Deep networks in uncertainty quantification

Artificial Neural Networks and machine learning

David K. E. Green
d.k.green@unsw.edu.au

School of Civil and Environmental Engineering
University of New South Wales
Sydney, Australia

The Alan Turing Institute

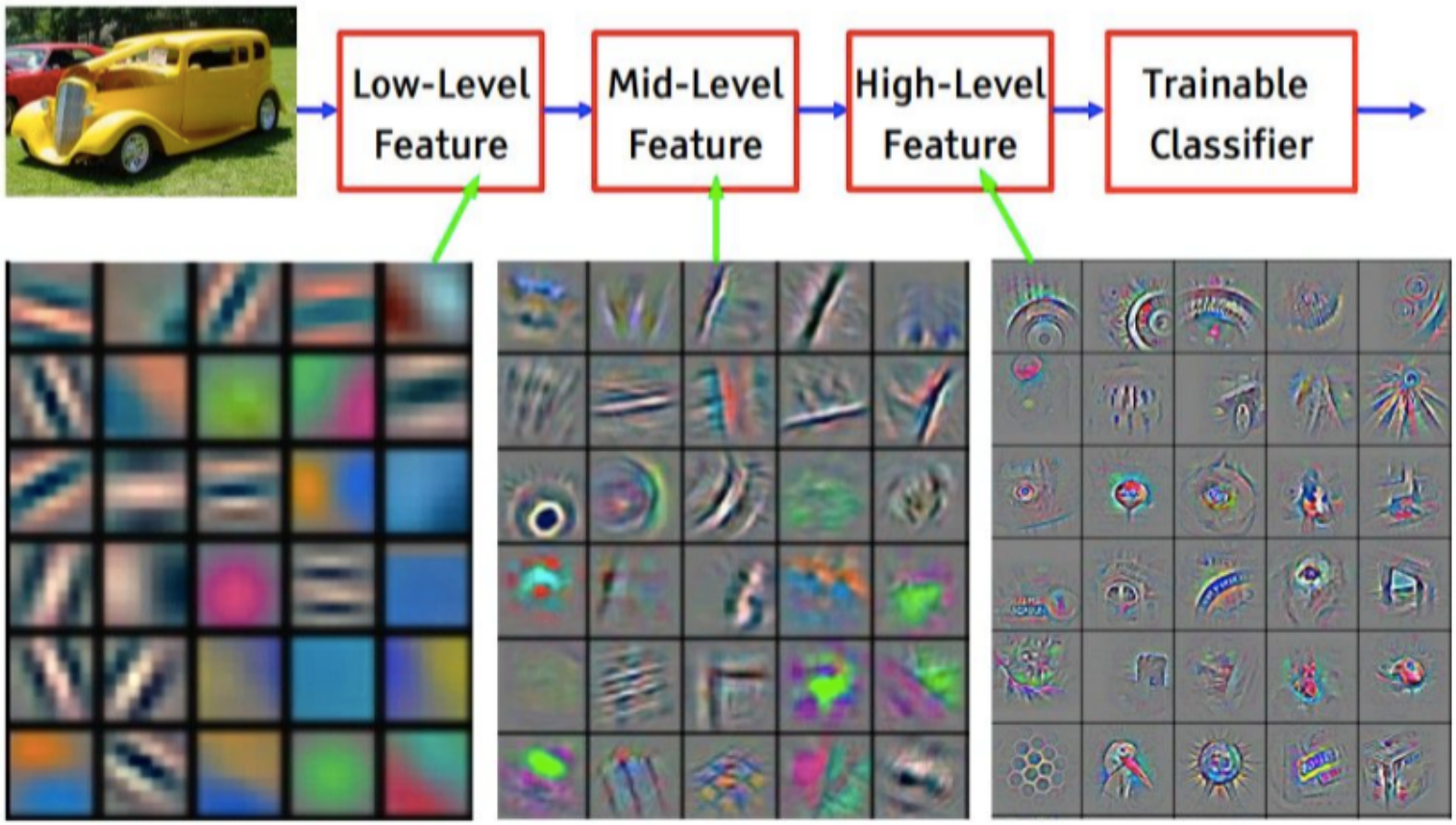
London, United Kingdom
www.turing.ac.uk

UNSW
AUSTRALIA

The Alan Turing Institute

Artificial Neural Networks - Deep Networks

Deep networks learn hierarchical basis functions - efficient encodings



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Image: Yann LeCunn `yann.lecun.com`

Overview

- **Machine learning:** Problem classification areas.
- **Uncertainty Quantification for engineering:** Combining probabilistic techniques with numerical methods.
- **Deep Artificial Neural Networks:** Powerful parametric optimisation methods. Background material.
- **Adaptive Basis Element Free Galerkin:** Adaptive Basis Element Free Galerkin algorithm derived by considering Probabilistic Numerical Methods, the Expectation-Maximisation algorithm, Sparse-Coding, Gaussian Processes and Bayesian Linear Regression.
- **Future directions:** Variational Bayesian Inference and Generative Models.

