

so

$$\log Z = \sum_i - \log(1 + \exp(b_i)) - \frac{1}{2} \sum_j \log 2\pi$$

$$\frac{d}{dW} \log Z = 0$$

$$\frac{d}{db} \log Z = -\sigma(b)$$

Easy and cheap!

Positive Phase

The positive phase is expensive:

$$\frac{d}{d\theta} \log \sum_h \exp\left(-E(h,v)\right)$$

$$= -\mathbb{E}_{h \sim p(h|v)} \frac{d}{d\theta} E(h,v)$$

The expectation requires computing $p(h \mid v)$.

For RBMs, this is easy. But for binary sparse

coding, even drawing samples is hard!

p(h|v) is complicated

- Hidden units must compete to explain the input
- If every hidden unit with a positive dot product turns on, our reconstruction Wh could actually overshoot v

$$p(h \mid v) \propto \exp\left(b^T h - \frac{1}{2}\|v - Wh\|_2^2\right)$$

$$= \exp\left(b^T h - \frac{1}{2}v^2 v + v^T W h - \frac{1}{2}h^T W^T W h\right)$$

$$\propto \exp\left(b^T h + v^T W h - \frac{1}{2}h^T W^T W h\right)$$

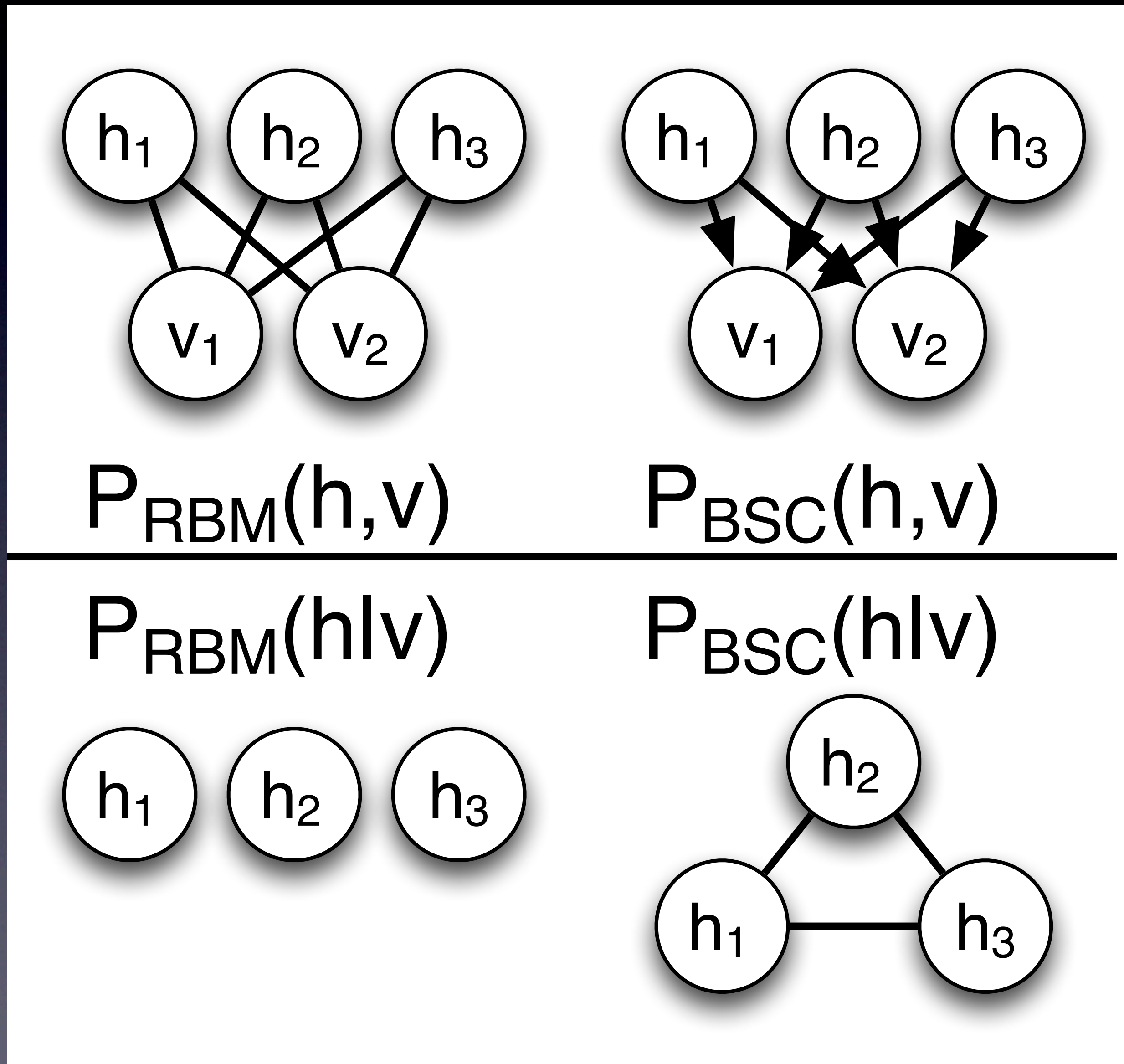
$$= \Pi_i \frac{\exp(b_i h_i)\exp(v^T W_{:i} h_i)}{\Pi_j \exp(\frac{1}{2}W_{:i}^T W_{:j} h_i h_j)}$$

