Bayesian Reasoning and Deep Learning

Shakir Mohamed

Google DeepMind

shakirm.com @shakir_za

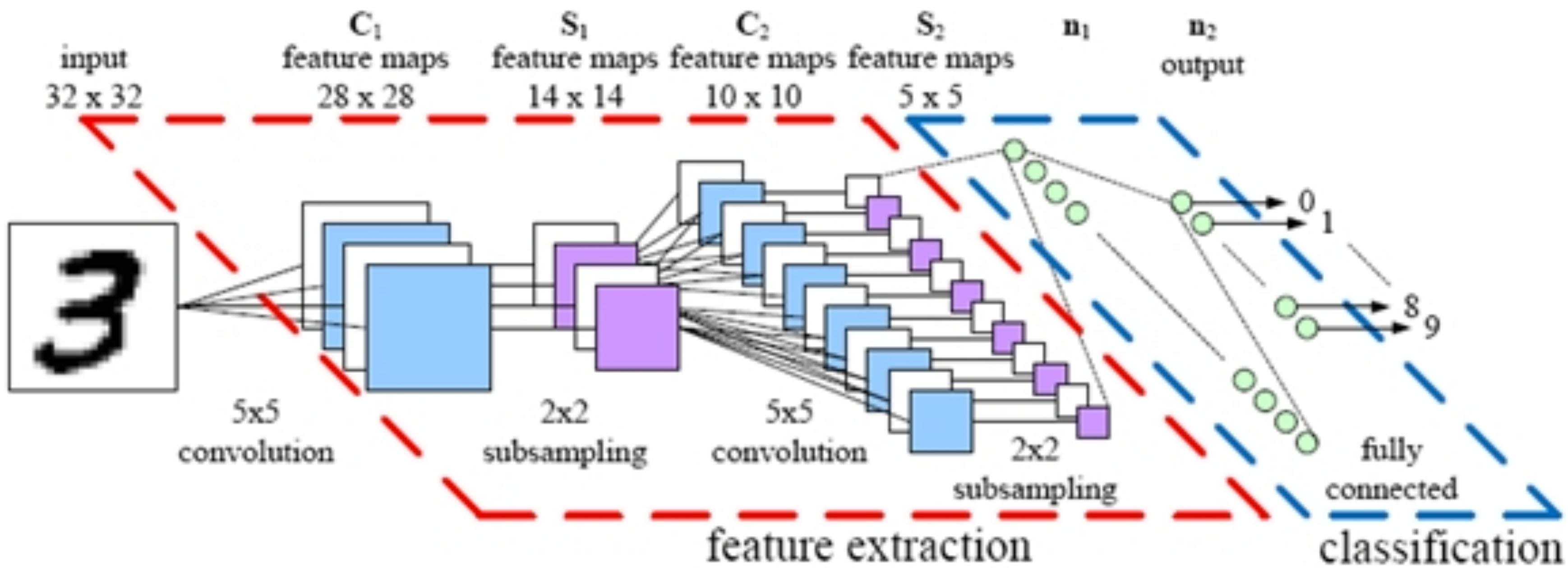
9 October 2015

Abstract

Deep learning and Bayesian machine learning are currently two of the most active areas of machine learning research. *Deep learning* provides a powerful class of models and an easy framework for learning that now provides state-of-the-art methods for applications ranging from image classification to speech recognition. *Bayesian reasoning* provides a powerful approach for information integration, inference and decision making that has established it as the key tool for data-efficient learning, uncertainty quantification and robust model composition that is widely used in applications ranging from information retrieval to large-scale ranking. Each of these research areas has shortcomings that can be effectively addressed by the other, pointing towards a needed convergence of these two areas of machine learning; the complementary aspects of these two research areas is the focus of this talk. Using the tools of auto-encoders and latent variable models, we shall discuss some of the ways in which our machine learning practice is enhanced by combining deep learning with Bayesian reasoning. This is an essential, and ongoing, convergence that will only continue to accelerate and provides some of the most exciting prospects, some of which we shall discuss, for contemporary machine learning research.


Deep Learning
Bayesian Reasoning
Better ML

Deep Learning



A framework for constructing flexible models

+ Rich non-linear models for classification and sequence prediction.

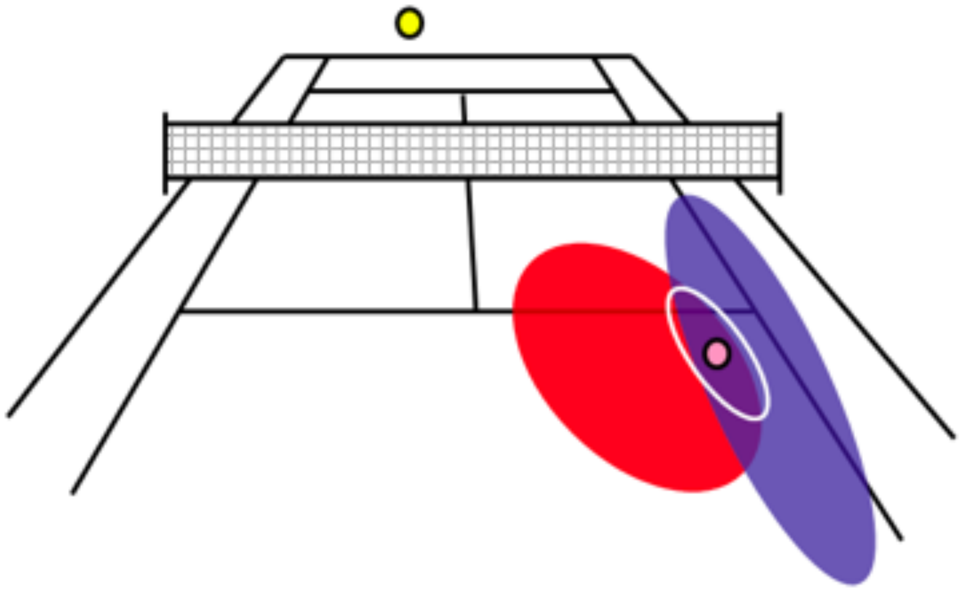
+ Scalable learning using stochastic approximations and conceptually simple.

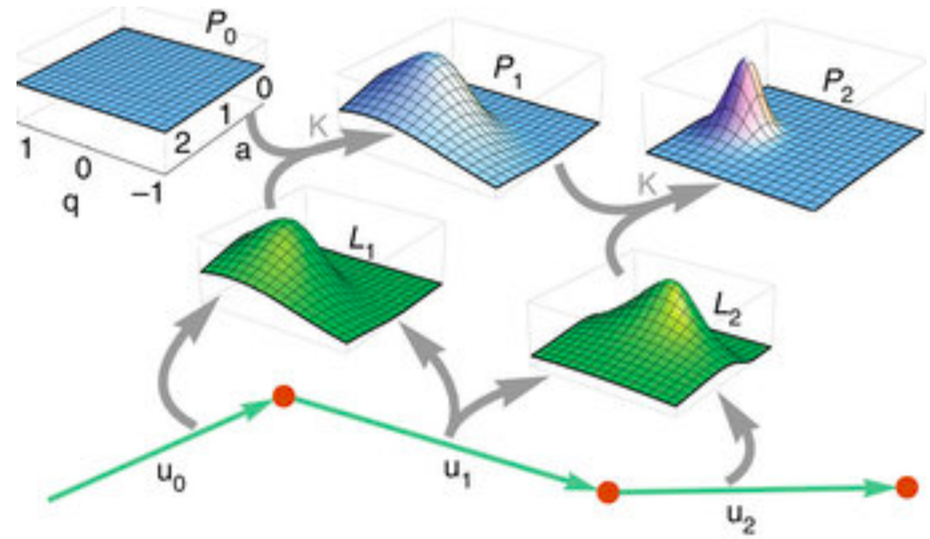
+ Easily composable with other gradient-based methods

- Only point estimates

- Hard to score models, do model selection and complexity penalisation.

Bayesian Reasoning





A framework for inference and decision making

+ Unified framework for model building, inference, prediction and decision making

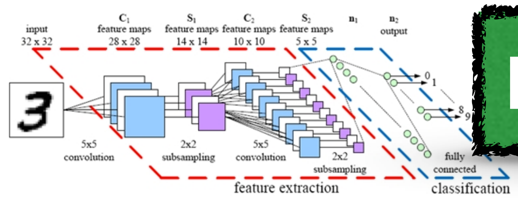
+ Explicit accounting for uncertainty and variability of outcomes

+ Robust to overfitting; tools for model selection and composition.

- Mainly conjugate and linear models

- Potentially intractable inference leading to expensive computation or long simulation times.

Two Streams of Machine Learning



Deep Learning

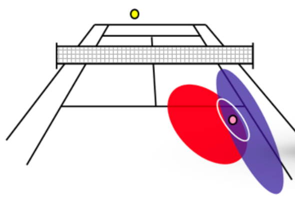
+ Rich non-linear models for classification and sequence prediction.

+ Scalable learning using stochastic approximation and conceptually simple.

+ Easily composable with other gradient-based methods

- Only point estimates

- Hard to score models, do selection and complexity penalisation.



Bayesian Reasoning

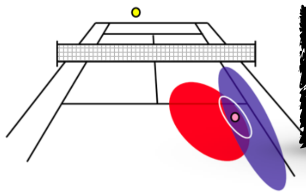
- Mainly conjugate and linear models

- Potentially intractable inference, computationally expensive or long simulation time.

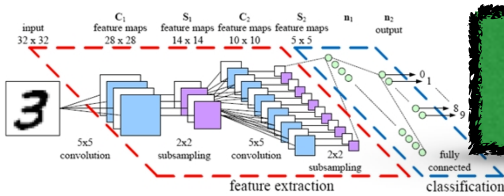
+ Unified framework for model building, inference, prediction and decision making

+ Explicit accounting for uncertainty and variability of outcomes

+ Robust to overfitting; tools for model selection and composition.



Bayesian Reasoning +



Deep Learning

Complementary strengths that we should expect to be successfully combined.

1. **Why is this a good idea?**
   - ✤ Review of deep learning
   - ✤ Limitations of maximum likelihood and MAP estimation
2. **How can we achieve this convergence?**
   - ✤ Case study using auto-encoders and latent variable models
   - ✤ Approximate Bayesian inference
3. **What else can we do?**
   - ✤ Semi-supervised learning, classification, better inference and more.

A (Statistical) Review of Deep Learning

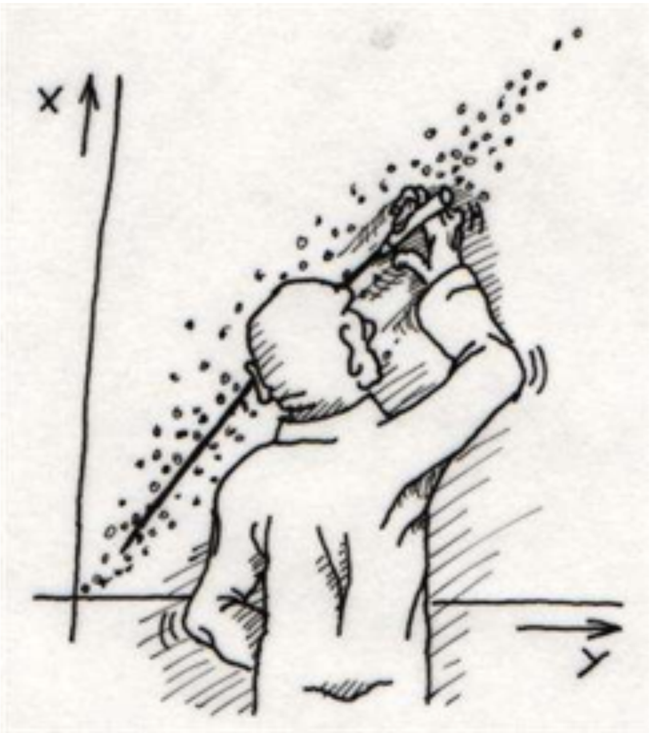
Generalised Linear Regression

$$\eta = \mathbf{w}^\top \mathbf{x} + b$$

$$p(y|\mathbf{x}) = p(y|g(\eta); \theta)$$

- ✦ The basic function can be any linear function, e.g., affine, convolution.
- ✦ *g(.)* is an ***inverse link function*** that we'll refer to as an activation function.

Target	Regression	Link	Inv link	Activation
Real	Linear	Identity	Identity	
Binary	Logistic	Logit $\log\frac{\mu}{1-\mu}$	Sigmoid $\frac{1}{1+\exp(-\eta)}$	Sigmoid
Binary	Probit	Inv Gauss CDF $\Phi^{-1}(\mu)$	Gauss CDF $\Phi(\eta)$	Probit
Binary	Gumbel	Compl. log-log $\log(-\log(\mu))$	Gumbel CDF $e^{-e^{-x}}$	
Binary	Logistic		Hyperbolic Tangent $\tanh(\eta)$	Tanh
Categorical	Multinomial		Multin. Logit $\frac{\eta_i}{\sum_j \eta_j}$	Softmax
Counts	Poisson	$\log(\mu)$	$\exp(\nu)$	
Counts	Poisson	$\sqrt{(\mu)}$	ν^2	
Non-neg.	Gamma	Reciprocal $\frac{1}{\mu}$	$\frac{1}{\nu}$	
Sparse	Tobit		max $\max(0;\nu)$	ReLU
Ordered	Ordinal		Cum. Logit $\sigma(\phi_k - \eta)$	

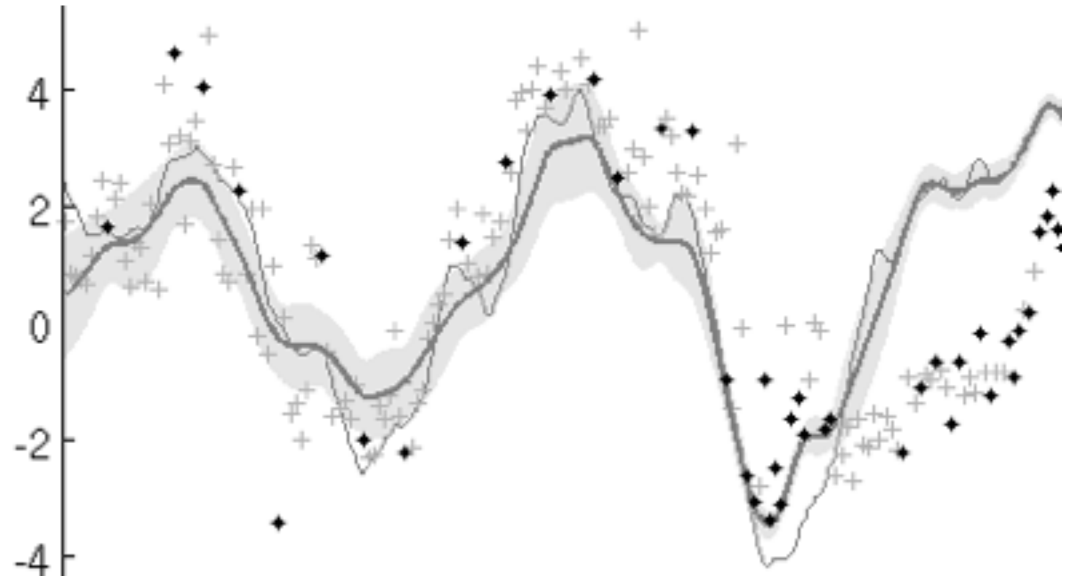


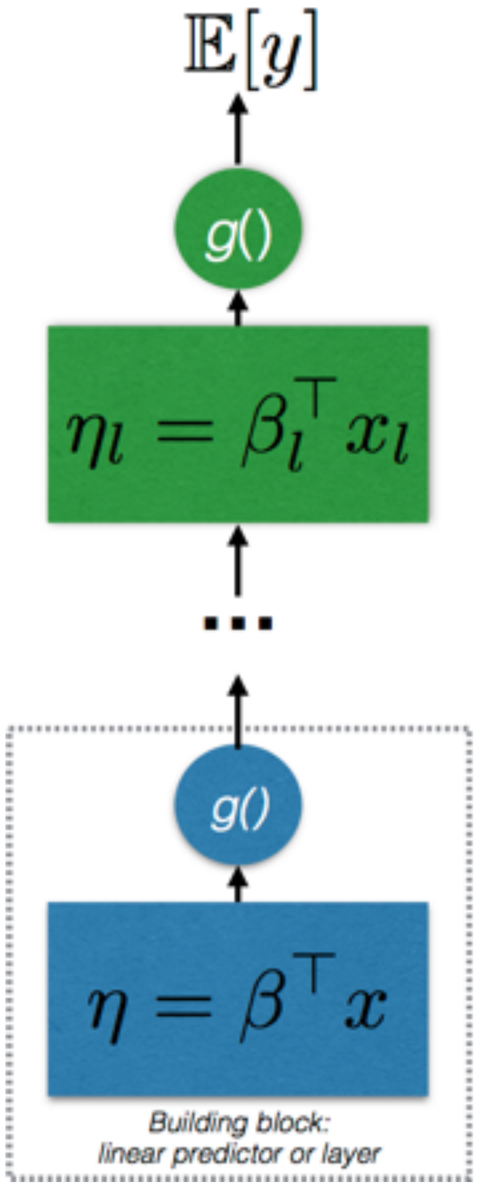
Maximum likelihood estimation

Optimise the negative log-likelihood

$$\mathcal{L} = -\log p(y|g(\eta); \theta)$$

A (Statistical) Review of Deep Learning





Recursive Generalised Linear Regression

- ✦ Recursively compose the basic linear functions.
- ✦ Gives a deep neural network.

$$\mathbb{E}[y] = h_L \circ \ldots \circ h_l \circ h_0(\mathbf{x})$$

A general framework for building non-linear, parametric models

Problem: Overfitting of MLE leading to limited generalisation.