Supervised learning

- Given: input/output pairs - x and $f(x)$.
- Classification: $f(x)$ is a discrete class vector, i.e. in $\mathbb{N}$.
  - Support Vector Machines (SVM)
  - Artificial Neural Networks
  - ANN Energy based models
- Regression: Learn weights for function $h_\theta(x) \in \mathbb{R}^N$
  - Kernel Methods/Support Vector Regression (SVR)
  - Artificial Neural Networks

Unsupervised learning

- Given: data set $x \in X$
- Task: learn efficient representations of the data
- Examples:
  - Principle Components Analysis (PCA)
  - Clustering
  - Anomaly detection
  - Restricted Boltzmann Machines
  - Autoencoders
  - Variational Autoencoders (VAE)
  - Generative Adversarial Networks (GANs)

Descriminative and Generative Models for classification

- Given:
  - Inputs - x
  - Labels - y
- Generative models:
  - Build model $P(x, y)$
  - Use Bayes rule to calculate $P(y|x)$ and pick maximum likelihood y
  - Example - Naïve Bayes
- Descriminative models:
  - Learn $P(y|x)$ directly
  - Equivalent - learn map $y = f(x)$
  - Logistic regression
- Good reference: Ng & Jordan, 2001, `https://tinyurl.com/y72r2hsk`

Uncertainty Quantification - From a learning perspective

- Given:
  - $P(x)$ - input data distribution
  - $P(y|x)$ - map from inputs to outputs
- Tasks:
  - Learn output distribution $P(y)$
  - Learn QoI $A = \mathbb{E}_{P(y)}[a(y)]$
- Issues:
  - $P(x)$ - usually ok, KL-Expansion or etc
  - $P(y|x)$ - very complicated (e.g. finite element output)
  - Output distribution $P(y)$ has no obvious nice analytic form
- Challenges:
  - Solving high dimensional integrals
  - Encoding complicated distributions
  - Computationally expensive sampling/mapping

Artificial Neural Networks - parametric function approximation

Goal: Given a function $f(x)$, and error functional $L[h(x), f(x)]$ find closest approximation from hypothesis space $\mathcal{H}$.

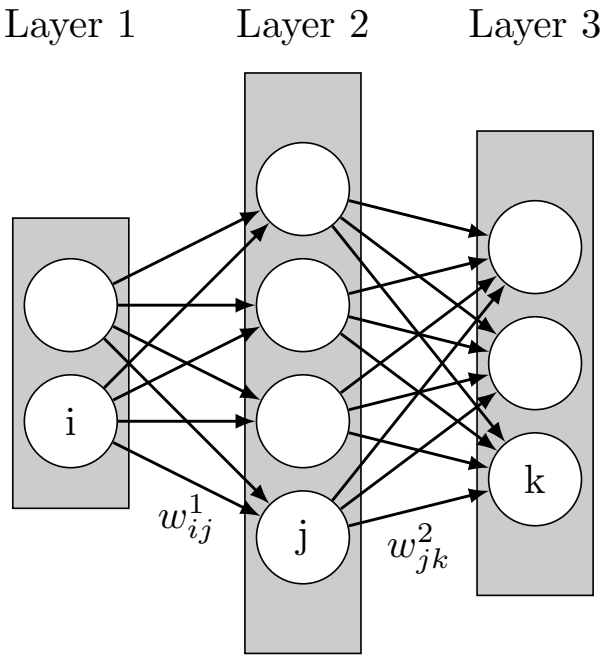
$$h^*(x) = \underset{h \in \mathcal{H}}{\text{argmin}}\ \mathbb{E}[h(x), f(x)]$$

Deep ANN's compute a series of activation functions, $\sigma(\circ)$ that depend on the weighted sum of inputs to a layer. For an N layer network with input vector $x \in \mathbb{R}^N$ and weight matrices $\theta_i \in \mathbb{R}^{nm}$, $h(x)$ is given by:

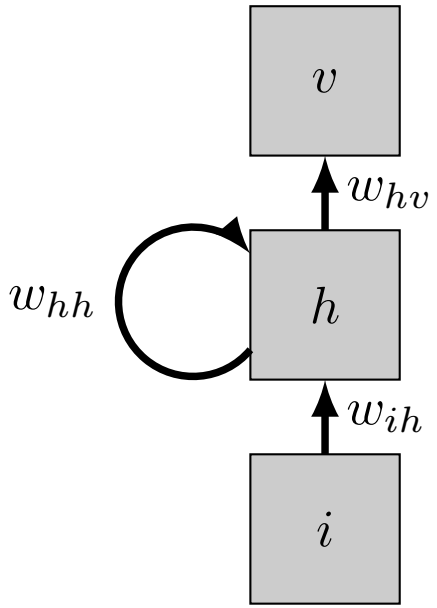
$$\begin{aligned} a_0 &= x \\ a_i &= \sigma_i(\theta_i a_{i-1}) \\ h_\theta(x) &= a_N \end{aligned}$$

What is $\mathcal{H}$? The space of all possible Neural Network weights.

Information flows - Feedforward and Recurrent nets



Feedforward architecture



Recurrent architecture

Information stored in the weights. Research area: use external memory e.g. Neural Turing Machines and Differentiable Neural Computers.

Why did Artificial Neural Networks suddenly become popular?

What went wrong?

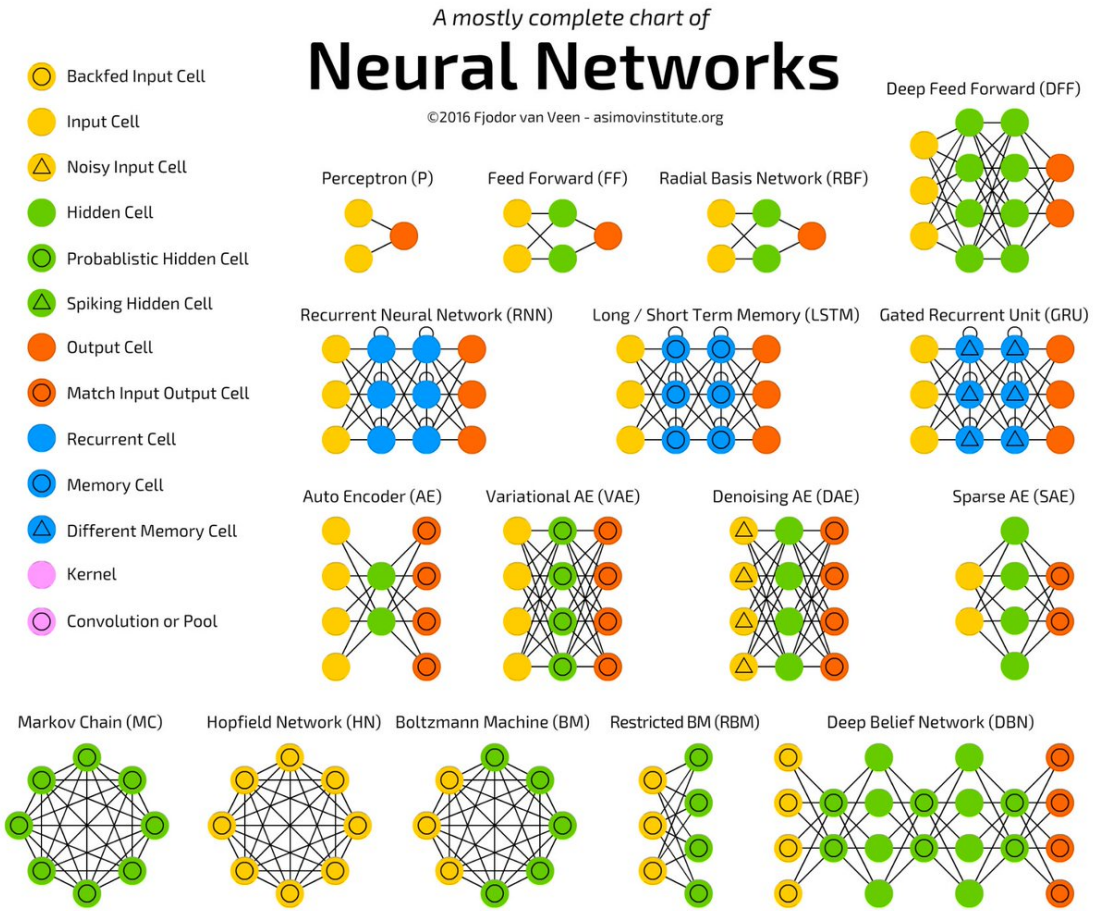
- Early promise of ANNs was not realised, leading to community resistance.
- Computers too weak.
- Vanishing gradient problem prevented training of deep networks.