Every h_i interacts with every h_j!

This means we can't normalize the distribution

over each h_i separately from the others.

Comparison to RBM



Let's simplify things

- $p(h|v)$ is just plain too hard to work with
- Let's make a new distribution $q(h)$
- We want $q(h)$ to be simple. It should be cheap to compute expectations over $q(h)$
- We want $q(h)$ to be close to $p(h|v)$ somehow

Enforcing simplicity

- One way to make sure that q(h) is simple is to constrain it to factorize:

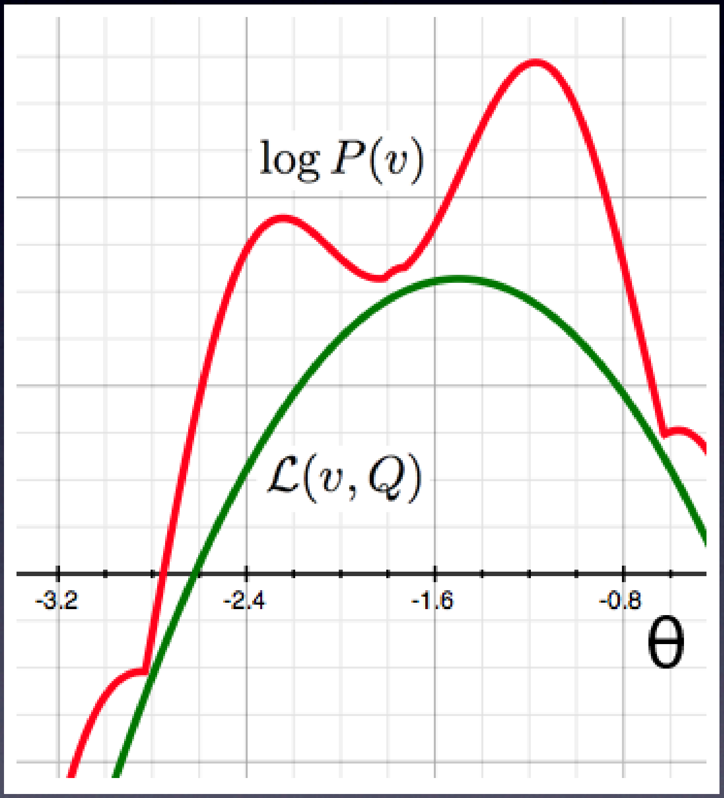
$$q(h) = \Pi_i q(h_i)$$

- Using this particular constraint for variational learning is usually called the *mean field approximation*.