A (Statistical) Review of Deep Learning

Regularisation Strategies for Deep Networks

- ✦ Regularisation is essential to overcome the limitations of maximum likelihood estimation.
- ✦ Regularisation, penalised regression, shrinkage.
- ✦ A wide range of available regularisation techniques:

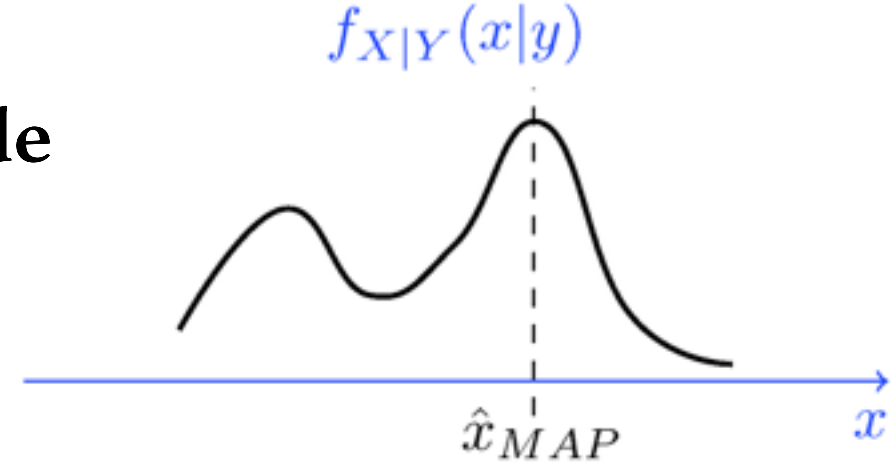
  - Large data sets
  - Input noise/jittering and data augmentation/expansion.
  - L2 /L1 regularisation (Weight decay, Gaussian prior)
  - Binary or Gaussian Dropout
  - Batch normalisation

More robust loss function using MAP estimation instead.

More Robust Learning

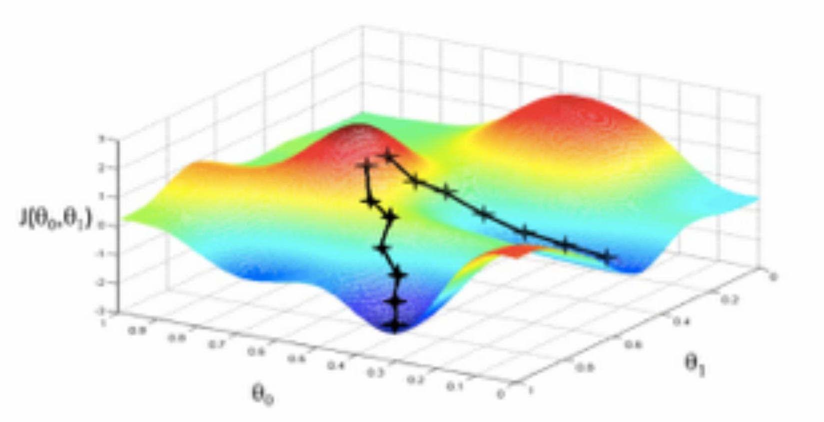
MAP estimators and limitations

- ✦ Power of MAP estimators is that they provide some robustness to overfitting.
- ✦ Creates sensitivities to parameterisation.

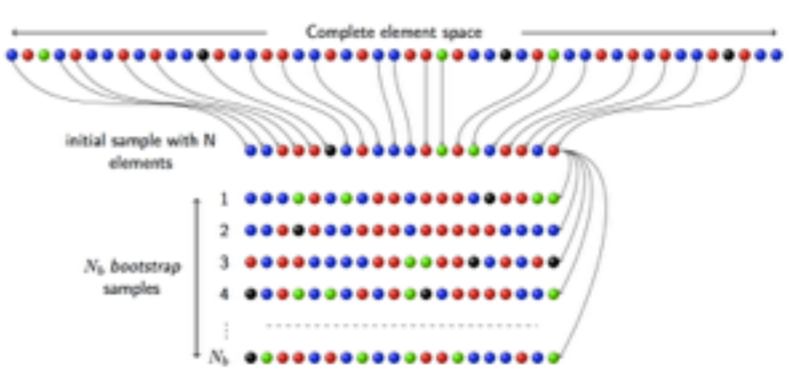


1. Sensitivities affect gradients and can make learning hard

Invariant MAP estimators and exploiting natural gradients, trust region methods and other improved optimisation.



2. Still no way to measure confidence of our model.



Can generate frequentist confidence intervals and bootstrap estimates.

Towards Bayesian Reasoning

Proposed solutions have not fully dealt with the underlying issues.

Issues arise as a consequence of:

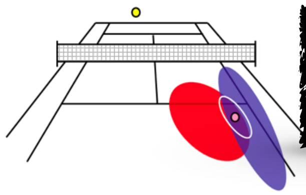
- **Reasoning only about the most likely solution and**
- **Not maintaining knowledge of the underlying variability (and averaging over this).**

Given this powerful model class and invaluable tools for regularisation and optimisation, let us develop a

Pragmatic Bayesian Approach for
Probabilistic Reasoning in Deep Networks.

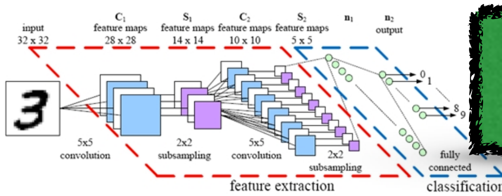
Bayesian reasoning over some, but not all parts of our models (yet).

Outline



Bayesian Reasoning

+

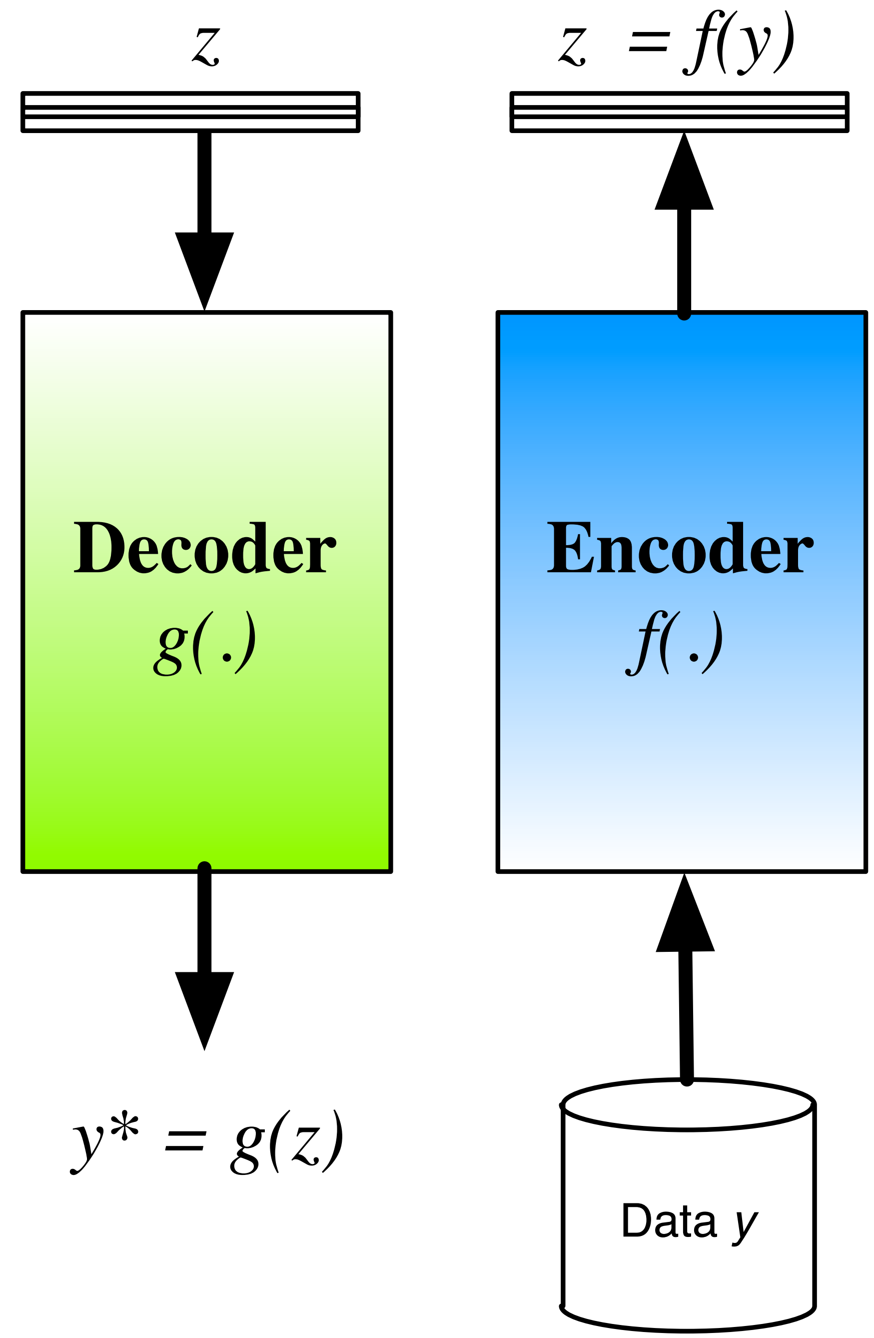


Deep Learning

Complementary strengths that we should expect to be successfully combined.

1. Why is this a good idea?
   - ✤ Review of deep learning
   - ✤ Limitations of maximum likelihood and MAP estimation
2. How can we achieve this convergence?
   - ✤ Case study using auto-encoders and latent variable models
   - ✤ Approximate Bayesian inference
3. What else can we do?
   - ✤ Semi-supervised learning, classification, better inference and more.

Dimensionality Reduction and Auto-encoders



Unsupervised learning and auto-encoders

- **A generic tool for dimensionality reduction and feature extraction.**
- **Minimise reconstruction error using an encoder and a decoder.**

\+ Non-linear dimensionality reduction using deep networks for encoder and decoder.

\+ Easy to implement as a single computational graph and train using SGD

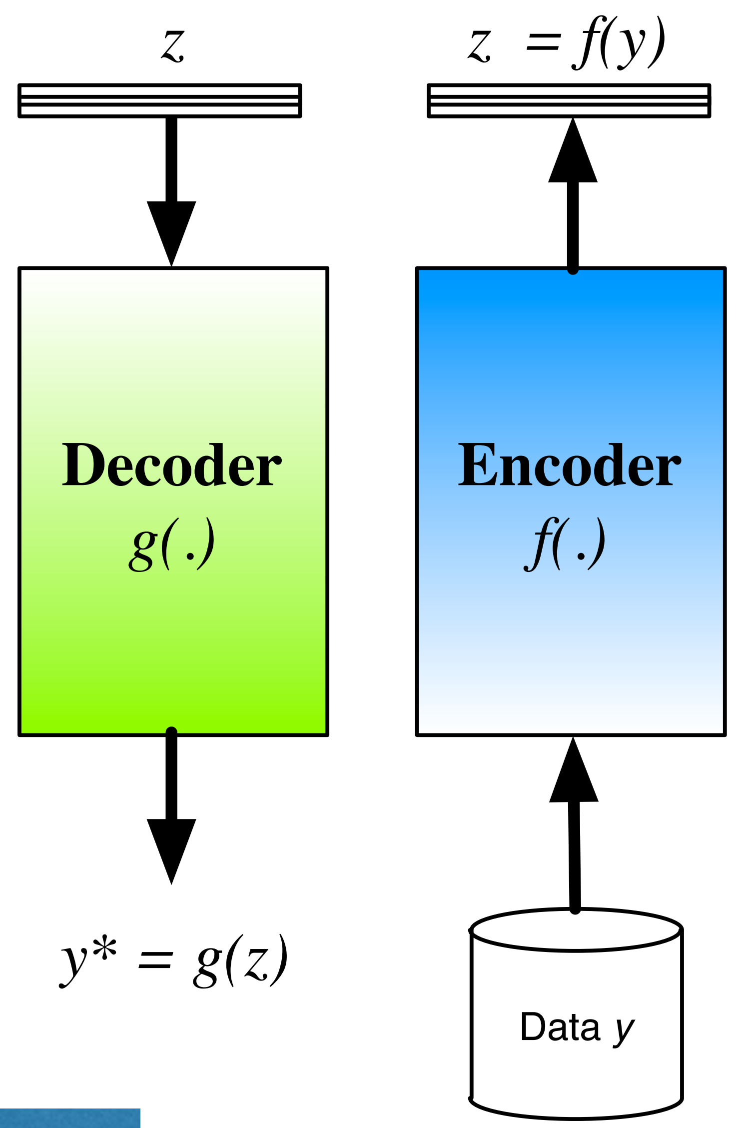
\- No natural handling of missing data

\- No representation of variability of the representation space.

$$\mathcal{L} = -\log p(y|g(z))$$

$$\mathcal{L} = \|y - g(f(y))\|_2^2$$

Dimensionality Reduction and Auto-encoders



Some questions about auto-encoders:

- **What is the model we are interested in?**
- **Why use an encoder?**
- **How do we regularise?**

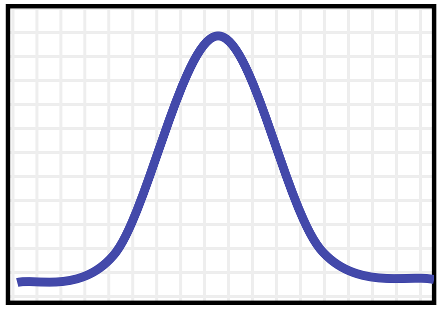
Best to be explicit about the:

- Probabilistic **model** of interest and
- Mechanism we use for **inference**.

Density Estimation and Latent Variable Models

Latent variable models:

- Generic and flexible model class for density estimation.
- Specifies a generative process that gives rise to the data.



Latent Gaussian Models:

- Probabilistic PCA, Factor analysis (FA), Bayesian Exponential Family PCA (BXPCA).

BXPCA

Latent Variable

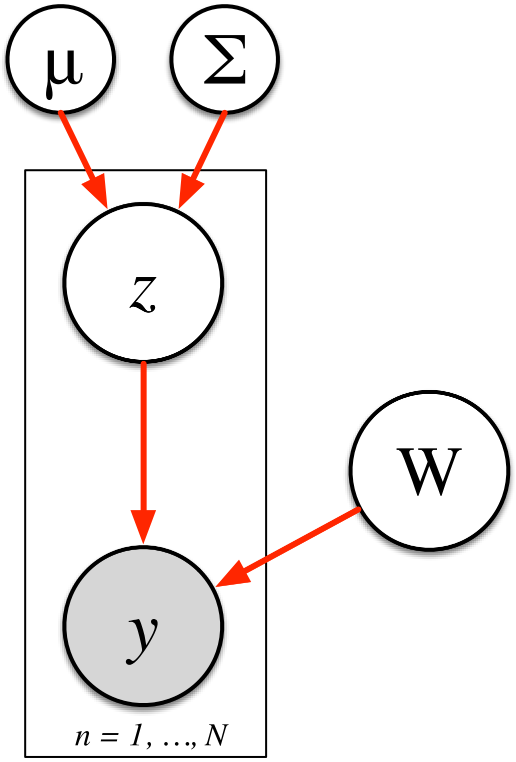
$$\mathbf{z} \sim \mathcal{N}(\mathbf{z}|\mu, \mathbf{\Sigma})$$

Observation Model

$$\boldsymbol{\eta} = \mathbf{W}\mathbf{z} + \mathbf{b}$$

$$\mathbf{y} \sim \mathrm{Expon}(\mathbf{y}|\boldsymbol{\eta})$$

Exponential fam natural parameters η.



Use our knowledge of deep learning to design even richer models.

Deep Generative Models

Rich extension of previous model using deep neural networks.

E.g., non-linear factor analysis, non-linear Gaussian belief networks, deep latent Gaussian models (DLGM).

DLGM

Latent Variables (Stochastic layers)

$$\mathbf{z}_l \sim \mathcal{N}(\mathbf{z}_l | f_l(\mathbf{z}_{l+1}), \mathbf{\Sigma}_l)$$

$$f_l(\mathbf{z}) = \sigma(\mathbf{W}h(\mathbf{z}) + \mathbf{b})$$

Deterministic layers

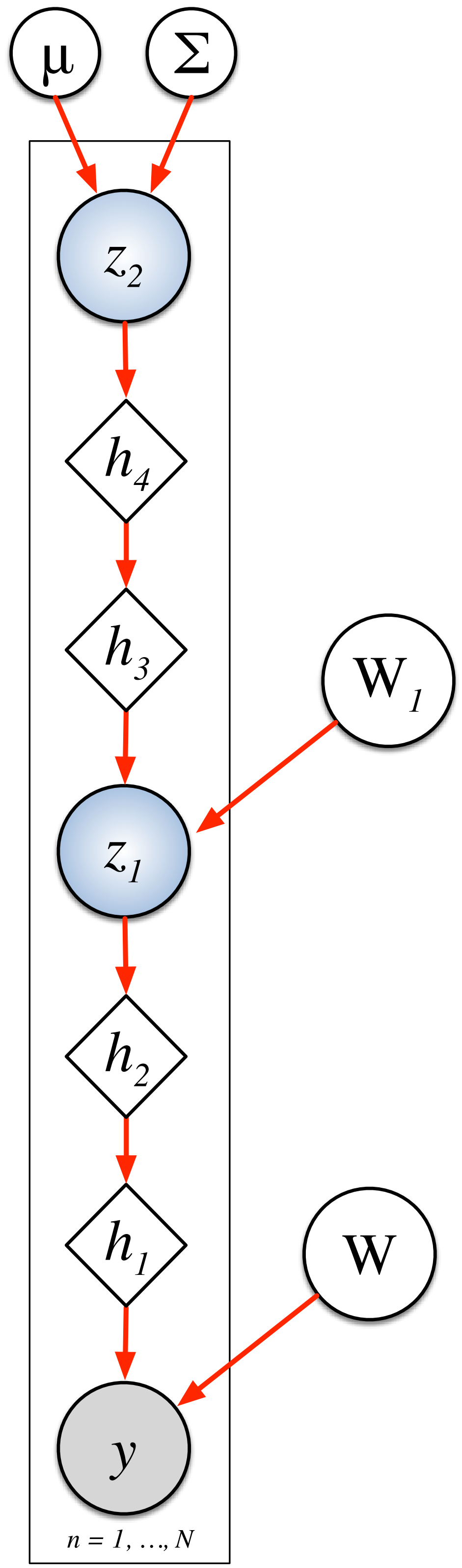
$$h_i(\mathbf{x}) = \sigma(\mathbf{A}\mathbf{x} + \mathbf{c})$$

Observation Model

$$\boldsymbol{\eta} = \mathbf{W}\mathbf{h_1} + \mathbf{b}$$

$$\mathbf{y} \sim \text{Expon}(\mathbf{y}|\boldsymbol{\eta})$$

Can also use non-exponential family.