Why is deep learning now so successful?

- Computers can now hold big models (many weights).
- Networks can be trained more quickly (GPUs).
- Developments in activation functions have helped to resolve the *vanishing gradient problem*.
- Adaptive training algorithms (improving Stochastic Gradient Descent).
- Benchmark gains led to huge development surge in last $\sim$5 years.

Artificial Neural Networks - general architecture

- ANNs are graphs with information flows
- Neural Network Zoo: `http://www.asimovinstitute.org/neural-network-zoo/`



A small sample from the full set!

Artificial Neural Networks - optimisation

- The loss functional $J(\theta) = L\left[h_\theta(x), f(x)\right]$ defines a surface on $\mathcal{H}$.
- Typically Artificial Neural Networks learn local optima on $\mathcal{H}$ by Stochastic Gradient Descent.
- Optimisation by Stochastic Gradient Descent:

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta J(\theta_i)$$

- Backpropagation uses the chain rule to calculate the local change in ANN weights to minimise the error:

$$\nabla_\theta J(\theta_i)$$

- Numerically, errors are calculated for training data sets (know input, x, and output, $f(x)$, pairs).

Artificial Neural Networks - automatic differentiation

Models become complicated. Use automatic differentiation!
Automatic differentiation:

- Use dual number representation of operations:

$$x + x'\epsilon$$

where (basic operations):

$$\left(x + x'\epsilon\right) + \left(y + y'\epsilon\right) = \left(x + y + \left(x' + y'\right)\right)\epsilon$$
$$\left(x + x'\epsilon\right) \times \left(y + y'\epsilon\right) = xy + xy'\epsilon + yx'\epsilon + x'y'\epsilon^2 = xy + \left(xy' + yx'\right)\epsilon$$

- Allows polynomials to be defined
- Can define derivatives for standard functions e.g. $\exp, \sin$ etc.
- Extension to multivariate models, other operations
- See: Rall, 1981, Automatic Differentiation: Techniques and Applications, doi:10.1007/3-540-10861-0

Artificial Neural Networks - automatic differentiation

$$\text{Write } x + x'\epsilon := \langle x, x' \rangle$$

$$g(\langle x, x' \rangle, \langle y, y' \rangle) = \langle g(x, y), \partial_x(g(x, y))x' + \partial_y(g(x, y))y' \rangle$$

$$\langle x, x' \rangle + \langle y, y' \rangle = \langle x + y, x' + y' \rangle$$

$$\langle x, x' \rangle - \langle y, y' \rangle = \langle x - y, x' - y' \rangle$$

$$\langle x, x' \rangle \times \langle y, y' \rangle = \langle xy, x'y + y'x \rangle$$

$$\langle x, x' \rangle \times \langle y, y' \rangle = \langle \frac{x}{y}, \frac{x'y - y'x}{y^2} \rangle$$

$$\langle x, x' \rangle / \langle y, y' \rangle = \langle \frac{x}{y}, \frac{x'y - y'x}{y^2} \rangle \ (y \neq 0)$$

$$\sin\langle x, x' \rangle = \langle \sin(x), x' \cos(x) \rangle$$

$$\cos\langle x, x' \rangle = \langle \cos(x), -x' \sin(x) \rangle$$

$$\exp\langle x, x' \rangle = \langle \exp(x), -x' \exp(x) \rangle$$

$$\log\langle x, x' \rangle = \langle \log(x), \frac{x'}{x} \rangle \ (x > 0)$$

$$\langle x, x' \rangle^k = \langle x^k, kx^{k-1}x' \rangle \ (x \neq 0)$$

and so on...