- This makes q(h) have a graph with no edges. Constraining q(h) to have some specific graph is called a *structured variational approximation.*

Variational lower bound

What if instead of maximizing log p(v) we maximize a lower bound on it?



L(v,Q)=log p(v)-KL(q(h)||p(h|v)) <= log p(v)

because the KL divergence is never negative.

When KL(q(h)||p(h|v)) is small, q(h) resembles p(h|v) *and* the bound is tight!

Properties of L

- L(v,q)=log p(v) - KL(q(h) || p(h|v)) seems like something arbitrary, that I just picked because it is obviously <= log p(v)
- When p=q, KL=0 so L=log p(v)
- Turns out to be tractable
- Depends only on q(h), not p(h|v)
- Only one term depends on the model parameters:

$$\mathbb{E}_{h \sim q(h)} \log p(h, v)$$

The variational approach

- *Variational inference:* Find q(h) by solving

$$q(h) = \mathrm{argmin}_q KL\left(q(h) \| p(h \mid v)\right)$$

$$\text{subject to } q(h) = \Pi_i q(h_i)$$

- *Variational learning:* Alternate between running variational inference to update q and maximizing log p(v) - KL(q(h)||p(h|v))

Binary sparse coding example

- For binary sparse coding, any legal q(h) can be represented as

$$q(h) = \Pi_i q(h_i)$$

where $q(h_i = 1) = \hat{h}_i$

and $\hat{h_i} \in [0, 1]$ is an optimization parameter

Zero gradient solution

We can solve the minimization problem

$$\min_{\hat{h}} KL\left(q(h) \| p(h \mid v)\right)$$

just by algebra, by solving

$$\nabla_{\hat{h}} KL(q(h) \| p(h) \mid v) = 0$$

for $\hat{h}$.

Fixed point equations

- Unfortunately, there is no closed form solution for the point where the whole gradient is zero.
- Instead, we can repeatedly pick one variable and set its gradient to zero, by solving

$$\frac{\partial}{\partial \hat{h}_i} KL\left(q(h) \| p(h \mid v)\right) = 0$$

- Eventually, the whole gradient will be zero.

Fixed point update

- After doing a bunch of calculus and algebra, we get that the fixed point update is