Deep Latent Gaussian Models

Our inferential tasks are:

1. Explain this data

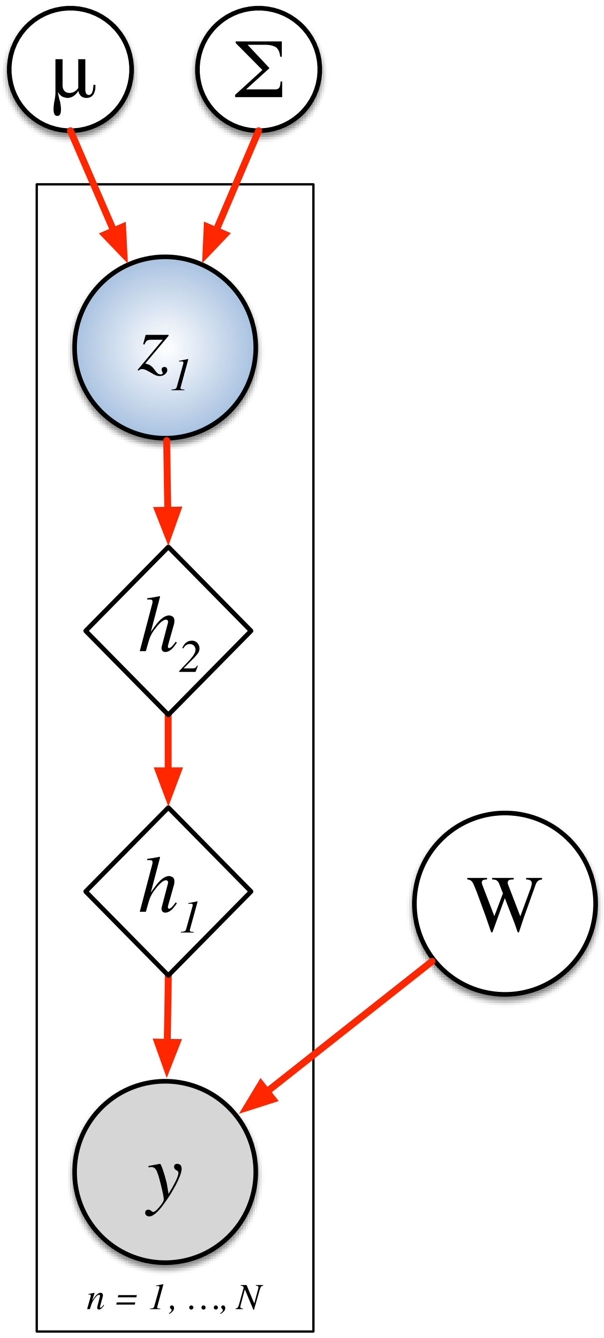
$$p(\mathbf{z}|\mathbf{y}, \mathbf{W}) \propto p(\mathbf{y}|\mathbf{z}, \mathbf{W})p(\mathbf{z})$$

2. Make predictions:

$$p(\mathbf{y}^*|\mathbf{y}) = \int p(\mathbf{y}^*|\mathbf{z}, \mathbf{W})p(\mathbf{z}|\mathbf{y}, \mathbf{W})d\mathbf{z}$$

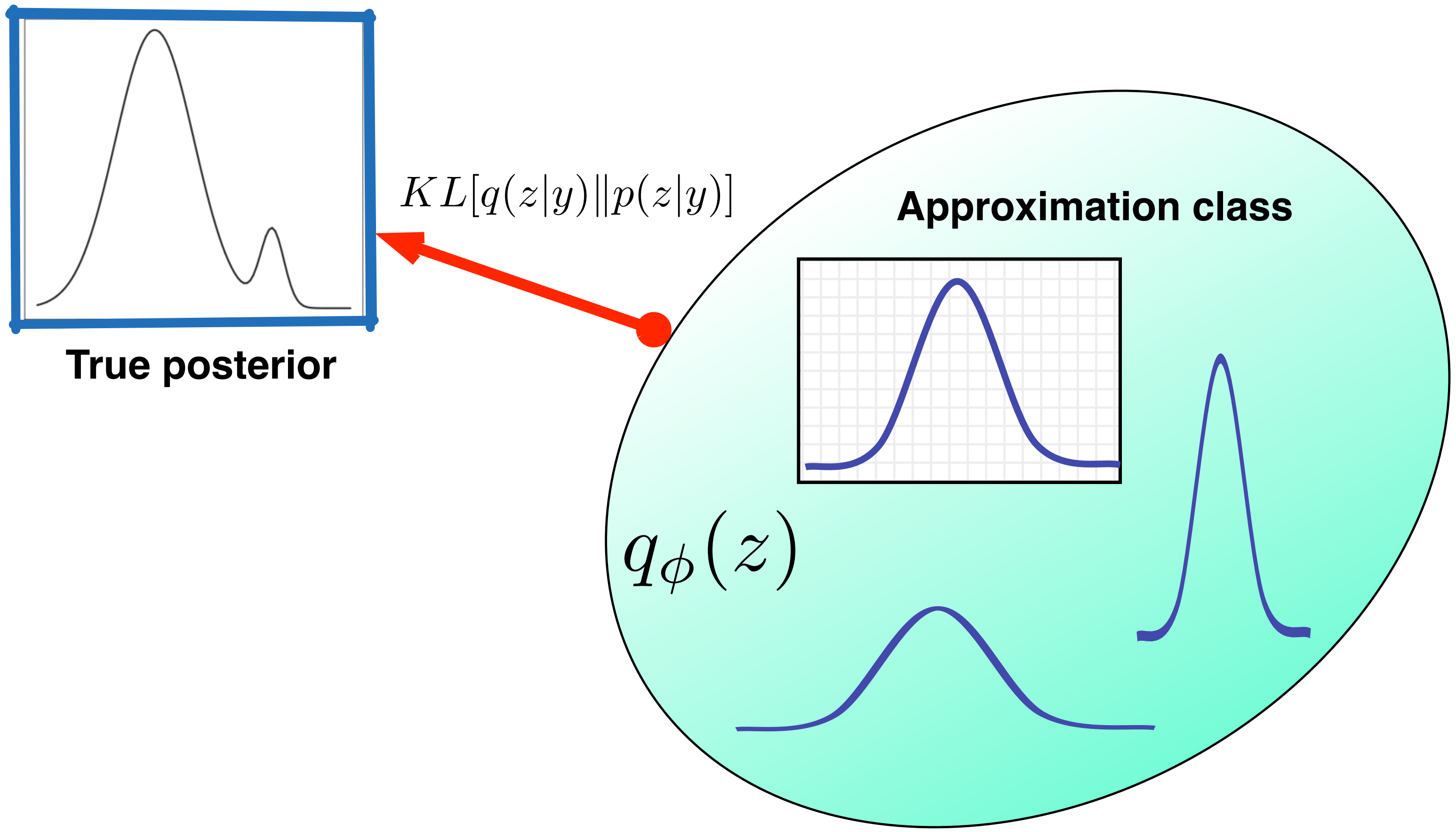
3. Choose the best model

$$p(\mathbf{y}|\mathbf{W}) = \int p(\mathbf{y}|\mathbf{z}, \mathbf{W})p(\mathbf{z})d\mathbf{z}$$



Variational Inference

Use tools from approximate inference to handle intractable integrals.



- **Reconstruction cost**: Expected log-likelihood measures how well samples from *q(z)* are able to explain the data *y*.
- **Penalty:** Explanation of the data *q(z)* doesn't deviate too far from your beliefs *p(z)* - Okham's razor.

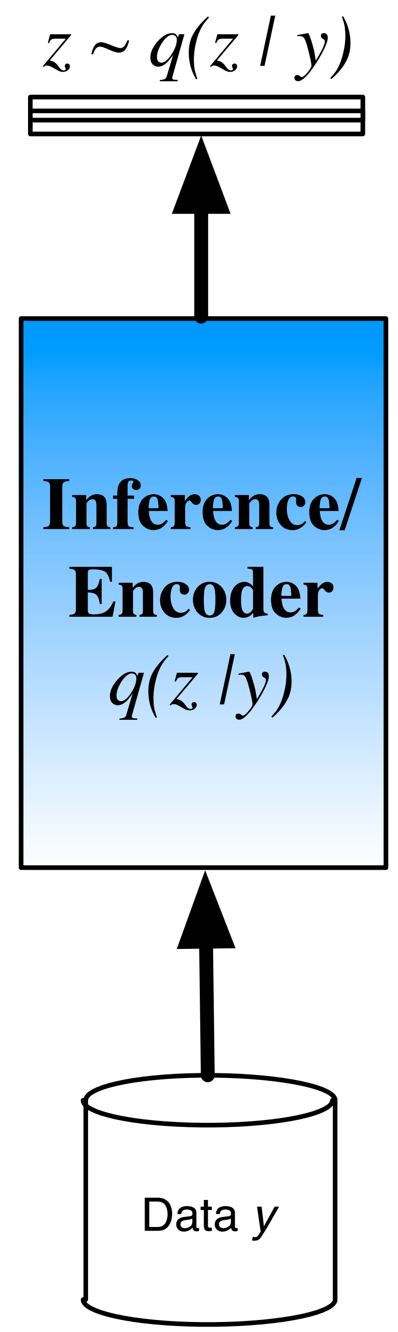
$$\mathcal{F}(y,q) = \overbrace{\boxed{\mathbb{E}_{q(z)}[\log p(y|z)]}}^{\text{Reconstruction}} - \overbrace{\boxed{KL[q(z)\|p(z)]}}^{\text{Penalty}}$$

Penalty is derived from your model and does not need to be designed.

Amortised Variational Inference

$$\mathcal{F}(y, q) = \boxed{\mathbb{E}_{q(z)}[\log p(y|z)]} - \boxed{KL[q(z)\|p(z)]}$$

Approx. Posterior | Reconstruction | Penalty



Approximate posterior distribution *q(z)*: Best match to true posterior *p(z|y)*, one of the unknown inferential quantities of interest to us.

Inference network: *q* is an encoder or inverse model.

Parameters of *q* are now a set of global parameters used for inference of all data points - test and train.

Amortise (spread) the cost of inference over all data.

Encoders provide an efficient mechanism for **amortised posterior inference**

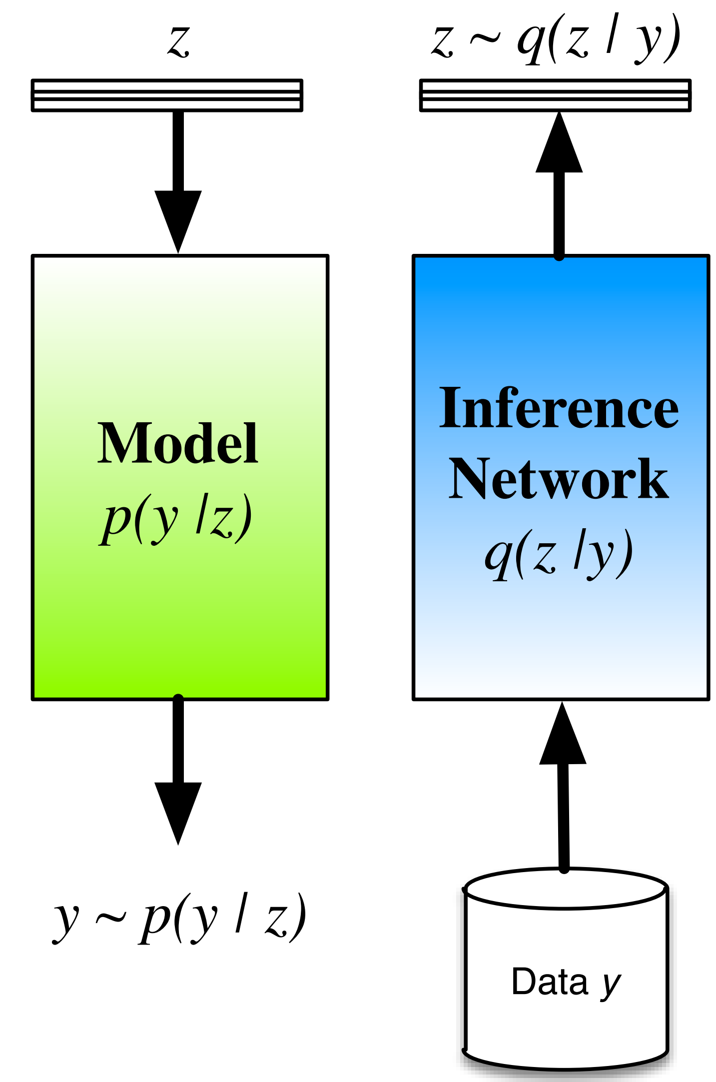
Auto-encoders and Inference in DGMs

$$\mathcal{F}(y, q) = \boxed{\mathbb{E}_{q(z)}[\log p(y|z)]} - \boxed{KL[q(z)\|p(z)]}$$

Approx. Posterior

Reconstruction

Penalty



- **Model (Decoder):** likelihood *p(y|z)*.
- **Inference (Encoder):** variational distribution *q(z|y)*

Stochastic encoder-decoder systems implement variational inference.

Specific combination of variational inference in latent variable models using inference networks

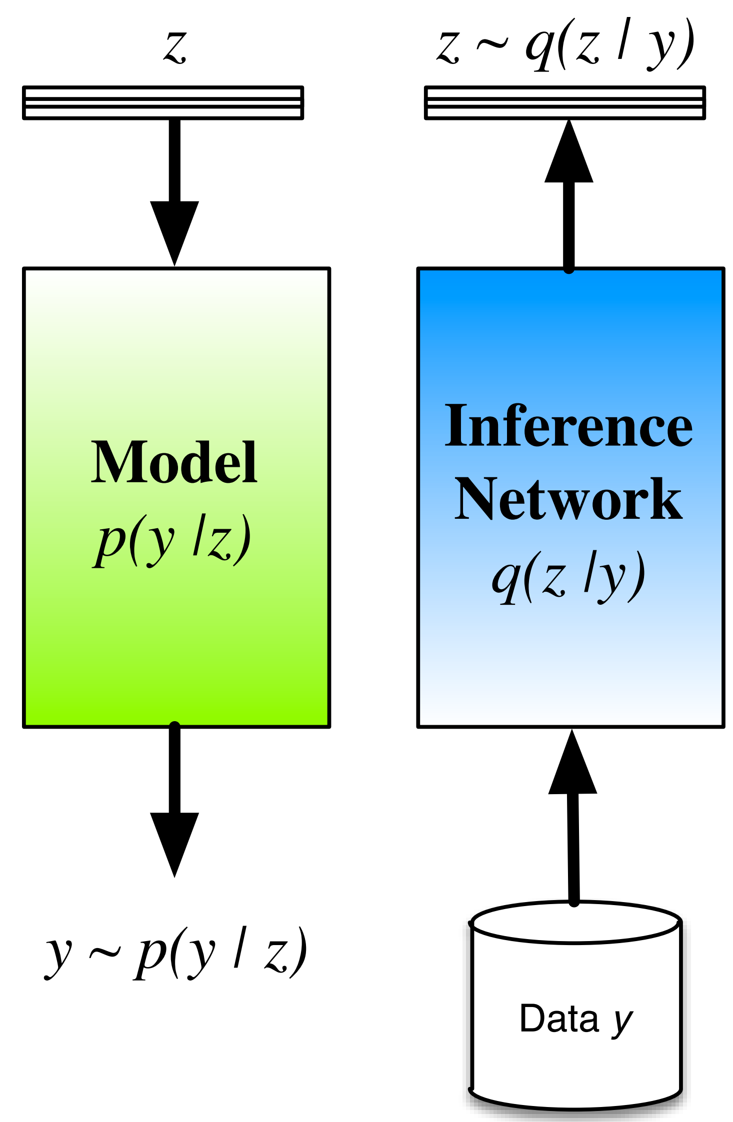
Variational Auto-encoder

But don't forget what your model is, and what inference you use.

What Have we Gained

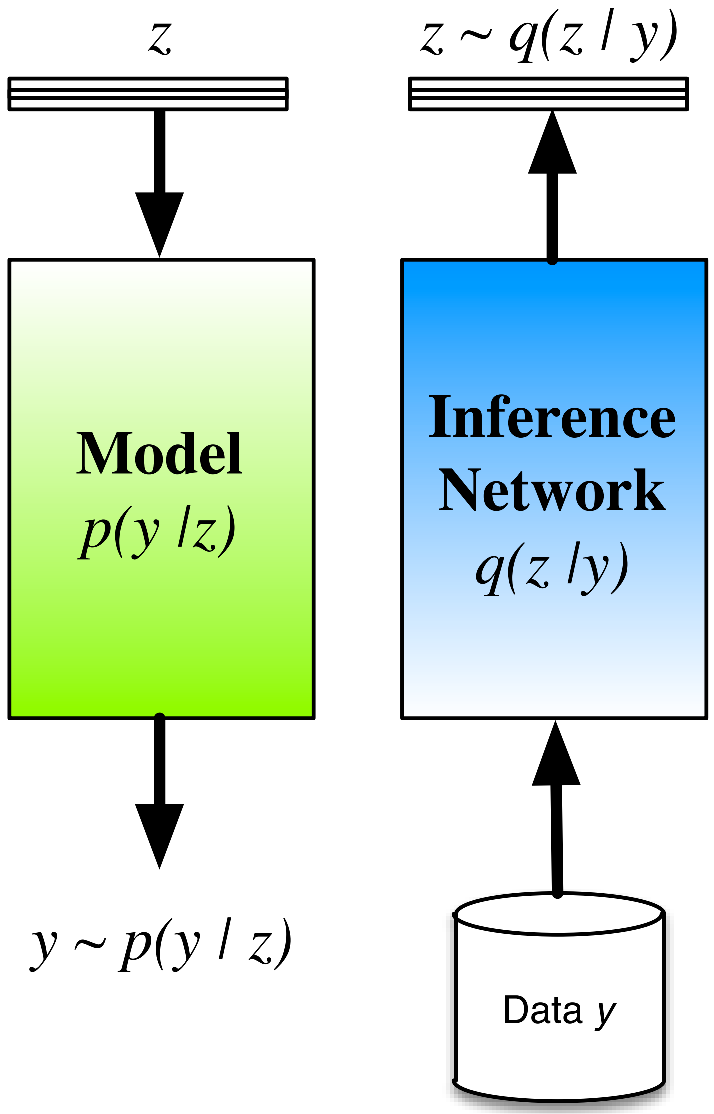
- Transformed an auto-encoders into more interesting **deep generative models.**
- Rich new class of density estimators built with **non-linear models.**
- Used a **principled approach** for deriving loss functions that automatically include appropriate penalty functions.
- Explained **how** an encoder enters into our models and **why** this is a good idea.
- Able to answer all our desired **inferential questions.**
- **Knowledge of the uncertainty** associated with our latent variables.

$$\mathcal{F}(y, q) = \mathbb{E}_{q(z)}[\log p(y|z)] - KL[q(z)\|p(z)]$$



What Have we Gained

$$\mathcal{F}(y, q) = \mathbb{E}_{q(z)}[\log p(y|z)] - KL[q(z)\|p(z)]$$



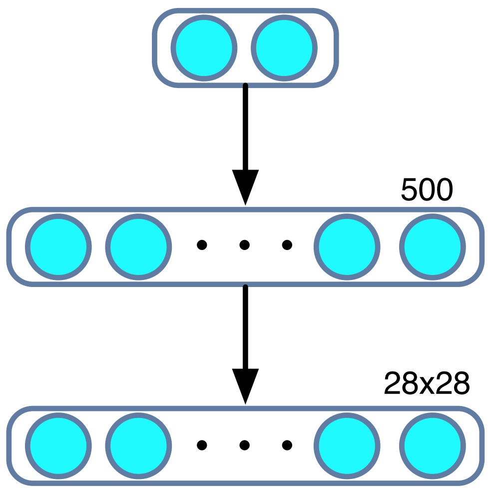
- Able to **score our models** and do model selection using the free energy.
- Can **impute missing data** under any missingness assumption
- Can still combine with natural gradient and improved optimisation tools.
- Easy implementation - have a single computational graph and simple Monte Carlo gradient estimators.
- Computational complexity the same as any large-scale deep learning system.