Artificial Neural Networks - reverse mode AD

Consider:

$$y = f(x) = f(g(x))$$

then:

$$\frac{dy}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

For composed functions:

$$y = f(x; w)$$

$$\frac{dy}{dx} = \frac{df}{dw_1}\frac{dw_1}{dx} = \left(\frac{df}{dw_2}\frac{dw_2}{dw_1}\right)\frac{dw_1}{dx} = \left(\left(\frac{df}{dw_3}\frac{dw_3}{dw_2}\right)\frac{dw_2}{dw_1}\right)\frac{dw_1}{dx}$$

Artificial Neural Networks - Jacobian chain rule

In higher dimensions:

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^k$$
$$g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$$
$$x \in \mathbb{R}^n$$

Jacobian chain rule in index form:

$$\frac{\partial(f_1, \cdots, f_k)}{\partial(x_1, \cdots, x_n)} = \frac{\partial(f_1, \cdots, f_k)}{\partial(g_1, \cdots, g_m)} \frac{\partial(g_1, \cdots, g_m)}{\partial(x_1, \cdots, x_n)}$$
$$\frac{\partial(f_1, \cdots, f_k)}{\partial x_i} = \frac{\partial(f_1, \cdots, f_k)}{\partial(g_1, \cdots, g_m)} \frac{\partial(g_1, \cdots, g_m)}{\partial x_i}$$
$$\frac{\partial(f_1, \cdots, f_k)}{\partial x_i} = \sum_{l=1}^{m} \frac{\partial(f_1, \cdots, f_k)}{\partial g_l} \frac{\partial g_l}{\partial x_i}$$

Artificial Neural Networks - Jacobian chain rule

In higher dimensions:

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^k$$
$$g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$$
$$x \in \mathbb{R}^n$$

Vectorised computations - use Jacobian matrices

$$\text{for } p \in g(x) \qquad f(g) = f(p) + J_f(p)(g(x) - p) + \cdots$$

$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial g_1} & \cdots & \frac{\partial f_1}{\partial g_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial g_1} & \cdots & \frac{\partial f_k}{\partial g_m} \end{bmatrix}$$

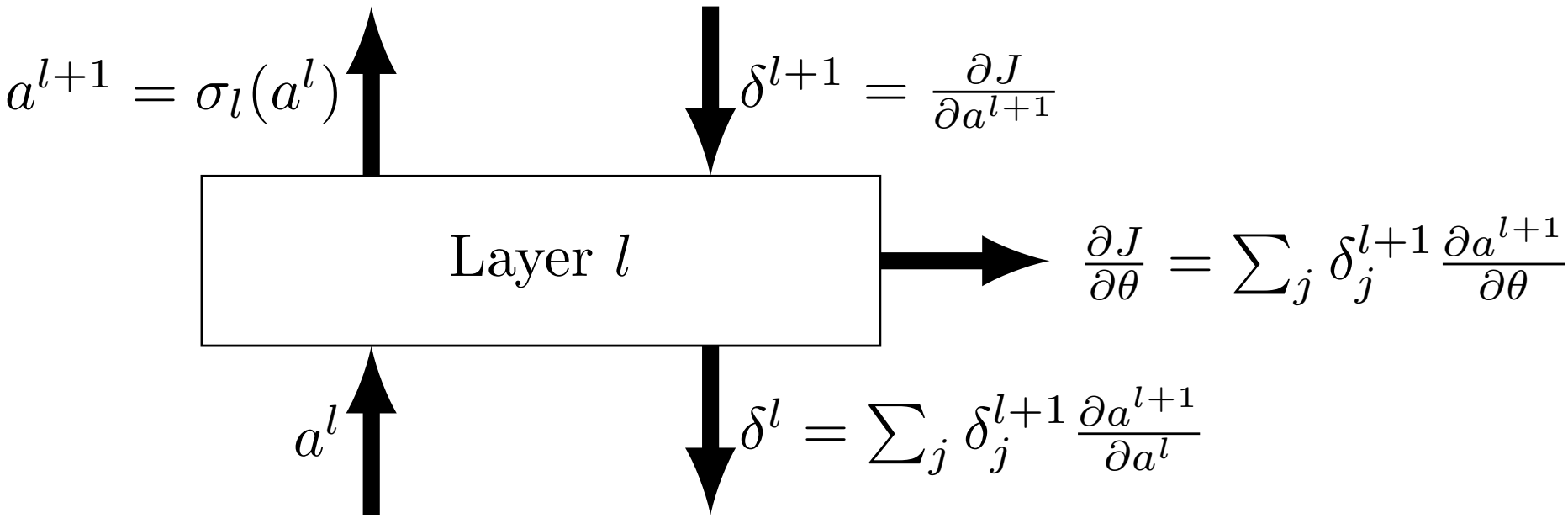
Composition of Jacobians:

$$J_{f \circ g}(x) = J_f(g(a))J_g(x) = J_f(g(x))J_g(x)$$

Artificial Neural Networks - backpropagation

Backpropagation - use the chain rule to calculate SGD derivatves.