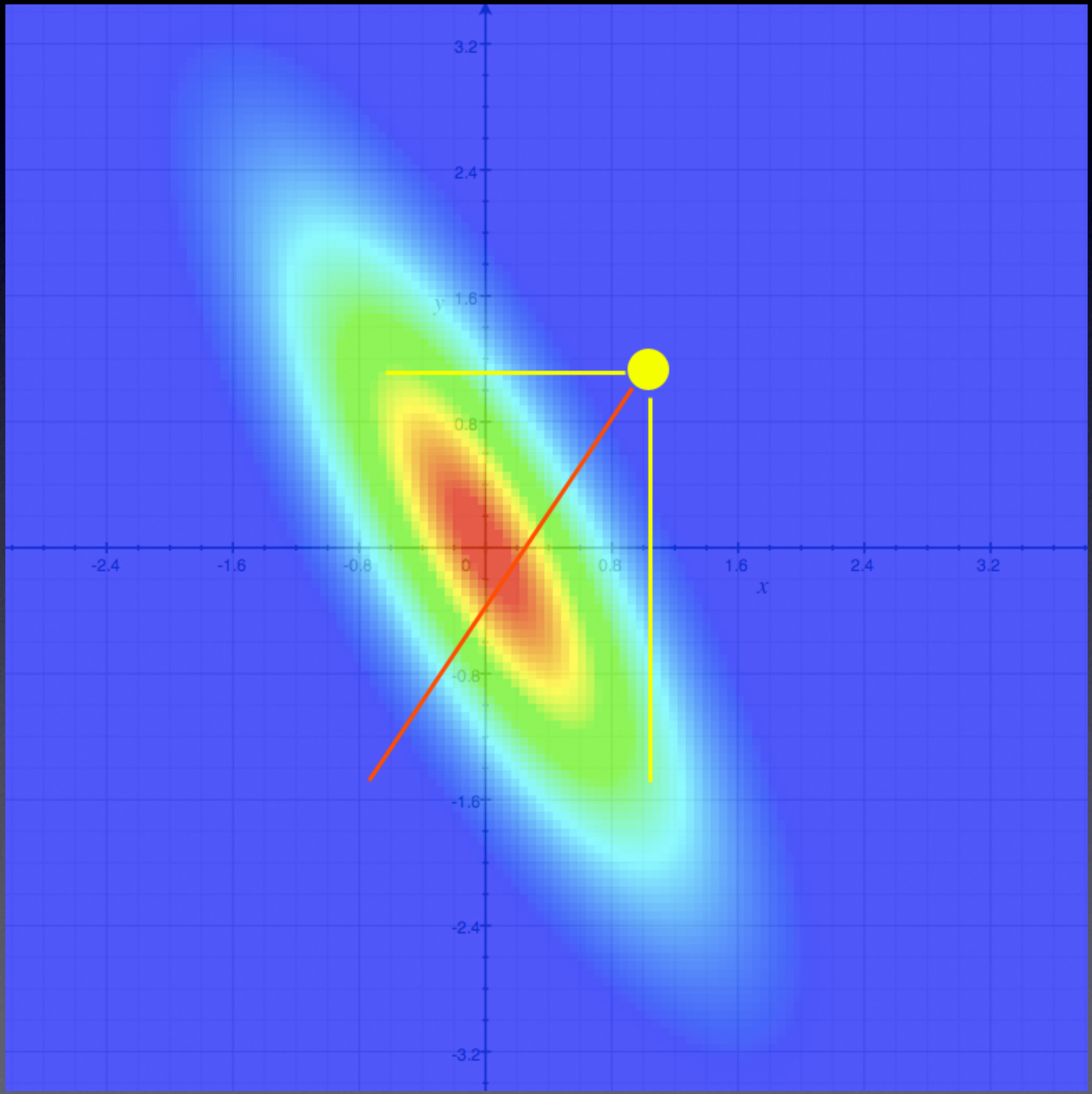
$$\hat{h}_i = \sigma\left(v^T W_{:i} + b_i - \frac{1}{2} W_i^T W_i - \frac{1}{2}\sum_j W_j^T W_i \hat{h}_j\right)$$

- It looks a lot like p(h|v) in an RBM, but now the different hidden units get to inhibit each other.

Parallel updates

- These equations just say how to update one equation at a time
- What if you want to update several?
- Updating each variable to its individually optimal value doesn't reach the global optimum. You have to scale back the step to avoid overshooting.

Overshooting visualization



Diagnosing Variational Model Problems

- Most important technical skill as a researcher, engineer, or consultant is deciding what to try next.
- Probabilistic methods are nice because you can isolate failures of inference from failures of learning or failures of representation
- What are some tests you could do to verify that variational inference is working?

Example Unit Tests

- Fixed point update sets a derivative of the KL to 0
- At convergence, all derivatives are near 0
- The KL decreases across updates
- BSC/S3C: with orthogonal weights, a single update drives the KL to 0 (can't test if the KL is 0, because that involves computing p(v))
- When using damping, monitor the KL after each iteration. This can detect problems with your damping schedule.

Continuous variables

- This was for discrete h, where q(h) can be described by a vector
- What about for continuous h, where q(h) is a function, aka a vector with uncountably infinitely many elements?
- This is where *calculus of variations* comes in

Calculus of variations

- We can define a function f that maps a vector x to some real number f(x)
- Using calculus, we can solve for the x where the gradient is 0 to minimize f(x)
- We can define a *functional* F that maps a function f to some real number F[f]
- Using *calculus of variations*, we can solve for the f that minimizes F[f]

Calculus of variations for variational inference

- In variational inference,
  - $q(h)$ is our function
  - $KL(q(h)||p(h|v))$ is our functional
- We want to solve for the $q(h)$ that minimizes the KL