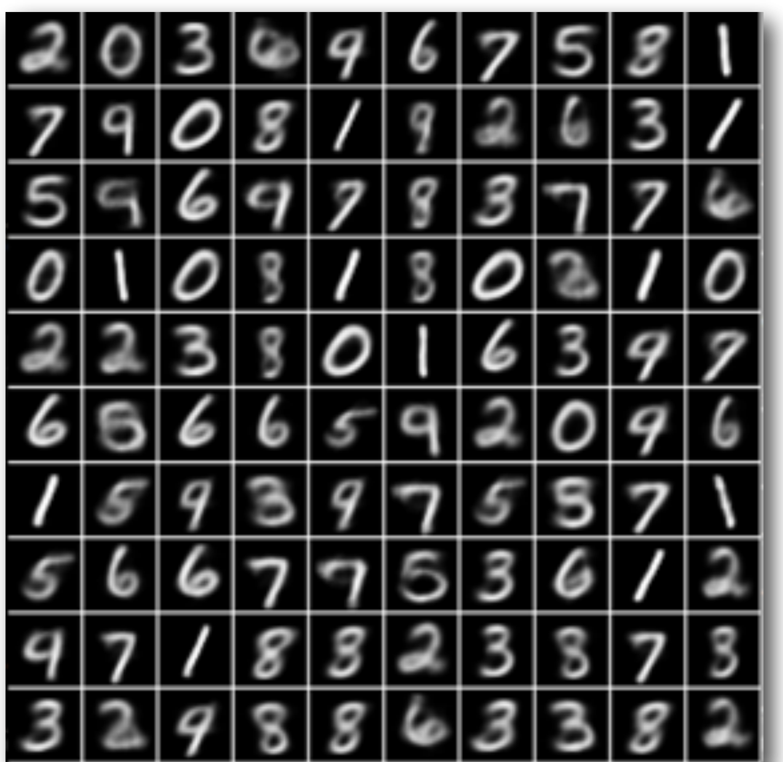
A true marriage of Bayesian Reasoning and Deep Learning

Data Visualisation

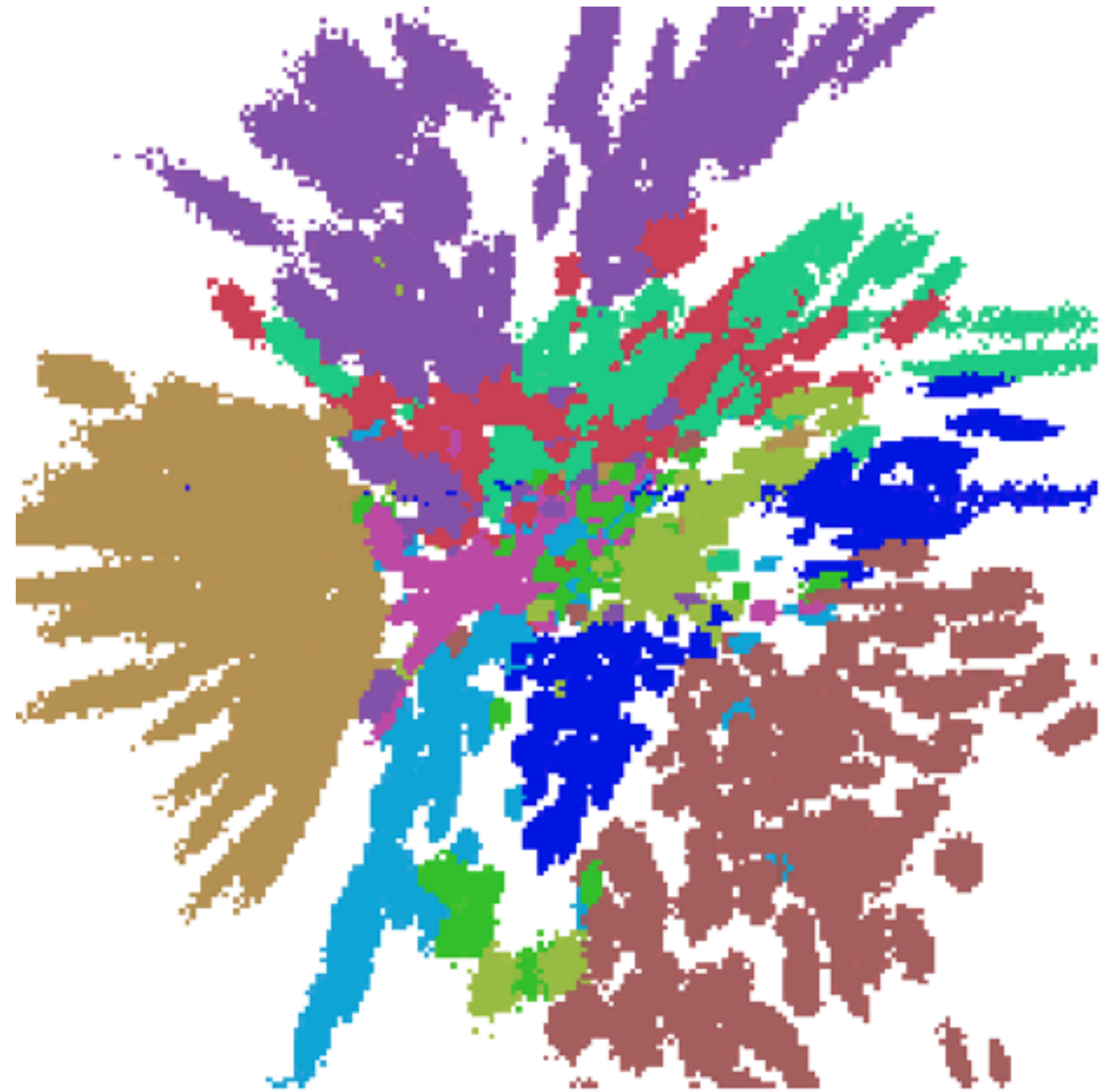
MNIST Handwritten digits

DLGM



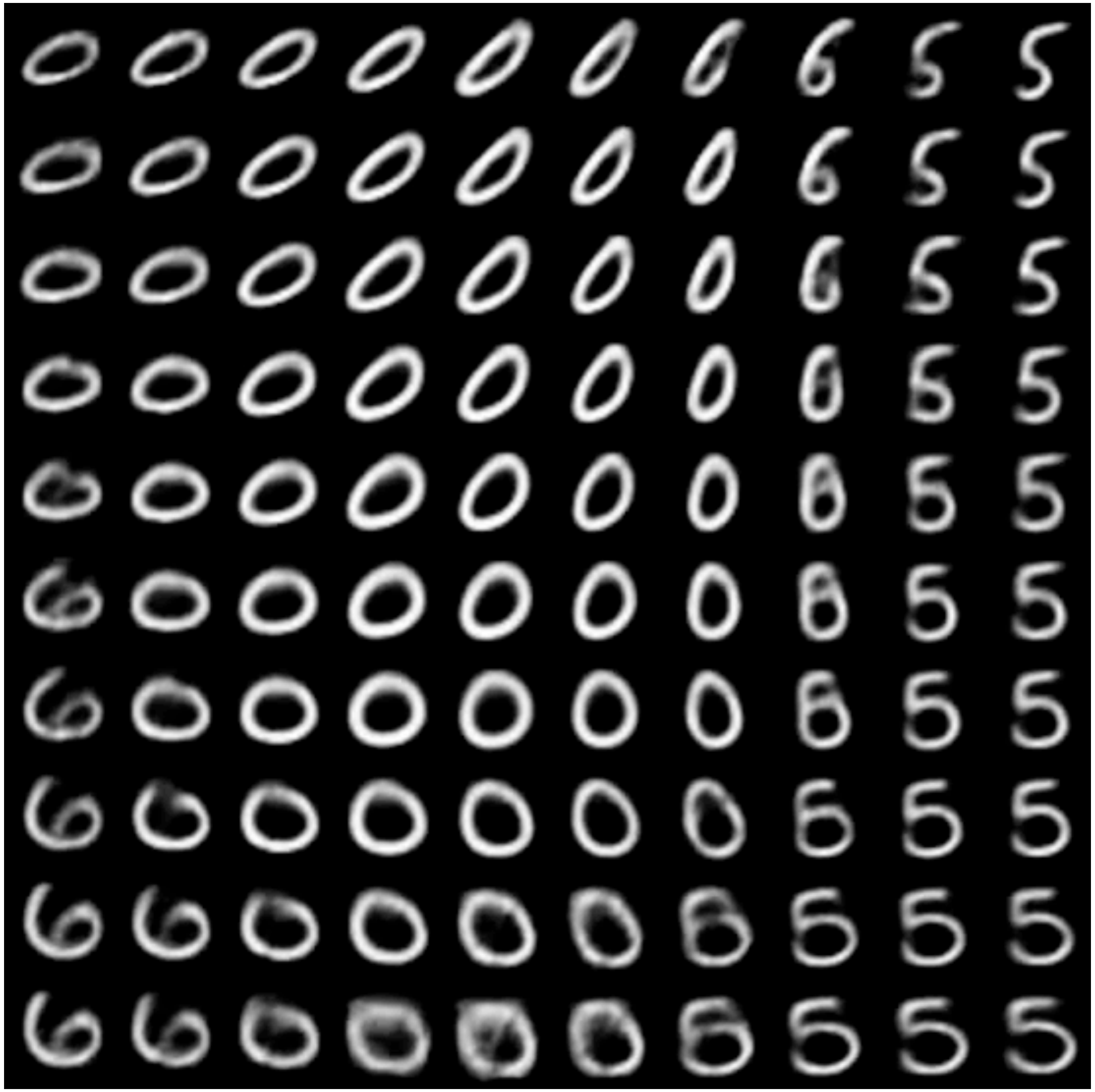


Samples from 2D latent model



Labels in 2D latent space

Visualising MNIST in 3D

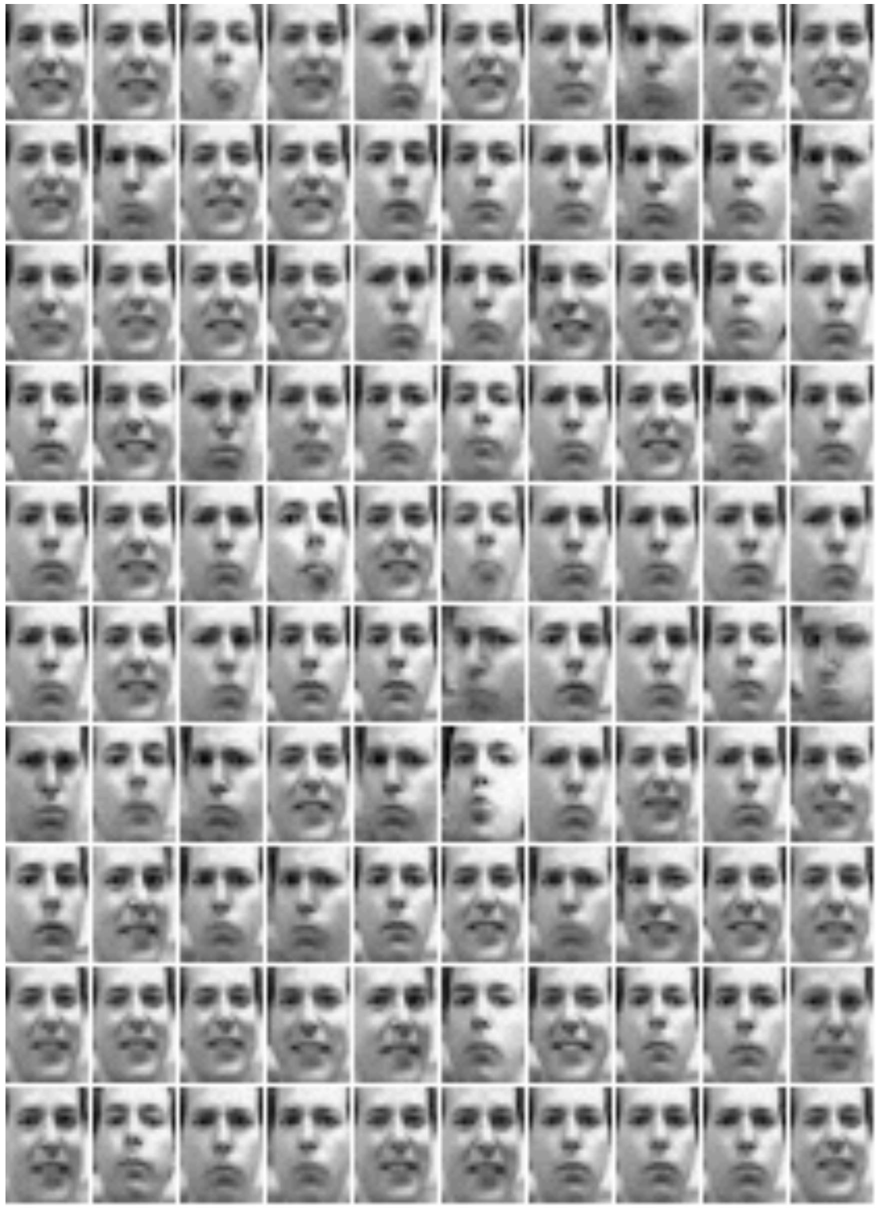


DLGM

Data Simulation

DLGM

Data



Samples

Missing Data Imputation

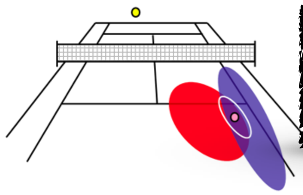
DLGM

Original Data | unobserved pixels | Inferred Image

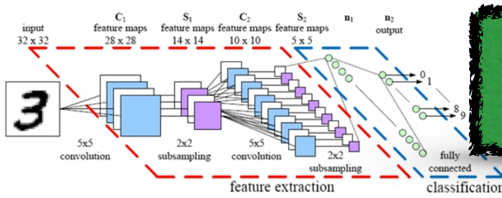
10% observed

50% observed

Outline



Bayesian Reasoning +



Deep Learning

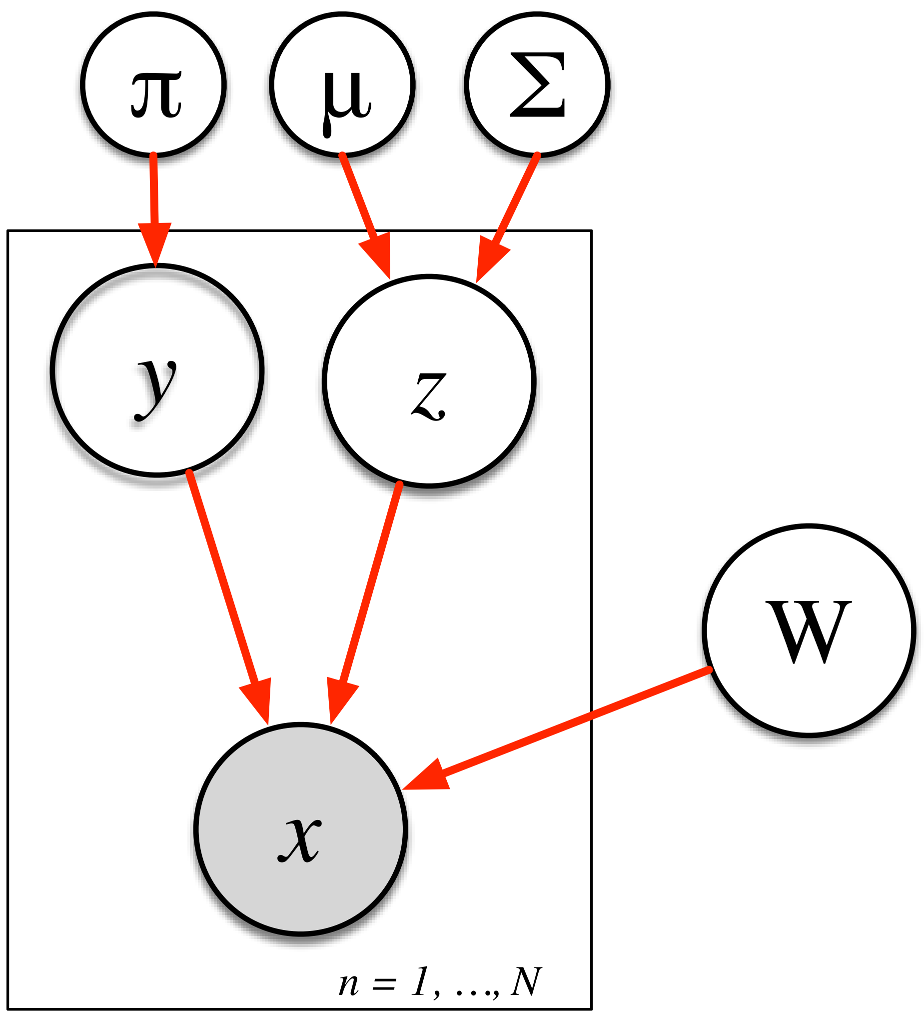
Complementary strengths that we should expect to be successfully combined.

1. Why is this a good idea?
   - ✤ Review of deep learning
   - ✤ Limitations of maximum likelihood and MAP estimation
2. How can we achieve this convergence?
   - ✤ Auto-encoders and latent variable models
   - ✤ Approximate and variational inference
3. What else can we do?
   - ✤ Semi-supervised learning, recurrent networks, classification, better inference and more.

Semi-supervised Learning

Can extend the marriage of Bayesian reasoning and deep learning to the problem of semi-supervised classification.