- Requires: ANN functions are differentiable
- Layered deep neural network structure has a convenient form for backprop



Artificial Neural Networks - backpropagation

Let $a^1 = x$ be the input vector.
Consider the layerwise activations, a^l, and activation functions, $\sigma_l(\circ)$:

$$a^l = \sigma_{l-1}(a^{l-1}) = \sigma_{l-1}\left(\sum_{i=0}^{m} W_{ij} a_i\right)$$

$$h(x;\theta) = a^{M-1}$$

Treat the loss at final layer as an activation:

$$a^M = J(\theta)[a^{M-1}]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta L\left[h_\theta(x_i), y_i\right]$$

Need to pass these errors backwards through the network!

Artificial Neural Networks - backpropagation

Let δ_i^l be the error derivative at node i in layer l:

$$\delta_i^l = \frac{\partial J}{\partial a_i^l}$$

$$\delta_i^l = \sum_{j=0}^{m} \frac{\partial J}{\partial a_j^{l+1}} \frac{\partial a_j^{l+1}}{\partial a_i^l}$$

$$\delta_i^l = \sum_{j=0}^{m} \delta_j^{l+1} \frac{\partial a_j^{l+1}}{\partial a_i^l}$$

At the final layer (reverse mode AD...):

$$\delta_i^M = \frac{\partial J}{\partial a_i^M} = \frac{\partial a_i^M}{\partial a_i^M} = 1$$

Artificial Neural Networks - backpropagation

Use backwards errors, δ, to find gradient of weights, θ, with respect to the loss function:

$$a_j^{l+1} = \sigma_l \left(\sum_{i=0}^{m} \theta_{ij}^l a_i^l \right)$$

So that at layer l, the gradient of the loss with respect to the weights is:

$$\nabla_{\theta^l} J = \frac{\partial J}{\partial \theta^l} \left[a^{l+1}(\theta^l) \right]$$

$$\nabla_{\theta^l} J = \sum_{j=0}^{m} \frac{\partial J}{\partial a_j^{l+1}} \frac{\partial a_j^{l+1}}{\partial \theta^l}$$

$$\nabla_{\theta^l} J = \sum_{j=0}^{m} \delta_j^{l+1} \frac{\partial a_j^{l+1}}{\partial \theta^l}$$

Artificial Neural Networks - backpropagation

So at each layer we need:

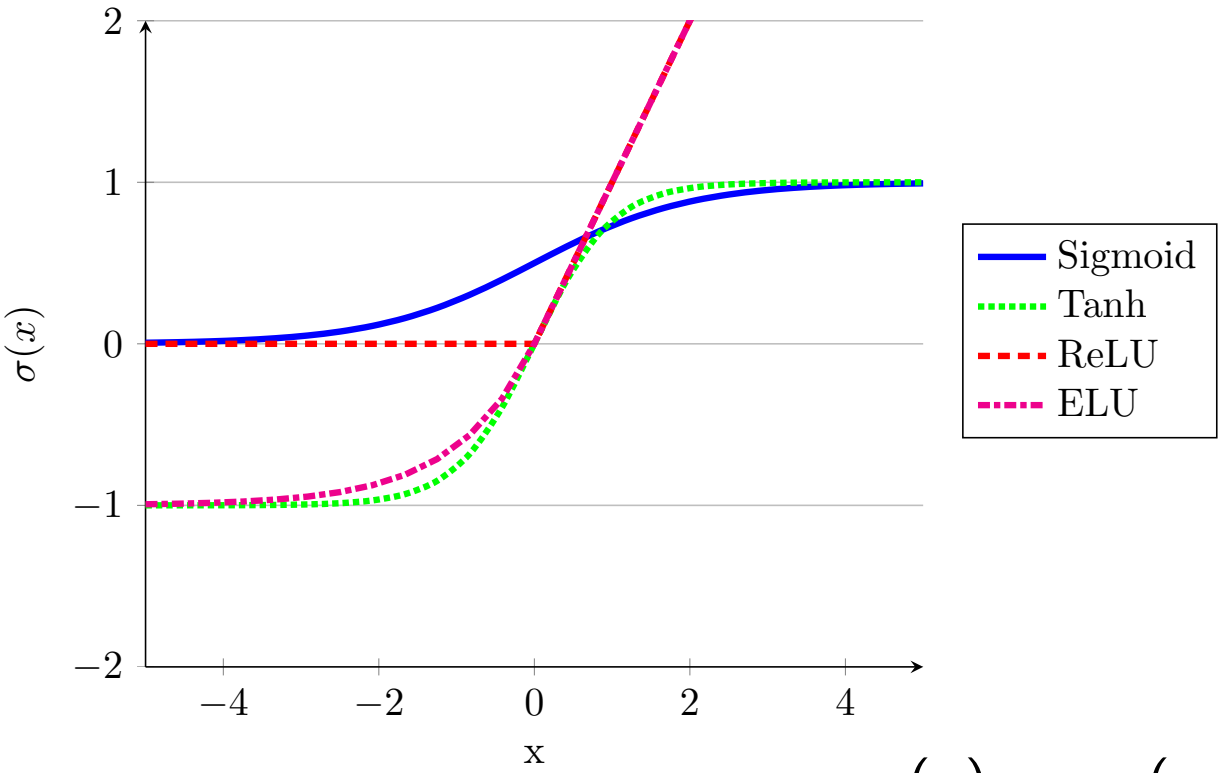
- Weights, θ
- Activation functions σ
- Backwards error gradient (reverse accumulated Jacobians through layer activations):