Euler-Lagrange equations

The Euler-Lagrange equations state that if

$$F[f] = \int G\left(f(x), f'(x), x\right) dx$$

then Fmay be minimized by solving

$$\frac{\partial G}{\partial f} = \frac{d}{dx}\left(\frac{\partial G}{\partial f'}\right)$$

Applications of Euler-Lagrange

- We can use the Euler-Lagrange equation to solve for the minimum of a functional
- When f is a probability distribution, we need to also add a Lagrange multiplier to make sure f is normalized
- You can use this to prove stuff like that the Gaussian distribution has the highest entropy of any distribution with fixed variance v

General Structured Variational Inference

- Using Euler-Lagrange and Lagrange multipliers to enforce that q(h) integrates to 1, we can solve the minimization for general p(h,s) and partitions of q:

If we split h into disjoint groups, then the group with indices in set S has

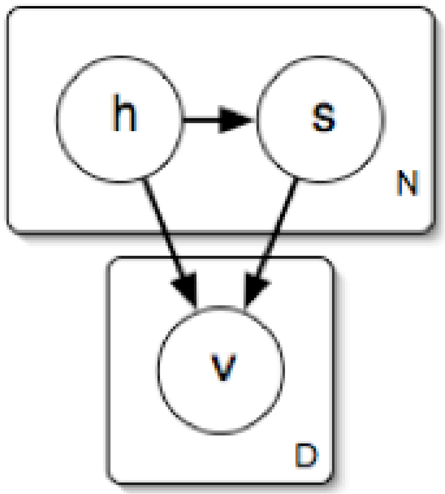
$$q(h_S) \propto \exp\left(\mathbb{E}_{h_{-S}\sim q} \log p(h, v)\right)$$

where h_{-S} is all of the variables not in this group.

More complicated structures

- You could also imagine making q(h) have a richer structure
  - chain
  - tree
  - any structure for which expectations of log p(h,s) are tractable
- It doesn't have to be a partition. But I don't cover that here.