Semi-supervised DLGM



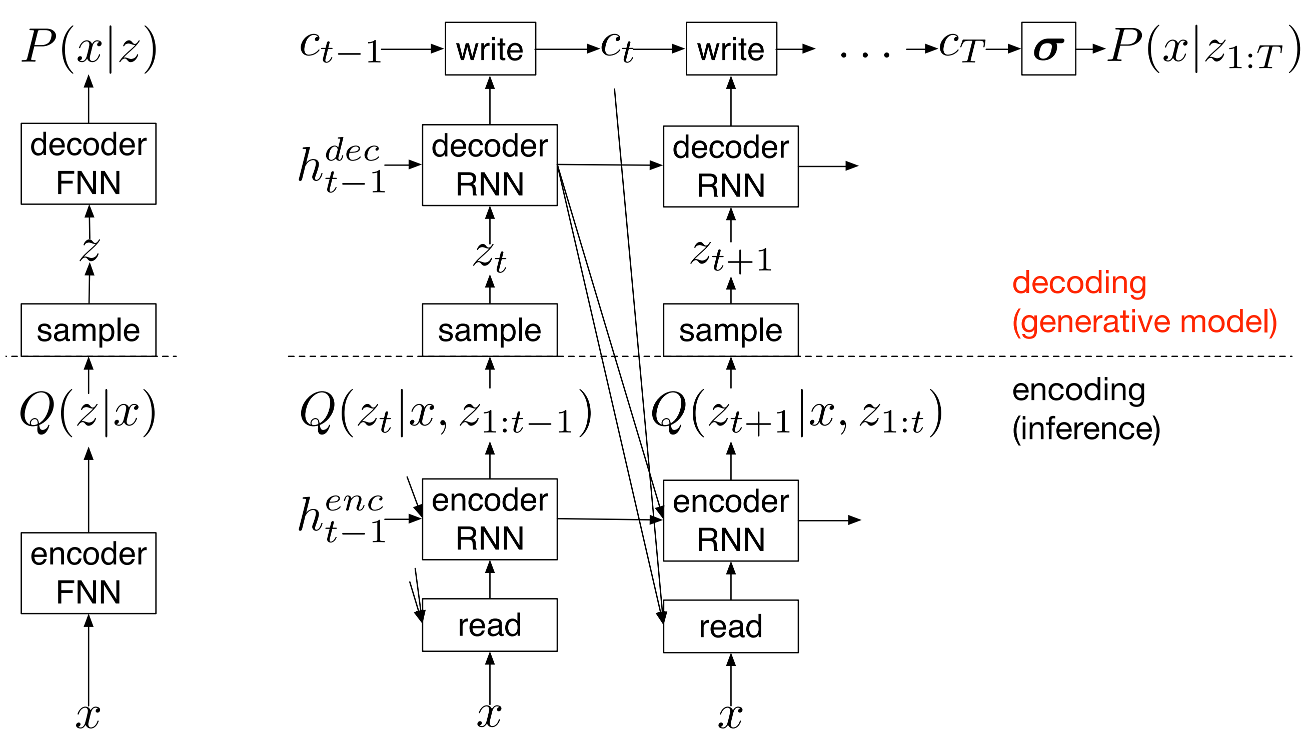
Analogical Reasoning

Semi-supervised DLGM

Generative Models with Attention

We can also combine other tools from deep learning to design even more powerful generative models: recurrent networks and attention.

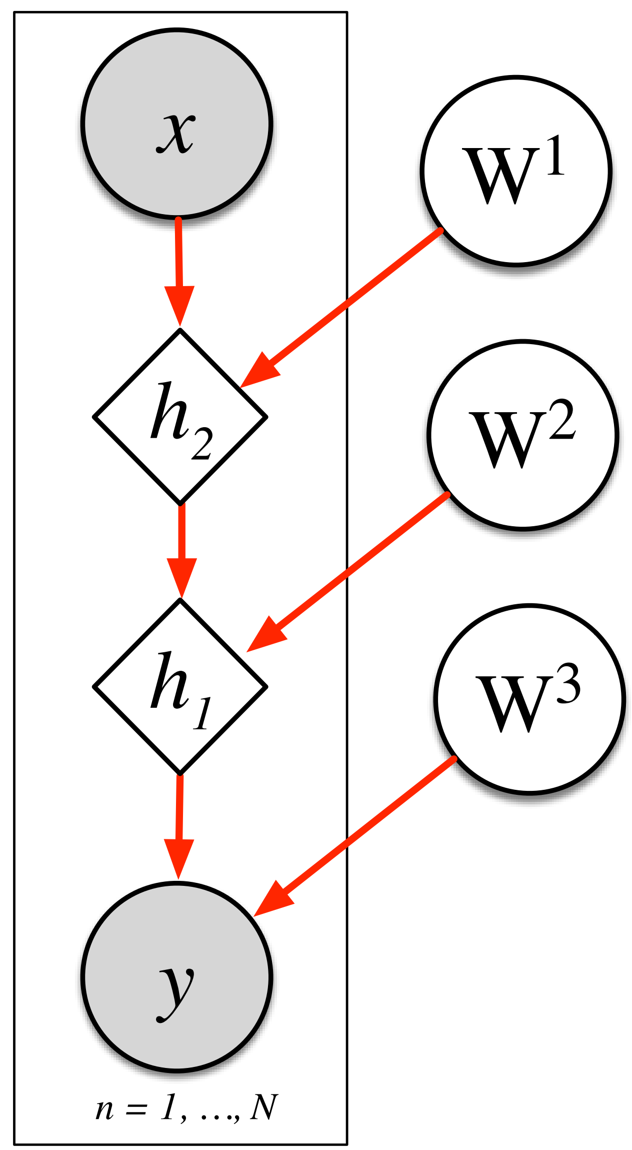
DRAW

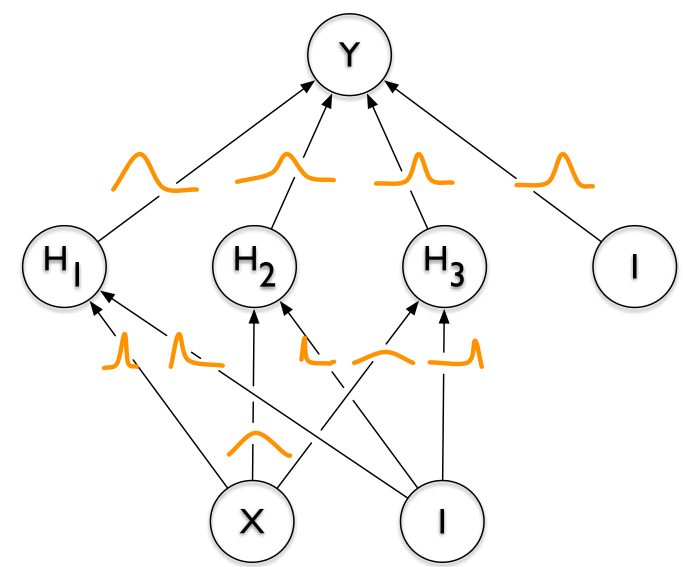


Uncertainty on Model Parameters

We can also combine other tools from deep learning to design even more powerful generative models: recurrent networks and attention.

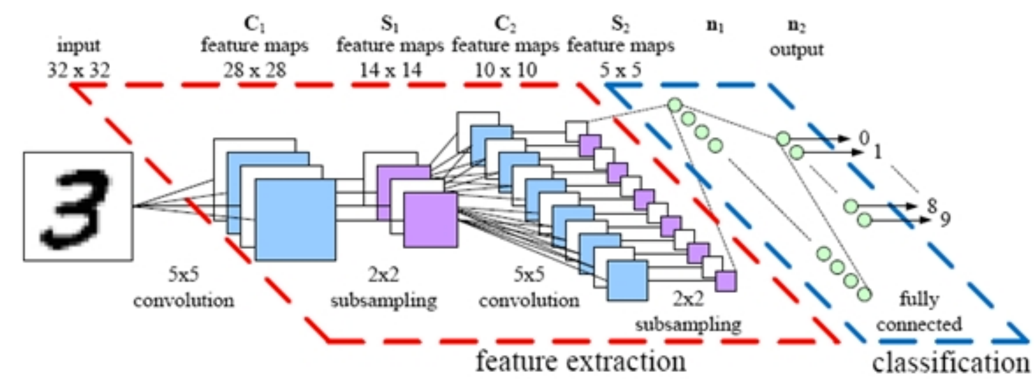
Bayesian Neural Networks

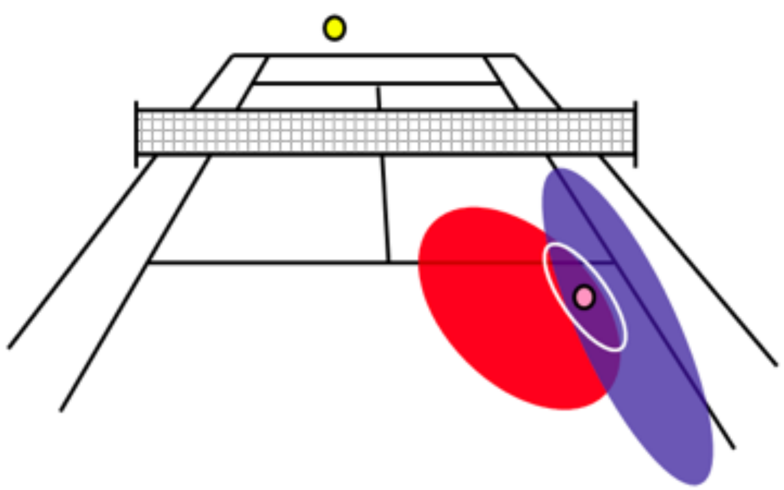




In Review ...

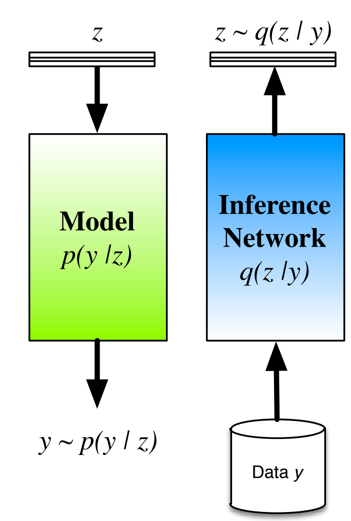
Deep learning as a ***framework for building highly flexible non-linear parametric models***, but regularisation and accounting for uncertainty and lack of knowledge is still needed.





Bayesian reasoning as a general ***framework for inference that allows us to account for uncertainty*** and a principled approach for regularisation and model scoring.

Combined Bayesian reasoning with auto-encoders and showed just how much can be gained by a ***marriage of these two streams*** of machine learning research.



Thanks to many people:

Danilo Rezende, Ivo Danihelka, Karol Gregor, Charles Blundell, Theophane Weber, Andriy Mnih, Daan Wierstra (*Google DeepMind*), Durk Kingma, Max Welling (*U. Amsterdam*)

Thank You.

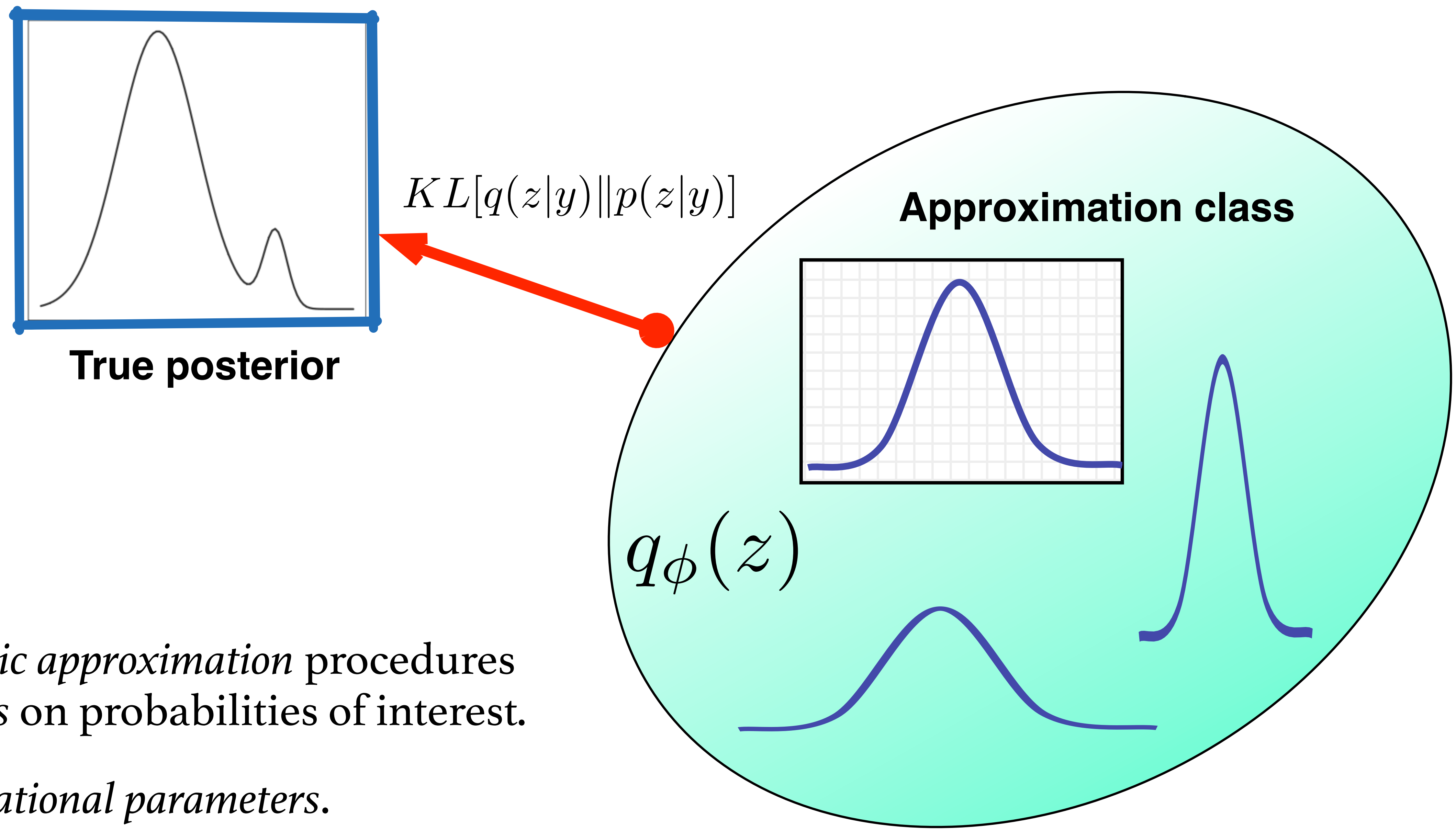
Some References

Probabilistic Deep Learning

- Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra. "*Stochastic backpropagation and approximate inference in deep generative models*." ICML (2014).
- Kingma, Diederik P., and Max Welling. "*Auto-encoding variational Bayes*." ICLR 2014.
- Mnih, Andriy, and Karol Gregor. "*Neural variational inference and learning in belief networks*." ICML (2014).
- Gregor, Karol, et al. "*Deep autoregressive networks*." ICML (2014).
- Kingma, D. P., Mohamed, S., Rezende, D. J., & Welling, M. (2014). *Semi-supervised learning with deep generative models*. NIPS (pp. 3581-3589).
- Gregor, K., Danihelka, I., Graves, A., & Wierstra, D. (2015). **DRAW: A recurrent neural network for image generation.** arXiv preprint arXiv:1502.04623.
- Rezende, D. J., & Mohamed, S. (2015). **Variational Inference with Normalizing Flows**. arXiv preprint arXiv:1505.05770.
- Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). **Weight Uncertainty in Neural Networks**. arXiv preprint arXiv:1505.05424.
- Hernández-Lobato, J. M., & Adams, R. P. (2015). **Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks**. arXiv preprint arXiv:1502.05336.
- Gal, Y., & Ghahramani, Z. (2015). **Dropout as a Bayesian approximation: Representing model uncertainty in deep learning.** arXiv preprint arXiv:1506.02142.

What is a Variational Method?

Variational Principle
General family of methods for approximating complicated densities by a simpler class of densities.



Deterministic approximation procedures with *bounds* on probabilities of interest.

Fit the *variational parameters*.

From IS to Variational Inference

Integral problem

$$\log p(y) = \log \int p(y|z)p(z)dz$$

Proposal

$$\log p(y) = \log \int p(y|z)p(z)\frac{q(z)}{q(z)}dz$$

Importance Weight

$$\log p(y) = \log \int p(y|z)\frac{p(z)}{q(z)}q(z)dz$$

Jensen's inequality

$$\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$$

$$\log p(y) \geq \int q(z) \log \left(p(y|z)\frac{p(z)}{q(z)} \right) dz$$

$$= \int q(z) \log p(y|z) - \int q(z) \log \frac{q(z)}{p(z)}$$

Variational lower bound

$$\boxed{= \mathbb{E}_{q(z)}[\log p(y|z)] - KL[q(z)\|p(z)]}$$

Minimum Description Length (MDL)

$$\mathcal{F}(y, q) = \boxed{\mathbb{E}_{q(z)}[\log p(y|z)]} - \boxed{KL[q(z)\|p(z)]}$$

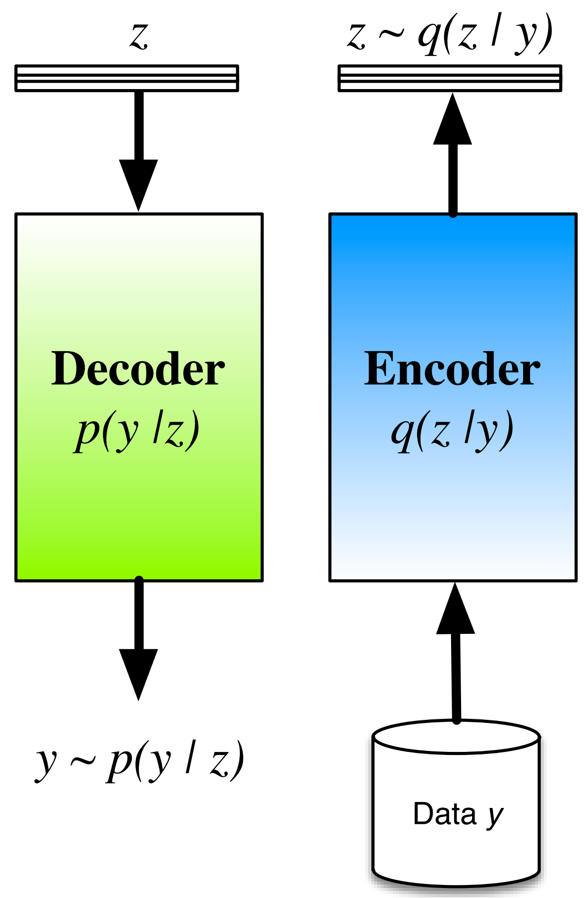
Stochastic encoder

Data code-length

Hypothesis code

Stochastic encoder-decoder systems implement variational inference.