$$\delta_i^l = \frac{\partial J}{\partial a_i^l} = \sum_{j=0}^{m} \frac{\partial J}{\partial a_j^{l+1}} \frac{\partial a_j^{l+1}}{\partial a_i^l} = \sum_{j=0}^{m} \delta_j^{l+1} \frac{\partial a_j^{l+1}}{\partial a_i^l}$$

More details:

- Bishop, 1994, Neural networks and their applications
- Oxford: Machine Learning 2014-2015 `tinyurl.com/mcwofzr`

Artificial Neural Networks - activation functions



Sigmoid: $\sigma(x) = \dfrac{1}{1+\exp(-x)}$ Tanh: $\sigma(x) = \dfrac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$

ReLU: $\sigma(x) = \max(0, x)$ ELU: $\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ a\left(\exp(x)-1\right) & \text{otherwise} \end{cases}$

Artificial Neural Networks - vanishing gradients

Problems:

- Gradients accumulate multiplicatively
- Tiny error gradients become increasingly small: weights cannot be updated in response to errors
- In recurrent networks: large error gradients: weights become so large that SGD fails (oscillations prevent convergence).

What to do:

- Sigmoid units have ≈zero gradients for high saturation
- Rectified units: linear error gradients
- Alternative approach: centre activations near 0 for a sigmoid - lead to Batch Normalisation.
- ELU's - Improved ReLU units: `arxiv.org/abs/1511.07289`
- Positive activation on an ELU acts like a ReLU
- Negative part on ELU allows information suppression - forces centring without complicated Batch Normalisation

Long short-term memory (LSTM)

LSTMs help to resolve vanishing gradient problem for BPTT.