Example: Spike and Slab Sparse Coding



$$p(h_i = 1) = \sigma(b_i)$$

$$p(s_i \mid h_i) = \mathcal{N}(s_i \mid h_i \mu_i, \alpha_{ii}^{-1})$$

$$p(v_d \mid s, h) = \mathcal{N}(v_d \mid W_{d:}(h \circ s), \beta_{dd}^{-1})$$

Variational Inference for S3C

$$\min_Q D_{KL}(Q(h,s) \| P(h,s \mid v))$$

$$Q(h,s) = \Pi_i Q(h_i, s_i)$$

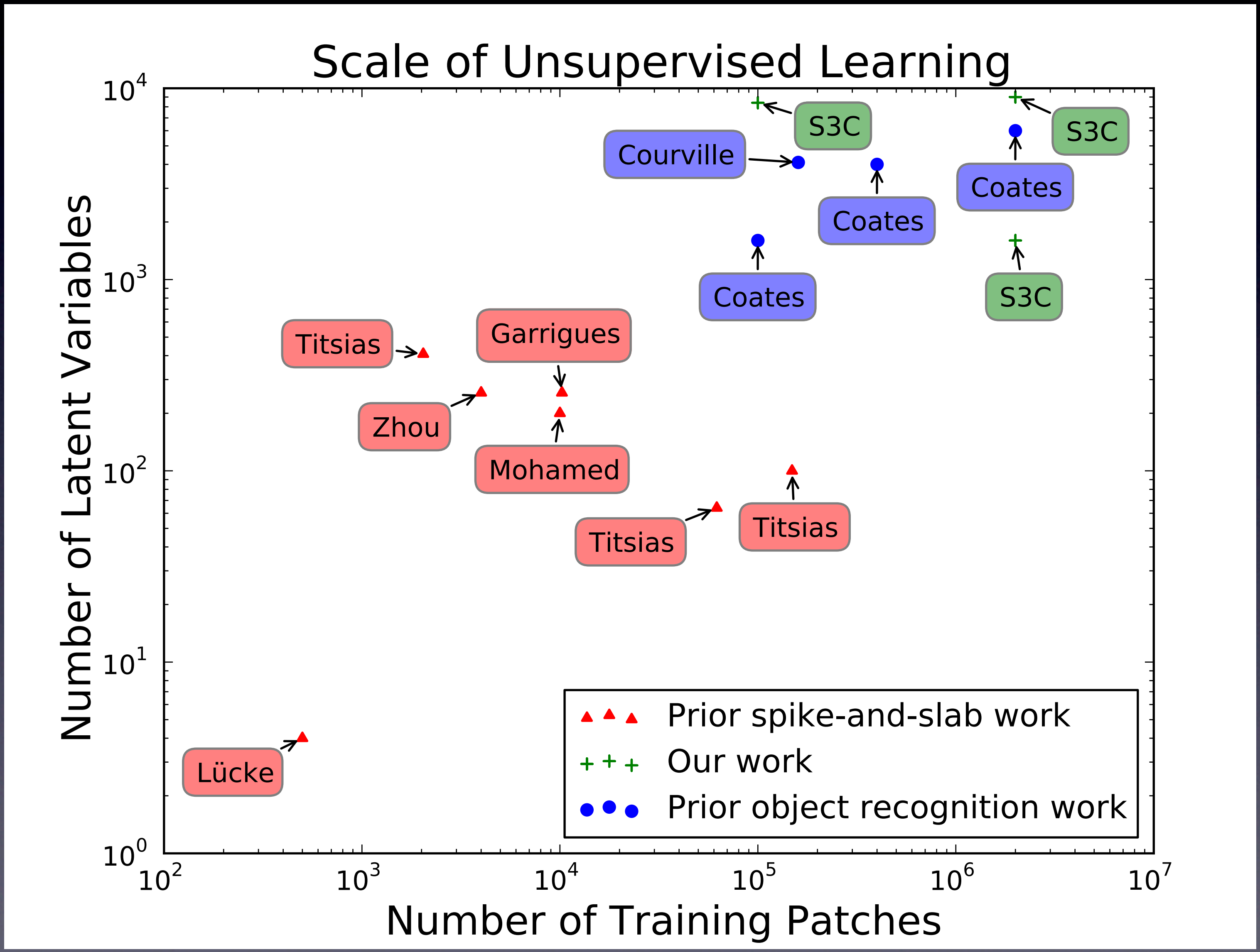
$$\begin{aligned} Q(h_i) &= \hat{h}_i \\ Q(s_i \mid h_i) &= \mathcal{N}(s_i \mid h_i \hat{s}_i,\ (\alpha_i + h_i W_i^T \beta W_i)^{-1}) \end{aligned}$$

- The fact that Q(s) is Gaussian is *not* hand designed. That comes out of the Euler-Lagrange equations! Neat!