- Regularity in our data that can be explained with latent variables, implies that the data is compressible.
- MDL: inference seen as a problem of compression — we must find the ideal shortest message of our data *y*: marginal likelihood.
- Must introduce an approximation to the ideal message.
- **Encoder:** variational distribution *q(z|y)*,
- **Decoder:** likelihood *p(y|z)*.



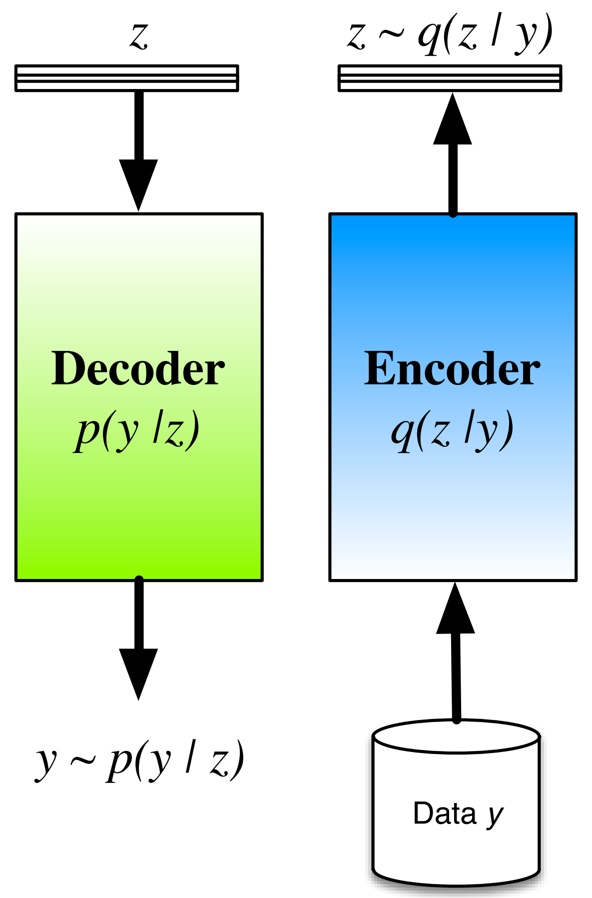
Denoising Auto-encoders (DAE)

$$\mathcal{F}(y, q) = \underbrace{\mathbb{E}_{q(z)}[\log p(y|z)]}_{\text{Reconstruction}} - \underbrace{\Omega(z, y)}_{\text{Penalty}}$$

Stochastic encoder

Stochastic encoder-decoder systems implement variational inference.

- DAE: A mechanism for finding representations or features of data (i.e. latent variable explanations).
- **Encoder:** variational distribution *q(z|y)*,
- **Decoder:** likelihood *p(y|z)*.



The variational approach requires you to be explicit about your assumptions. Penalty is derived from your model and does not need to be designed.

Amortising the Cost of Inference

Repeat:

E-step

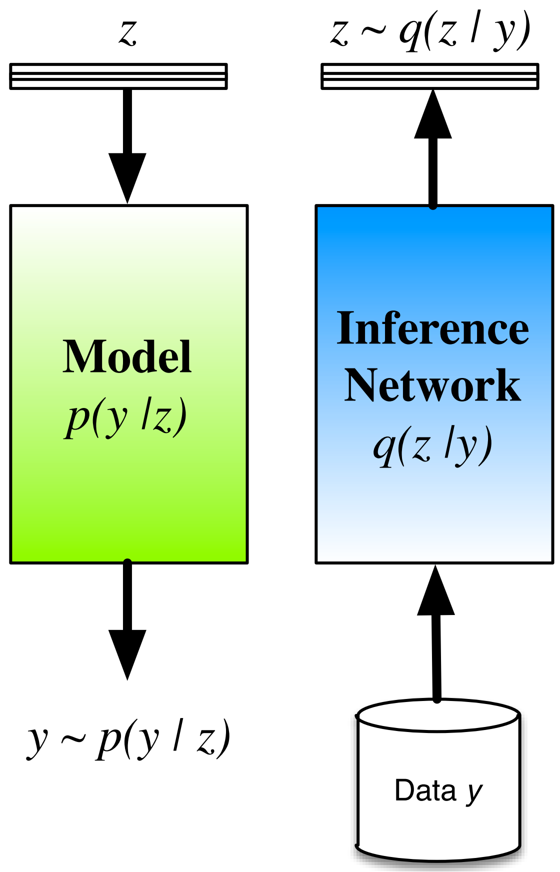
For i = 1, … N

$$\phi_n \propto \nabla_\phi \mathbb{E}_{q_\phi(z)}[\log p_\theta(y_n|z_n)] - \nabla_\phi KL[q(z_n)\|p(z_n)]$$

Instead of solving this optimisation for every data point *n*, we can instead use a model.

M-step

$$\theta \propto \frac{1}{N}\sum_n \nabla_\theta \log p_\theta(y_n|z_n)$$



Inference network: q is an encoder or inverse model.

Parameters of q are now a set of global parameters used for inference of all data points - test and train.

Share the cost of inference (amortise) over all data.

Combines easily with mini-batches and Monte Carlo expectations.

Can jointly optimise variational and model parameters: no need for alternating optimisation.

Implementing your Variational Algorithm

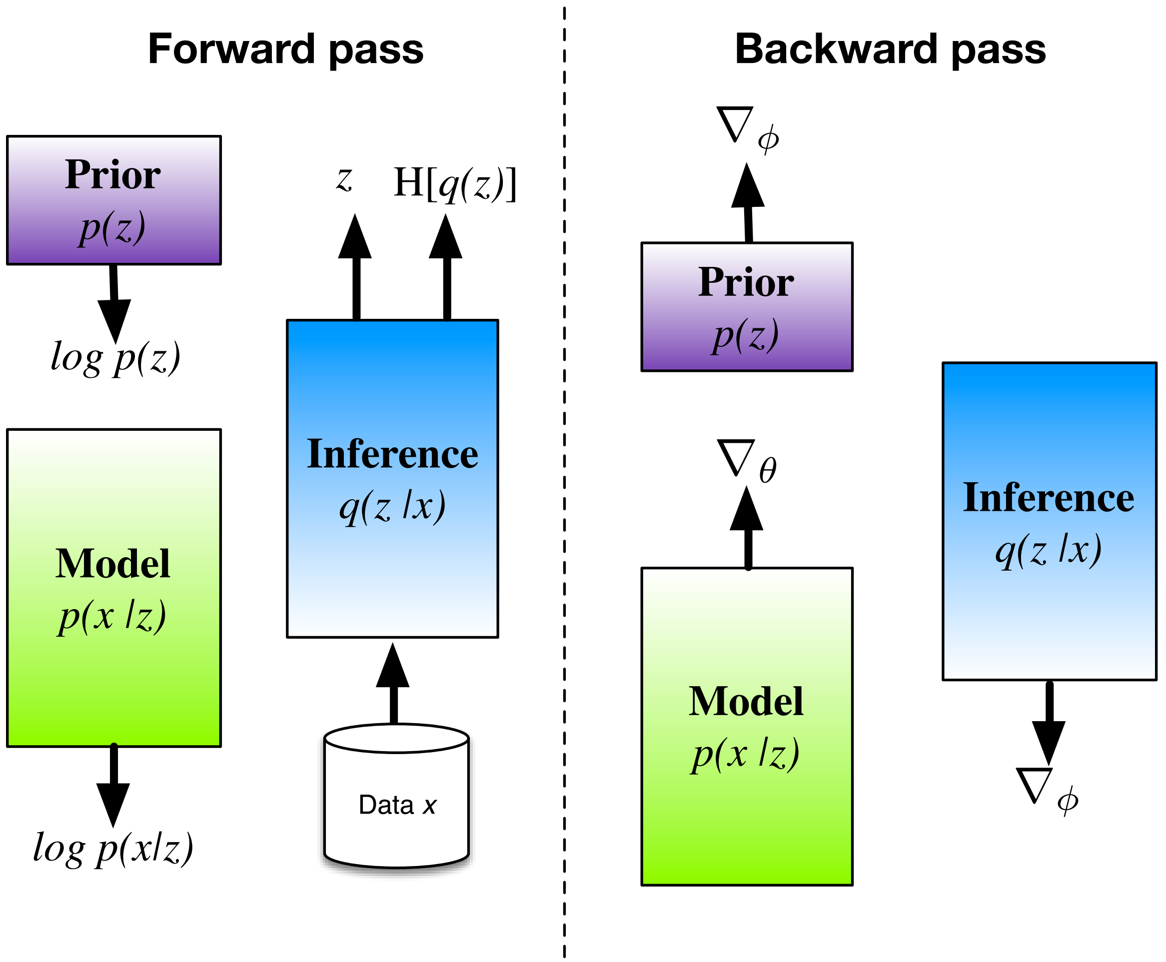
Avoid deriving pages of gradient updates for variational inference.

Variational inference turns integration into optimisation:

- **Automated Tools:**
  Differentiation: Theano, Torch7, Stan
  Message passing: infer.NET

- Stochastic gradient descent and other preconditioned optimisation.
- Same code can run on both GPUs or on distributed clusters.
- Probabilistic models are modular, can easily be combined.

$$\mathbb{E}_q[(-\log p(y|z) + \log q(z) - \log p(z)]$$



Ideally want probabilistic programming using variational inference.

Stochastic Backpropagation

A Monte Carlo method that works with continuous latent variables.

Original problem

$$\nabla_\xi \mathbb{E}_{q(z)}[f(z)]$$

Reparameterisation

$$z \sim \mathcal{N}(\mu, \sigma^2)$$
$$z = \mu + \sigma\epsilon \quad \epsilon \sim \mathcal{N}(0, 1)$$

Backpropagation with Monte Carlo

$$\nabla_\xi \mathbb{E}_{\mathcal{N}(0,1)}[f(\mu + \sigma\epsilon)]$$
$$\mathbb{E}_{\mathcal{N}(0,1)}[\nabla_{\xi=\{\mu,\sigma\}} f(\mu + \sigma\epsilon)]$$

- Can use *any likelihood function*, avoids the need for additional lower bounds.
- *Low-variance*, unbiased estimator of the gradient.
- Can use just *one sample* from the base distribution.
- Possible for many distributions with location-scale or other known transformations, such as the CDF.

Monte Carlo Control Variate Estimators

More general Monte Carlo approach that can be used with both discrete or continuous latent variables.

Property of the score function: $$\nabla_\xi \log q_\xi(z|x) = \frac{\nabla_\xi q_\xi(z|x)}{q_\xi(z|x)}$$

Original problem $$\nabla_\phi \mathbb{E}_{q_\phi(z)}[\log p_\theta(y|z)]$$

Score ratio $$\mathbb{E}_{q_\phi(z)}[\log p_\theta(y|z)\nabla_\phi \log q(z|y)]$$

MCCV Estimate $$\mathbb{E}_{q_\phi(z)}[(\log p_\theta(y|z) - c)\nabla_\phi \log q(z|y)]$$

c is known as a ***control variate*** and is used to control the variance of the estimator.

Variational renormalisation for stacked Boltzmann machines

Lars Haringa

Universiteit van Amsterdam

March 17, 2016

Mehta, Schwab (2014) - An exact mapping between the variational renormalization group and deep learning

Overview

Renormalisation group

- In 1954, coupling parameter g in quantum electrodynamics was found to satisfy

$$g(\mu) = G^{-1}\left(\left(\frac{\mu}{M}\right)^{d} G\left(g(M)\right)\right) \text{ for } \mu, M \text{ scales}$$

Renormalisation group

- In 1954, coupling parameter g in quantum electrodynamics was found to satisfy

$$g(\mu) = G^{-1}\left(\left(\frac{\mu}{M}\right)^{d} G\left(g(M)\right)\right) \text{ for } \mu, M \text{ scales}$$

- Group equation

Renormalisation group

- In 1954, coupling parameter g in quantum electrodynamics was found to satisfy

$$g(\mu) = G^{-1}\left(\left(\frac{\mu}{M}\right)^{d} G\left(g(M)\right)\right) \text{ for } \mu, M \text{ scales}$$

- Group equation
- Describes interactions at different scales: coupling changes, but system remains self-similar

Renormalisation group

- In 1954, coupling parameter g in quantum electrodynamics was found to satisfy

$$g(\mu) = G^{-1}\left(\left(\frac{\mu}{M}\right)^{d} G\left(g(M)\right)\right) \text{ for } \mu, M \text{ scales}$$

- Group equation
- Describes interactions at different scales: coupling changes, but system remains self-similar
- Important tool in modern physics (quantum, particle, string), and Nobel prizes have been awarded

Renormalisation group

- In 1954, coupling parameter g in quantum electrodynamics was found to satisfy

$$g(\mu) = G^{-1}\left(\left(\frac{\mu}{M}\right)^{d} G\left(g(M)\right)\right) \text{ for } \mu, M \text{ scales}$$

- Group equation
- Describes interactions at different scales: coupling changes, but system remains self-similar
- Important tool in modern physics (quantum, particle, string), and Nobel prizes have been awarded
- Intuition: micro to macro, but math is abstract

Variational renormalisation

- Variational renormalisation group introduced in 1976 by Kadanoff et alii for spin models

Variational renormalisation

- Variational renormalisation group introduced in 1976 by Kadanoff et alii for spin models
- Block spin renormalisation

Variational renormalisation

- Variational renormalisation group introduced in 1976 by Kadanoff et alii for spin models
- Block spin renormalisation
- N spins $\{v_i\}_{i=1,\ldots,N}$ can take binary values ± 1

Variational renormalisation

- Variational renormalisation group introduced in 1976 by Kadanoff et alii for spin models
- Block spin renormalisation
- N spins $\{v_i\}_{i=1,\dots,N}$ can take binary values ± 1
- For configuration v, system has energy (Hamiltonian)

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{ijk} K_{i,j,k} v_i v_j v_k + \dots \right)$$

Variational renormalisation

- Variational renormalisation group introduced in 1976 by Kadanoff et alii for spin models
- Block spin renormalisation
- N spins $\{v_i\}_{i=1,\dots,N}$ can take binary values ± 1
- For configuration v, system has energy (Hamiltonian)

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{ijk} K_{i,j,k} v_i v_j v_k + \dots\right)$$

- Probability of configuration v given by Boltzmann distribution $p(v) = \frac{e^{-H(v)}}{Z}$ with partition function $Z = \sum_{\tilde{v}} e^{-H(\tilde{v})}$

Variational renormalisation

- Variational renormalisation group introduced in 1976 by Kadanoff et alii for spin models
- Block spin renormalisation
- N spins $\{v_i\}_{i=1,\ldots,N}$ can take binary values ± 1
- For configuration v, system has energy (Hamiltonian)

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{ijk} K_{i,j,k} v_i v_j v_k + \ldots \right)$$

- Probability of configuration v given by Boltzmann distribution $p(v) = \frac{e^{-H(v)}}{Z}$ with partition function $Z = \sum_{\tilde{v}} e^{-H(\tilde{v})}$
- Free energy $F^v = -\log Z$

Variational renormalisation

- Variational renormalisation group introduced in 1976 by Kadanoff et alii for spin models
- Block spin renormalisation
- N spins $\{v_i\}_{i=1,\ldots,N}$ can take binary values ± 1
- For configuration v, system has energy (Hamiltonian)

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{ijk} K_{i,j,k} v_i v_j v_k + \ldots \right)$$

- Probability of configuration v given by Boltzmann distribution $p(v) = \frac{e^{-H(v)}}{Z}$ with partition function $Z = \sum_{\tilde{v}} e^{-H(\tilde{v})}$
- Free energy $F^v = -\log Z$
- Goal: coarse-grained description with conservation of energy

Variational renormalisation

- For configuration v, system has Hamiltonian

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{i,j,k} K_{ijk} v_i v_j v_k + \dots\right)$$

- Goal: coarse-grained description with conservation of energy

Variational renormalisation

- For configuration v, system has Hamiltonian

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{i,j,k} K_{ijk} v_i v_j v_k + \dots \right)$$

- Goal: coarse-grained description with conservation of energy
- Introduce 'hidden' spins $\{h_j\}_{j=1,\dots,M}$ with $M < N$

Variational renormalisation

- For configuration v, system has Hamiltonian

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{i,j,k} K_{ijk} v_i v_j v_k + \ldots\right)$$

- Goal: coarse-grained description with conservation of energy
- Introduce 'hidden' spins $\{h_j\}_{j=1,\ldots,M}$ with $M < N$
- Describe system using hidden spins h and Hamiltonian

$$\tilde{H}(h) = -\left(\sum_i \tilde{K}_i h_i + \sum_{i,j} \tilde{K}_{ij} h_i h_j + \sum_{i,j,k} \tilde{K}_{ijk} h_i h_j h_k + \ldots\right)$$

Variational renormalisation

- For configuration v, system has Hamiltonian

$$H(v) = -\left(\sum_i K_i v_i + \sum_{i,j} K_{ij} v_i v_j + \sum_{i,j,k} K_{ijk} v_i v_j v_k + \ldots\right)$$

- Goal: coarse-grained description with conservation of energy
- Introduce 'hidden' spins $\{h_j\}_{j=1,\ldots,M}$ with $M < N$
- Describe system using hidden spins h and Hamiltonian

$$\tilde{H}(h) = -\left(\sum_i \tilde{K}_i h_i + \sum_{i,j} \tilde{K}_{ij} h_i h_j + \sum_{i,j,k} \tilde{K}_{ijk} h_i h_j h_k + \ldots\right)$$

- Find RG mapping $\{K\} \to \{\tilde{K}\}$, in terms of λ, with $F^v = F^h_\lambda$

Variational renormalisation

- Find RG mapping $\{K\} \to \{\tilde{K}\}$, in terms of λ, with $F^v = F^h_\lambda$

Variational renormalisation

- Find RG mapping $\{K\} \to \{\tilde{K}\}$, in terms of λ, with $F^v = F^h_\lambda$
- Mapping will depend on unknown parameters λ

Variational renormalisation

- Find RG mapping $\{K\} \to \{\tilde{K}\}$, in terms of λ, with $F^v = F^h_\lambda$
- Mapping will depend on unknown parameters λ
- Also on v, but: marginalise out (average over observations)

Variational renormalisation

- Find RG mapping $\{K\} \rightarrow \{\tilde{K}\}$, in terms of λ, with $F^v = F^h_\lambda$
- Mapping will depend on unknown parameters λ
- Also on v, but: marginalise out (average over observations)
- Energy conservation often not exact; optimise $F^h_\lambda - F^v$ with λ

Variational renormalisation

- Find RG mapping $\{K\} \rightarrow \{\tilde{K}\}$, in terms of λ, with $F^v = F^h_\lambda$
- Mapping will depend on unknown parameters λ
- Also on v, but: marginalise out (average over observations)
- Energy conservation often not exact; optimise $F^h_\lambda - F^v$ with λ
- In that case, RG typically not invertible, so in fact a semigroup

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0
- Hamiltonian of system is $H(v) = -J^0 \sum_{i,i+1} v_i v_{i+1}$

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0
- Hamiltonian of system is $H(v) = -J^0 \sum_{i,i+1} v_i v_{i+1}$
- Coupling favours spins that agree

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0
- Hamiltonian of system is $H(v) = -J^0 \sum_{i,i+1} v_i v_{i+1}$
- Coupling favours spins that agree
- Skip over every other spin, so that spacing is $2a$; solve for J^1

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0
- Hamiltonian of system is $H(v) = -J^0 \sum_{i,i+1} v_i v_{i+1}$
- Coupling favours spins that agree
- Skip over every other spin, so that spacing is $2a$; solve for J^1
- Iterating RG turns out to satisfy $\tanh(J^{n+1}) = \tanh^2(J^n)$

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0
- Hamiltonian of system is $H(v) = -J^0 \sum_{i,i+1} v_i v_{i+1}$
- Coupling favours spins that agree
- Skip over every other spin, so that spacing is $2a$; solve for J^1
- Iterating RG turns out to satisfy $\tanh(J^{n+1}) = \tanh^2(J^n)$
- Higher order coupling iterations represent 'effective' behaviour

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0
- Hamiltonian of system is $H(v) = -J^0 \sum_{i,i+1} v_i v_{i+1}$
- Coupling favours spins that agree
- Skip over every other spin, so that spacing is $2a$; solve for J^1
- Iterating RG turns out to satisfy $\tanh(J^{n+1}) = \tanh^2(J^n)$
- Higher order coupling iterations represent 'effective' behaviour
- Exact solution, conserves energy, gives large-scale behaviour

1D Ising spin model

- Spins $\{v_i\}$ inline with spacing a; nearest neighbour coupling J^0
- Hamiltonian of system is $H(v) = -J^0 \sum_{i,i+1} v_i v_{i+1}$
- Coupling favours spins that agree
- Skip over every other spin, so that spacing is $2a$; solve for J^1
- Iterating RG turns out to satisfy $\tanh(J^{n+1}) = \tanh^2(J^n)$
- Higher order coupling iterations represent ‘effective’ behaviour
- Exact solution, conserves energy, gives large-scale behaviour
- Shortcoming: averaging discards half of the spins

Boltzmann machine

- General framework for neural computation

Boltzmann machine

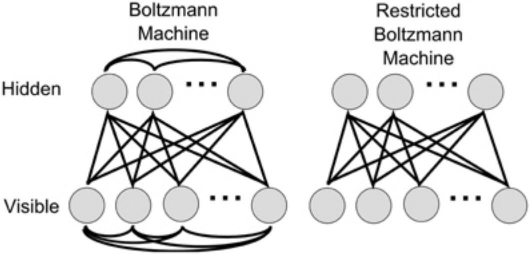
- General framework for neural computation
- Lossy compression

Boltzmann machine

- General framework for neural computation
- Lossy compression
- Binary, but extendible to multinomial and also real-valued

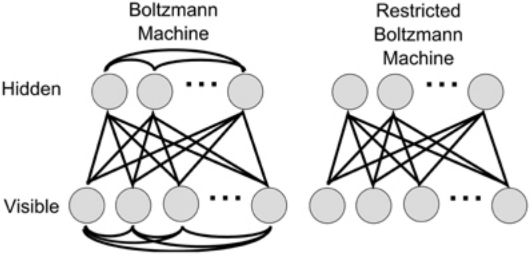
Boltzmann machine

- General framework for neural computation
- Lossy compression
- Binary, but extendible to multinomial and also real-valued



Boltzmann machine

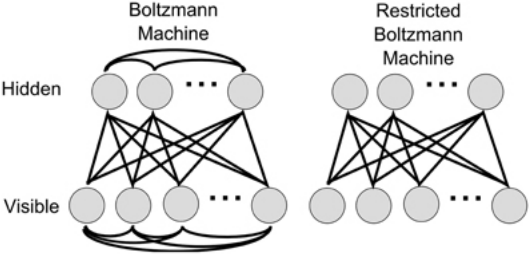
- General framework for neural computation
- Lossy compression
- Binary, but extendible to multinomial and also real-valued



- RBM visible layer: input; no intra-layer connections

Boltzmann machine

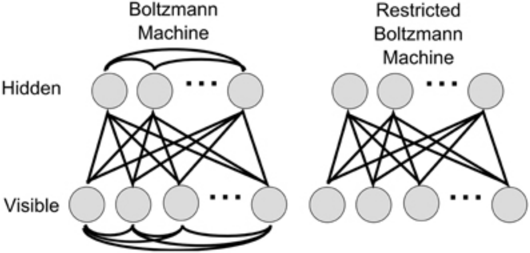
- General framework for neural computation
- Lossy compression
- Binary, but extendible to multinomial and also real-valued



- RBM visible layer: input; no intra-layer connections
- RBM hidden layer: feature detectors; no intra-layer connections

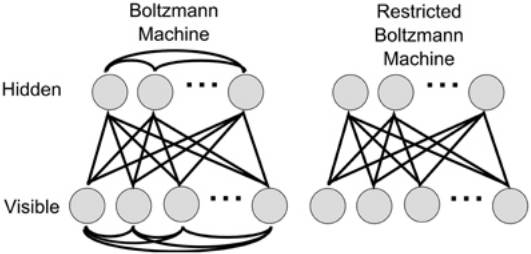
Boltzmann machine

- General framework for neural computation
- Lossy compression
- Binary, but extendible to multinomial and also real-valued

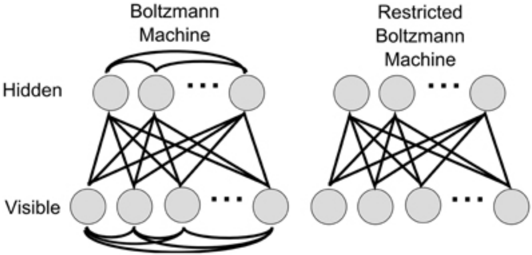


- RBM visible layer: input; no intra-layer connections
- RBM hidden layer: feature detectors; no intra-layer connections
- Layers are fully connected to each other

Restricted Boltzmann machine

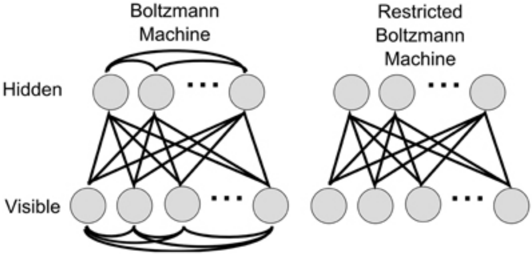


Restricted Boltzmann machine



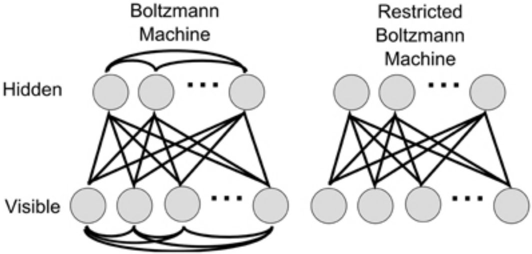
- Unsupervised (e.g. contrastive divergence or reconstruction)

Restricted Boltzmann machine



- Unsupervised (e.g. contrastive divergence or reconstruction)
- Model is stochastic: learns with what probability to turn a hidden node to $+1$ or -1 given some input

Restricted Boltzmann machine



- Unsupervised (e.g. contrastive divergence or reconstruction)
- Model is stochastic: learns with what probability to turn a hidden node to $+1$ or -1 given some input
- Learns probability distribution over its nodes by storing biases and weights related to the connections between nodes

Restricted Boltzmann machine

- Bias a_i goes with visible node v_i, b_j with h_j, and matrix entry w_{ij} with the connection between v_i and h_j

Restricted Boltzmann machine

- Bias a_i goes with visible node v_i, b_j with h_j, and matrix entry w_{ij} with the connection between v_i and h_j
- Central: energy (Hopfield 1982)
  $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$

Restricted Boltzmann machine

- Bias a_i goes with visible node v_i, b_j with h_j, and matrix entry w_{ij} with the connection between v_i and h_j
- Central: energy (Hopfield 1982)
  $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Boltzmann probability of configuration/state/observation v

$$p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

Restricted Boltzmann machine

- Bias a_i goes with visible node v_i, b_j with h_j, and matrix entry w_{ij} with the connection between v_i and h_j
- Central: energy (Hopfield 1982)
  $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Boltzmann probability of configuration/state/observation v

$$p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$, sum over all possible configurations— intractable (Long and Servedio, 2010)

Restricted Boltzmann machine

- Bias a_i goes with visible node v_i, b_j with h_j, and matrix entry w_{ij} with the connection between v_i and h_j
- Central: energy (Hopfield 1982)
  $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Boltzmann probability of configuration/state/observation v

$$p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$, sum over all possible configurations— intractable (Long and Servedio, 2010)
- No intra-layer connections: probability of hidden neuron given visible layer is easy (independent of other hidden neurons)

Restricted Boltzmann machine

- Bias a_i goes with visible node v_i, b_j with h_j, and matrix entry w_{ij} with the connection between v_i and h_j
- Central: energy (Hopfield 1982)
  $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Boltzmann probability of configuration/state/observation v

$$p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$, sum over all possible configurations— intractable (Long and Servedio, 2010)
- No intra-layer connections: probability of hidden neuron given visible layer is easy (independent of other hidden neurons)
- Probability of visible neuron given hidden layer just as well

Restricted Boltzmann machine

- Bias a_i goes with visible node v_i, b_j with h_j, and matrix entry w_{ij} with the connection between v_i and h_j
- Central: energy (Hopfield 1982)
  $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Boltzmann probability of configuration/state/observation v

$$p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$, sum over all possible configurations— intractable (Long and Servedio, 2010)
- No intra-layer connections: probability of hidden neuron given visible layer is easy (independent of other hidden neurons)
- Probability of visible neuron given hidden layer just as well
- Model is stochastic, but $p(v \mid h)$ and $p(h \mid v)$ are easy

Classification with an RBM

- RBMs learn a distribution over the training set

Classification with an RBM

- RBMs learn a distribution over the training set
- Boltzmann probabilities are assigned by the machine and are trained to fit the data

Classification with an RBM

- RBMs learn a distribution over the training set
- Boltzmann probabilities are assigned by the machine and are trained to fit the data
- How to do classification?

Classification with an RBM

- RBMs learn a distribution over the training set
- Boltzmann probabilities are assigned by the machine and are trained to fit the data
- How to do classification?
- Different suggestions by Hinton (2012)

Classification with an RBM

- RBMs learn a distribution over the training set
- Boltzmann probabilities are assigned by the machine and are trained to fit the data
- How to do classification?
- Different suggestions by Hinton (2012)
  - (a) Use hidden layer to train normal classifier (arguably most important)

Classification with an RBM

- RBMs learn a distribution over the training set
- Boltzmann probabilities are assigned by the machine and are trained to fit the data
- How to do classification?
- Different suggestions by Hinton (2012)
  - (a) Use hidden layer to train normal classifier (arguably most important)
  - (b) Train an RBM for each class and use class-specific free energy and (ML-approximation of) partition function in a softmax

Classification with an RBM

- RBMs learn a distribution over the training set
- Boltzmann probabilities are assigned by the machine and are trained to fit the data
- How to do classification?
- Different suggestions by Hinton (2012)
  - (a) Use hidden layer to train normal classifier (arguably most important)
  - (b) Train an RBM for each class and use class-specific free energy and (ML-approximation of) partition function in a softmax
  - (c) Include label in visible layer during training, so that RBM learns the probability of a class, and then evaluate joint probabilities for a test vector with each of the classes—comparison is easy because partition function is the same

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$

Training an RBM

- Recall energy $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Training is maximising joint probability of training set

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Training is maximising joint probability of training set
- Joint probability is product of probabilities of observations

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Training is maximising joint probability of training set
- Joint probability is product of probabilities of observations
- Tune bias vectors a and b and weight matrix w

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Training is maximising joint probability of training set
- Joint probability is product of probabilities of observations
- Tune bias vectors a and b and weight matrix w
- Gradient ascent using individual probability for each observation

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$

Training an RBM

- Recall energy $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Probability of observation v is $p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Probability of observation v is $p(v) = \frac{1}{Z}\sum_h e^{-E(v,h)}$
- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Probability of observation v is $p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$
- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$
- Maximise individual (log-)probability by gradient ascent with

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Probability of observation v is $p(v) = \frac{1}{Z}\sum_h e^{-E(v,h)}$
- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$
- Maximise individual (log-)probability by gradient ascent with

$$\begin{aligned}\frac{\delta \log p(v)}{\delta w_{ij}} &= \frac{\delta}{\delta w_{ij}} \log \sum_h e^{-E(v,h)} - \frac{\delta}{\delta w_{ij}} \log Z \\ &= \frac{\sum_h e^{-E(v,h)} v_i h_j}{\sum_h e^{-E(v,h)}} - \frac{\sum_{\tilde{v},h} e^{-E(\tilde{v},h)} \tilde{v}_i h_j}{\sum_{\tilde{v},h} e^{-E(\tilde{v},h)}} \\ &= \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]\end{aligned}$$

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Probability of observation v is $p(v) = \frac{1}{Z}\sum_h e^{-E(v,h)}$
- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$
- Maximise individual (log-)probability by gradient ascent with

$$\begin{aligned}\frac{\delta \log p(v)}{\delta w_{ij}} &= \frac{\delta}{\delta w_{ij}} \log \sum_h e^{-E(v,h)} - \frac{\delta}{\delta w_{ij}} \log Z \\ &= \frac{\sum_h e^{-E(v,h)} v_i h_j}{\sum_h e^{-E(v,h)}} - \frac{\sum_{\tilde{v},h} e^{-E(\tilde{v},h)} \tilde{v}_i h_j}{\sum_{\tilde{v},h} e^{-E(\tilde{v},h)}} \\ &= \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]\end{aligned}$$

- Expectations under model distribution with current weights

Training an RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- Probability of observation v is $p(v) = \frac{1}{Z}\sum_h e^{-E(v,h)}$
- Partition function $Z = \sum_{v,h} e^{-E(v,h)}$
- Maximise individual (log-)probability by gradient ascent with

$$\begin{aligned}\frac{\delta \log p(v)}{\delta w_{ij}} &= \frac{\delta}{\delta w_{ij}} \log \sum_h e^{-E(v,h)} - \frac{\delta}{\delta w_{ij}} \log Z \\ &= \frac{\sum_h e^{-E(v,h)} v_i h_j}{\sum_h e^{-E(v,h)}} - \frac{\sum_{\tilde{v},h} e^{-E(\tilde{v},h)} \tilde{v}_i h_j}{\sum_{\tilde{v},h} e^{-E(\tilde{v},h)}} \\ &= \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]\end{aligned}$$

- Expectations under model distribution with current weights
- Can gradient be computed?

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Model is stochastic

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Model is stochastic
- Conditional on v, the hidden activations are independent and readily computed

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Model is stochastic
- Conditional on v, the hidden activations are independent and readily computed
- Given v, hidden activation h_j is 1 with probability

$$p(h_j \mid v_i) = \sigma\left(b_j + \sum_i v_i w_{ij}\right) \text{ with } \sigma(x) = \frac{1}{1 + e^{-x}}$$

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Model is stochastic
- Conditional on v, the hidden activations are independent and readily computed
- Given v, hidden activation h_j is 1 with probability

$$p(h_j \mid v_i) = \sigma\left(b_j + \sum_i v_i w_{ij}\right) \text{ with } \sigma(x) = \frac{1}{1 + e^{-x}}$$

- This follows from the Boltzmann distribution

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Model is stochastic
- Conditional on v, the hidden activations are independent and readily computed
- Given v, hidden activation h_j is 1 with probability

$$p(h_j \mid v_i) = \sigma\left(b_j + \sum_i v_i w_{ij}\right) \text{ with } \sigma(x) = \frac{1}{1+e^{-x}}$$

- This follows from the Boltzmann distribution
- So an unbiased estimate of $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Model is stochastic
- Conditional on v, the hidden activations are independent and readily computed
- Given v, hidden activation h_j is 1 with probability

$$p(h_j \mid v_i) = \sigma\left(b_j + \sum_i v_i w_{ij}\right) \text{ with } \sigma(x) = \frac{1}{1+e^{-x}}$$

- This follows from the Boltzmann distribution
- So an unbiased estimate of $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy
- But what about $\mathbf{E}\left[v_i h_j\right]$?

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Computing unconditional $\mathbf{E}\left[v_i h_j\right]$ is much harder

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Computing unconditional $\mathbf{E}\left[v_i h_j\right]$ is much harder
- Option: Gibbs sampling, since $p(v \mid h)$ and $p(h \mid v)$ are easy

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}[v_i h_j \mid v] - \mathbf{E}[v_i h_j]$
- Computing unconditional $\mathbf{E}[v_i h_j]$ is much harder
- Option: Gibbs sampling, since $p(v \mid h)$ and $p(h \mid v)$ are easy
- Both are logistic sigmoids

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Computing unconditional $\mathbf{E}\left[v_i h_j\right]$ is much harder
- Option: Gibbs sampling, since $p(v \mid h)$ and $p(h \mid v)$ are easy
- Both are logistic sigmoids
- Start with random v, sample h, sample v, sample h, repeat...