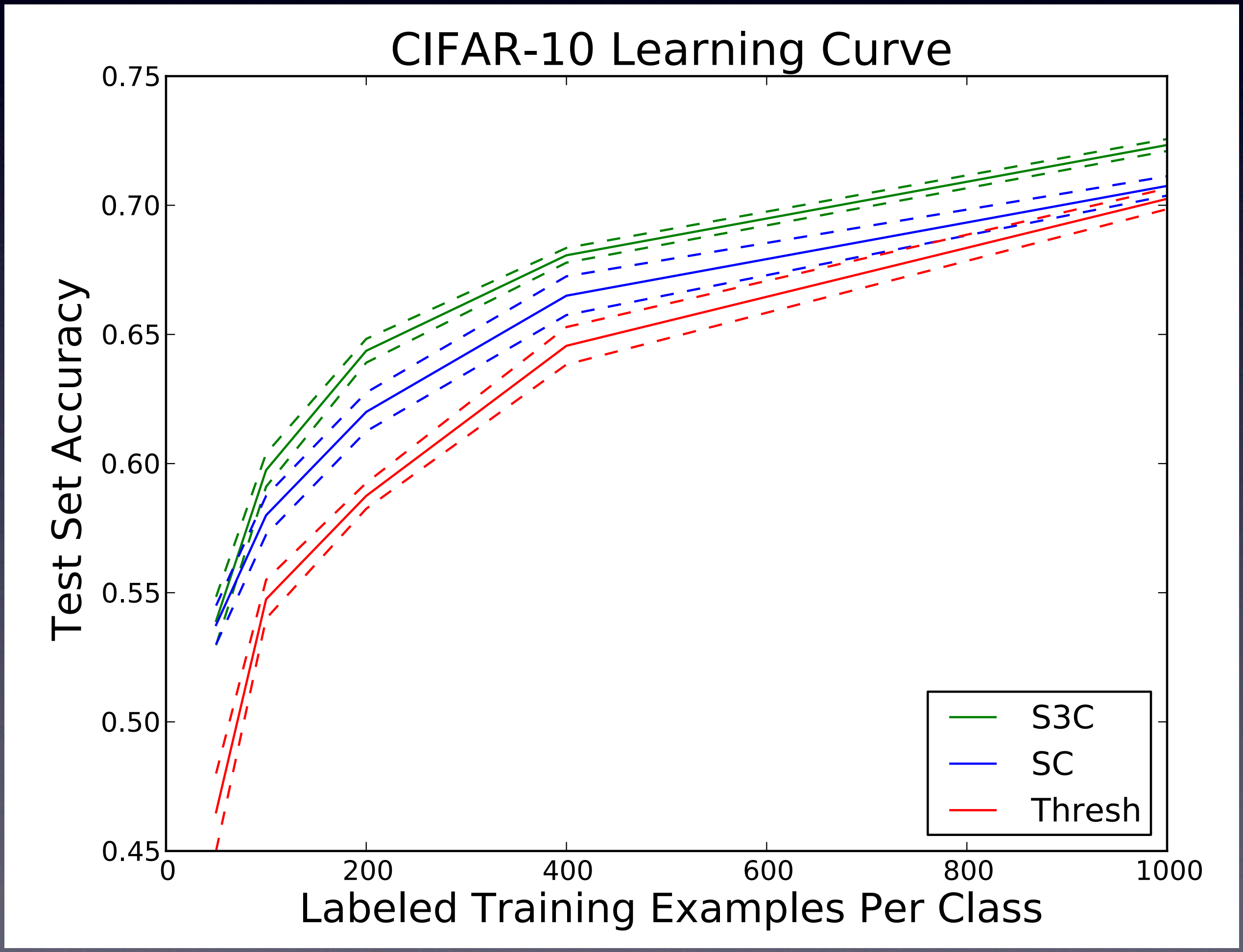
Optimization

- Euler-Lagrange only tells us the functional form of the answer
- We still need to use fixed point iteration to solve for the mean of each variable
- I wrote a paper about a principled way of doing this at medium speed and a hacky way of doing this very fast on GPU last year

Fast optimization lets us do object recognition



Inference helps when labels are scarce



Transfer Learning Challenge

- Won the "NIPS 2011 Workshop on Challenges in Hierarchical Learning: Transfer Learning Challenge"... without using transfer learning
- Train on a large amount of unlabeled data
- Optionally train on a medium amount of labeled data, of other object categories
- Train on just 5-20 examples per category for 10 new categories, and test on those

Further reading

- *Probabilistic Graphical Models: Principals and Techniques* by Daphne Koller and Nir Friedman. Chapter 11
- *Pattern Recognition and Machine Learning* by Christopher M. Bishop, Chapter 10.1 and Appendix D
- "Scaling Spike-and-Slab Models for Unsupervised Feature Learning" Ian J. Goodfellow, Aaron Courville, Yoshua Bengio. To appear in IEEE TPAMI special issue on deep learning. http://www-etud.iro.umontreal.ca/~goodfeli/tpami.pdf