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Computing unconditional $\mathbf{E}\left[v_i h_j\right]$ is much harder
- Option: Gibbs sampling, since $p(v \mid h)$ and $p(h \mid v)$ are easy
- Both are logistic sigmoids
- Start with random v, sample h, sample v, sample h, repeat...
- After a 'while', both v and h follow joint (unconditional) distribution

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Computing unconditional $\mathbf{E}\left[v_i h_j\right]$ is much harder
- Option: Gibbs sampling, since $p(v \mid h)$ and $p(h \mid v)$ are easy
- Both are logistic sigmoids
- Start with random v, sample h, sample v, sample h, repeat...
- After a 'while', both v and h follow joint (unconditional) distribution
- This could take a while

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Conditional $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy, unconditional $\mathbf{E}\left[v_i h_j\right]$ is harder

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Conditional $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy, unconditional $\mathbf{E}\left[v_i h_j\right]$ is harder
- Gibbs sampling is expensive

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Conditional $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy, unconditional $\mathbf{E}\left[v_i h_j\right]$ is harder
- Gibbs sampling is expensive
- Faster: start with training vector v, sample h, reconstruct v'

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Conditional $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy, unconditional $\mathbf{E}\left[v_i h_j\right]$ is harder
- Gibbs sampling is expensive
- Faster: start with training vector v, sample h, reconstruct v'
- Use $v_i' h_j$ to approximate $\mathbf{E}\left[v_i h_j\right]$

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Conditional $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy, unconditional $\mathbf{E}\left[v_i h_j\right]$ is harder
- Gibbs sampling is expensive
- Faster: start with training vector v, sample h, reconstruct v'
- Use $v_i' h_j$ to approximate $\mathbf{E}\left[v_i h_j\right]$
- Looks like Contrastive Divergence, but not quite

Training an RBM

- Goal: find gradient $\frac{\delta \log p(v)}{\delta w_{ij}} = \mathbf{E}\left[v_i h_j \mid v\right] - \mathbf{E}\left[v_i h_j\right]$
- Conditional $\mathbf{E}\left[v_i h_j \mid v\right]$ is easy, unconditional $\mathbf{E}\left[v_i h_j\right]$ is harder
- Gibbs sampling is expensive
- Faster: start with training vector v, sample h, reconstruct v'
- Use $v_i' h_j$ to approximate $\mathbf{E}\left[v_i h_j\right]$
- Looks like Contrastive Divergence, but not quite
- Works better if alternated a couple of times

Training an RBM

- To summarise RBM training, let ε be learning rate

Training an RBM

- To summarise RBM training, let ε be learning rate
- Update weights with $\Delta w_{ij} = \varepsilon \left(\hat{\mathbf{E}} \left[v_i h_j \mid v \right] - \hat{\mathbf{E}} \left[v_i h_j \right] \right)$

Training an RBM

- To summarise RBM training, let ε be learning rate
- Update weights with $\Delta w_{ij} = \varepsilon \left(\hat{\mathbf{E}} \left[v_i h_j \mid v \right] - \hat{\mathbf{E}} \left[v_i h_j \right] \right)$
- Sample $\hat{\mathbf{E}} \left[v_i h_j \mid v \right]$ from input v and one-time stochastic activation h

Training an RBM

- To summarise RBM training, let ε be learning rate
- Update weights with $\Delta w_{ij} = \varepsilon \left(\hat{\mathbf{E}} \left[v_i h_j \mid v \right] - \hat{\mathbf{E}} \left[v_i h_j \right] \right)$
- Sample $\hat{\mathbf{E}} \left[v_i h_j \mid v \right]$ from input v and one-time stochastic activation h
- Sample $\hat{\mathbf{E}} \left[v_i h_j \right]$ from reconstruction v' of v via h

Training an RBM

- To summarise RBM training, let ε be learning rate
- Update weights with $\Delta w_{ij} = \varepsilon \left(\hat{\mathbf{E}} \left[v_i h_j \mid v \right] - \hat{\mathbf{E}} \left[v_i h_j \right] \right)$
- Sample $\hat{\mathbf{E}} \left[v_i h_j \mid v \right]$ from input v and one-time stochastic activation h
- Sample $\hat{\mathbf{E}} \left[v_i h_j \right]$ from reconstruction v' of v via h
- Obtain $\Delta w_{ij} = \varepsilon \left(v_j h_j - v'_i h_j \right)$

Training an RBM

- To summarise RBM training, let ε be learning rate
- Update weights with $\Delta w_{ij} = \varepsilon \left(\hat{\mathbf{E}} \left[v_i h_j \mid v \right] - \hat{\mathbf{E}} \left[v_i h_j \right] \right)$
- Sample $\hat{\mathbf{E}} \left[v_i h_j \mid v \right]$ from input v and one-time stochastic activation h
- Sample $\hat{\mathbf{E}} \left[v_i h_j \right]$ from reconstruction v' of v via h
- Obtain $\Delta w_{ij} = \varepsilon \left(v_j h_j - v_i' h_j \right)$
- Biologically plausible (?)

Training an RBM

- To summarise RBM training, let ε be learning rate
- Update weights with $\Delta w_{ij} = \varepsilon \left(\hat{\mathbf{E}} \left[v_i h_j \mid v \right] - \hat{\mathbf{E}} \left[v_i h_j \right] \right)$
- Sample $\hat{\mathbf{E}} \left[v_i h_j \mid v \right]$ from input v and one-time stochastic activation h
- Sample $\hat{\mathbf{E}} \left[v_i h_j \right]$ from reconstruction v' of v via h
- Obtain $\Delta w_{ij} = \varepsilon \left(v_j h_j - v_i' h_j \right)$
- Biologically plausible (?)
- Does not approximate gradient, but works well (Hinton 2012)

Training an RBM

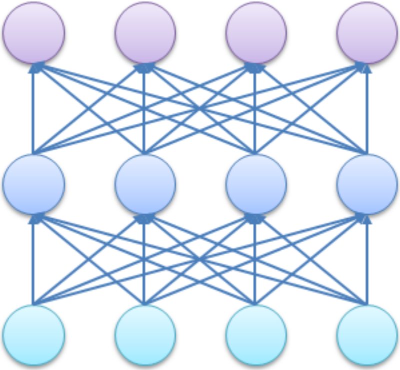
- To summarise RBM training, let ε be learning rate
- Update weights with $\Delta w_{ij} = \varepsilon \left(\hat{\mathbf{E}} [v_i h_j \mid v] - \hat{\mathbf{E}} [v_i h_j] \right)$
- Sample $\hat{\mathbf{E}} [v_i h_j \mid v]$ from input v and one-time stochastic activation h
- Sample $\hat{\mathbf{E}} [v_i h_j]$ from reconstruction v' of v via h
- Obtain $\Delta w_{ij} = \varepsilon (v_j h_j - v_i' h_j)$
- Biologically plausible (?)
- Does not approximate gradient, but works well (Hinton 2012)
- Bias vectors a and b are updated similarly

Stacking RBMs

- Hidden layer of first level is visible layer of the next, and so on

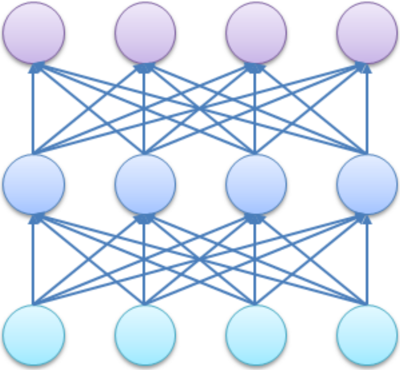
Stacking RBMs

- Hidden layer of first level is visible layer of the next, and so on



Stacking RBMs

- Hidden layer of first level is visible layer of the next, and so on



- Reducing dimensionality: lossy compression

Renormalisation of stacked RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$

Renormalisation of stacked RBM

- Recall energy $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v

Renormalisation of stacked RBM

- Recall energy $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v
- Less h_j than v_i

Renormalisation of stacked RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v
- Less h_j than v_i
- RG yields transformation of energy in terms of h only: $\tilde{E}(h)$

Renormalisation of stacked RBM

- Recall energy $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v
- Less h_j than v_i
- RG yields transformation of energy in terms of h only: $\tilde{E}(h)$
- Equivalent to the way the Hamiltonian is rewritten in physics

Renormalisation of stacked RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v
- Less h_j than v_i
- RG yields transformation of energy in terms of h only: $\tilde{E}(h)$
- Equivalent to the way the Hamiltonian is rewritten in physics
- At the same time, Boltzmann probabilities of hidden layer may be marginalised over the visible layer, yielding an energy function defined by marginal probabilities $p(h)$

Renormalisation of stacked RBM

- Recall energy $E(v, h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v
- Less h_j than v_i
- RG yields transformation of energy in terms of h only: $\tilde{E}(h)$
- Equivalent to the way the Hamiltonian is rewritten in physics
- At the same time, Boltzmann probabilities of hidden layer may be marginalised over the visible layer, yielding an energy function defined by marginal probabilities $p(h)$
- Mehta and Schwab show that both energies are the same

Renormalisation of stacked RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v
- Less h_j than v_i
- RG yields transformation of energy in terms of h only: $\tilde{E}(h)$
- Equivalent to the way the Hamiltonian is rewritten in physics
- At the same time, Boltzmann probabilities of hidden layer may be marginalised over the visible layer, yielding an energy function defined by marginal probabilities $p(h)$
- Mehta and Schwab show that both energies are the same
- This means that stacked RBM feature extraction employs RG

Renormalisation of stacked RBM

- Recall energy $E(v,h) = -\left(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} v_i h_j\right)$
- RBMs use hidden neurons h to model data in visible neurons v
- Less h_j than v_i
- RG yields transformation of energy in terms of h only: $\tilde{E}(h)$
- Equivalent to the way the Hamiltonian is rewritten in physics
- At the same time, Boltzmann probabilities of hidden layer may be marginalised over the visible layer, yielding an energy function defined by marginal probabilities $p(h)$
- Mehta and Schwab show that both energies are the same
- This means that stacked RBM feature extraction employs RG
- Understanding how stacked RBMs synthesise features gives insight in why and when they work

Numerical experiment

- To see this in action: 2D Ising model

Numerical experiment

- To see this in action: 2D Ising model
- Training set of spins with Hamiltonian $H(v) = -J \sum_{\langle i,j \rangle} v_i v_j$

Numerical experiment

- To see this in action: 2D Ising model
- Training set of spins with Hamiltonian $H(v) = -J \sum_{\langle i,j \rangle} v_i v_j$
- Nearest neighbours only $\langle i, j \rangle$

Numerical experiment

- To see this in action: 2D Ising model
- Training set of spins with Hamiltonian $H(v) = -J \sum_{\langle i,j \rangle} v_i v_j$
- Nearest neighbours only $\langle i, j \rangle$
- Dimensionality $1600 \rightarrow 400 \rightarrow 100 \rightarrow 25$

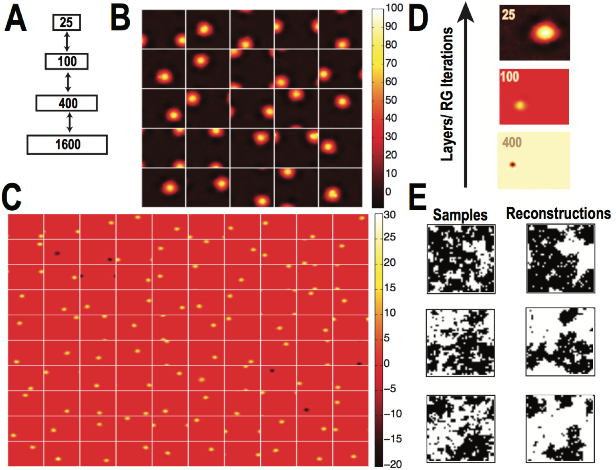
Numerical experiment

- To see this in action: 2D Ising model
- Training set of spins with Hamiltonian $H(v) = -J \sum_{\langle i,j \rangle} v_i v_j$
- Nearest neighbours only $\langle i, j \rangle$
- Dimensionality $1600 \rightarrow 400 \rightarrow 100 \rightarrow 25$
- Appears to learn Kadanoff block spin renormalisation by itself

Numerical experiment

- To see this in action: 2D Ising model
- Training set of spins with Hamiltonian $H(v) = -J \sum_{\langle i,j \rangle} v_i v_j$
- Nearest neighbours only $\langle i, j \rangle$
- Dimensionality $1600 \to 400 \to 100 \to 25$
- Appears to learn Kadanoff block spin renormalisation by itself
- Stacked RBM learns spatiality without imposing it

Numerical experiment



Summary

- RG is an abstract, powerful technique from statistical physics

Summary

- RG is an abstract, powerful technique from statistical physics
- Kadanoff's block spin renormalisation works for binary configurations

Summary

- RG is an abstract, powerful technique from statistical physics
- Kadanoff's block spin renormalisation works for binary configurations
- Stacked RBMs, which learn a distribution without supervision, automatically implement this renormalisation

Summary

- RG is an abstract, powerful technique from statistical physics
- Kadanoff's block spin renormalisation works for binary configurations
- Stacked RBMs, which learn a distribution without supervision, automatically implement this renormalisation
- Theoretical insight may bring clarification about why deep learning recognises features so well

Conclusions and implications

- Interesting perspective

Conclusions and implications

- Interesting perspective
- Perhaps statistical physics will yield more insight into DL

Conclusions and implications

- Interesting perspective
- Perhaps statistical physics will yield more insight into DL
- Physics are typically very symmetric, while data is not

Conclusions and implications

- Interesting perspective
- Perhaps statistical physics will yield more insight into DL
- Physics are typically very symmetric, while data is not
- Relevant: critical temperature to operate near phase transition

Conclusions and implications

- Interesting perspective
- Perhaps statistical physics will yield more insight into DL
- Physics are typically very symmetric, while data is not
- Relevant: critical temperature to operate near phase transition
- No breakthrough follow-up yet

References

- Hopfield (1982) - Neural networks and physical systems with emergent collective computational abilities
- Hinton (2002) - Training products of experts by minimizing contrastive divergence
- Hinton, Salakhutdinov (2006) - Reducing the dimensionality of data with neural networks
- Hinton (2012) - A practical guide to training restricted Boltzmann machines
- Kadanoff, Houghton, and Yalabik (1976) - Variational approximations for renormalization group transformations
- McComb (2004) - Renormalization methods, a guide for beginners
- Mehta and Schwab (2014) - An exact mapping between the variational renormalization group and deep learning

Matthieu Courbariaux & Yoshua Bengio
BinaryNet :
Training Deep Neural Networks with Weights and Activations Constrained to $+1$ or -1

Francesco Stablum

17th March 2016

Introduction

- Method focused on computational optimization and implementation details
- Based on well-known MLPs and ConvNets
- Reduces memory usage
- Reduces number of instructions

How? Binarization

- Weights and activations are constrained to have values either -1 or $+1$
- Binarization function $x^b = \mathrm{Sign}(x)$
- Multiplications replaced with 1-bit XNOR operations

Gradients and noise

Although the weights are binary, the gradient is real-valued.

- SGD makes small and noisy steps to explore the space of parameters
- noise is averaged out by the stochastic gradient contributions
- noise to weights and activations when computing the gradient can can act as regularization
- Binarization, being a form of quantization, adds noise

Propagating Gradients Trough Discretization

Problem

The derivative of $q = \mathrm{Sign}(r)$ is always 0

Solution: Straight-Trough Gradient Estimator (Hinton)

- estimator $g_q = \frac{\partial C}{\partial q}$ is assumed to be obtained
- straight-trough estimator $g_r = \frac{\partial C}{\partial r} = g_q 1_{|r| \leq 1}$

the derivative $1_{|r| \leq 1}$ can be seen as propagating the gradient trough *hard tanh*, that is:

$$\mathrm{Htanh}(x) = \mathrm{Clip}(x, -1, +1) = \max(-1, \min(1, x)) \qquad (1)$$

A few helpful ingredients

- Reduction of the impact of the weights' scale achieved by:
    1. Batch normalization (that also accelerates the training)
    2. ADAM learning rule

Observations

- Augmenting the number of hidden units can compensate for the discretization noise
- BinaryNet is faster to train than BinaryConnect but leads to worse results.
    - Maybe it's overfitting and might benefit from additional noise

Experiments: MLP on MNIST

- 3 hidden layers with 4096 binary units
- L2-SVM output layer
- Model regularization with Dropout
- ADAM
- Exponentially decaying global learning rate

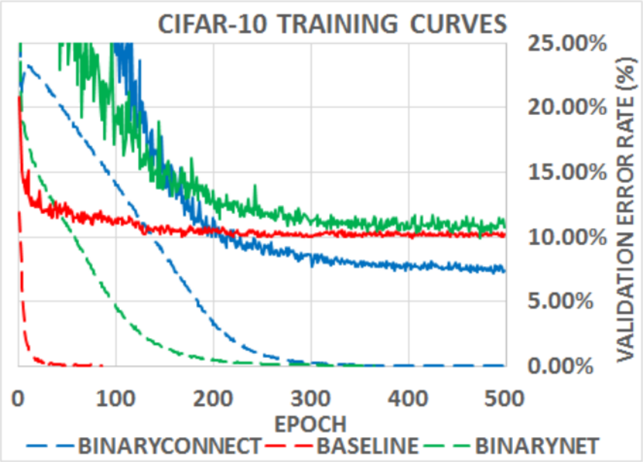
Experiments: ConvNet

On CIFAR-10

- No preprocessing
- Square hinge loss
- ADAM
- Exponentially decaying learning rate
- Batch normalization (minibatch size: 50)
- Validation set: last 5000 samples
- Amount of epochs: 500

On SVHN

- Configuration like on CIFAR-10
- Amount of epochs: 200



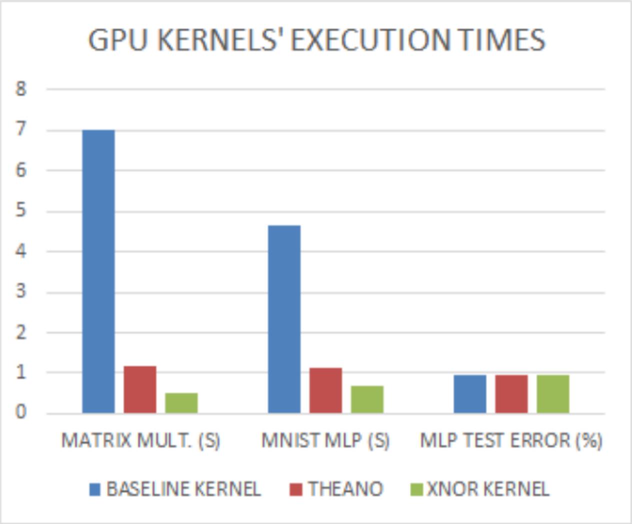
CIFAR-10 TRAINING CURVES
25.00%
20.00%
15.00%
10.00%
5.00%
0.00%
VALIDATION ERROR RATE (%)
0
100
200
300
400
500
EPOCH
BINARYCONNECT
BASELINE
BINARYNET

Performance improvement via XNOR-accumulate

By using GPU:

- SIMD: Single Instruction, Multiple Data
- SWAR: SIMD In A Register:
  - Concatenates groups of 32 binary variable in a 32-bit register
  - This way, 32 connections evaluated with only 4 instructions: $a_1 += \mathrm{popcount}(\mathrm{not}(\mathrm{xor}(a_0^{32b}, w_1)))$

GPU Execution Times



Related works

Binary Connect

- binary weights
- Some activations quantizations
- slower to train
- worse on MNIST
- better on CIFAR-10
- good with fully connected networks, not good with ConvNets

Hwang & Sung, 2014; Kim, 2014

- Network is trained with high precision
- Afterwards, the weights are ternarized $-H, 0, +H$
- re-training with ternary weights and 3-bit activations
- good for fully connected networks, not good with ConvNets

Future works

- Binarization of the gradients
- Benchmark results to other models (e.g. RNN) and datasets