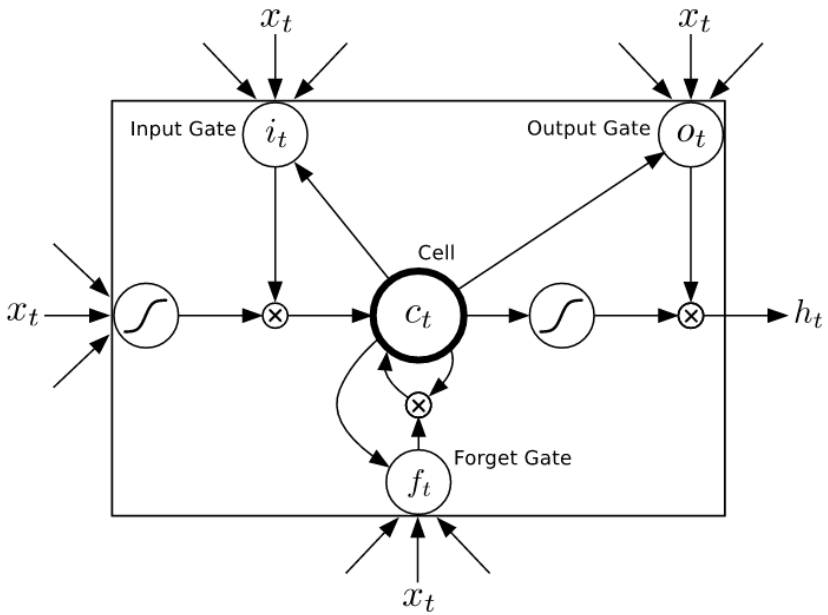


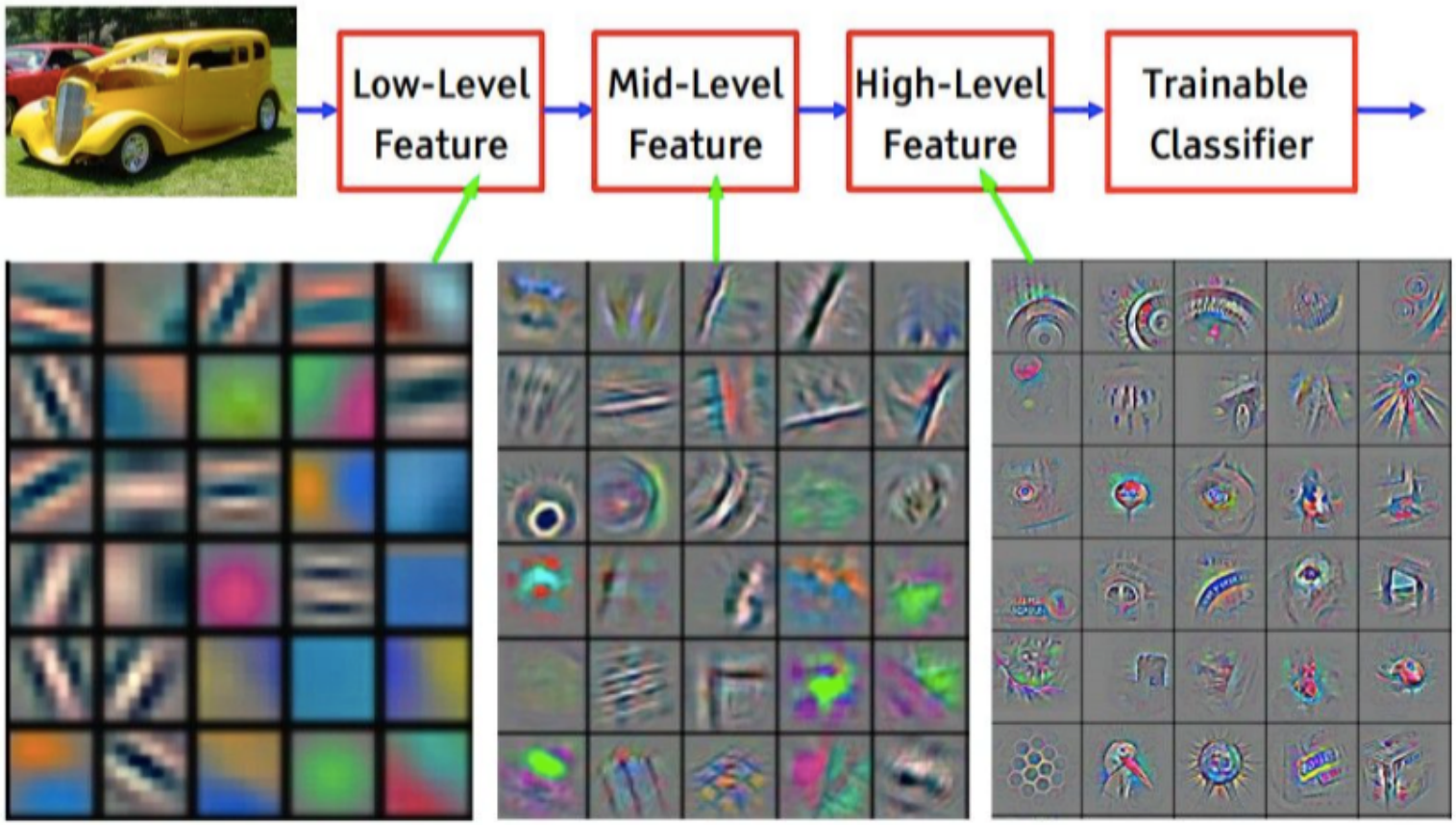
Image: `arxiv.org/abs/1303.5778`

See: `arxiv.org/abs/1503.04069`

- Sigmoid function gates: value 0 to 1
- Cell represents internal memory state
- Input gate: Decides if information is relevant to cell. Decides whether or not to save incoming information.
- Forget gate: Discards cell state if required.
- Output gate: Works as a focus mechanism. Decides if current cell state is useful to outputs. Decides if long-term useful information is/is not useful immediately.

Artificial Neural Networks - Deep Networks

Deep networks learn hierarchical basis functions - efficient encodings



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Image: Yann LeCunn `yann.lecun.com`

Artificial Neural Networks - RNN applications

Image captioning by sequential attention filtering

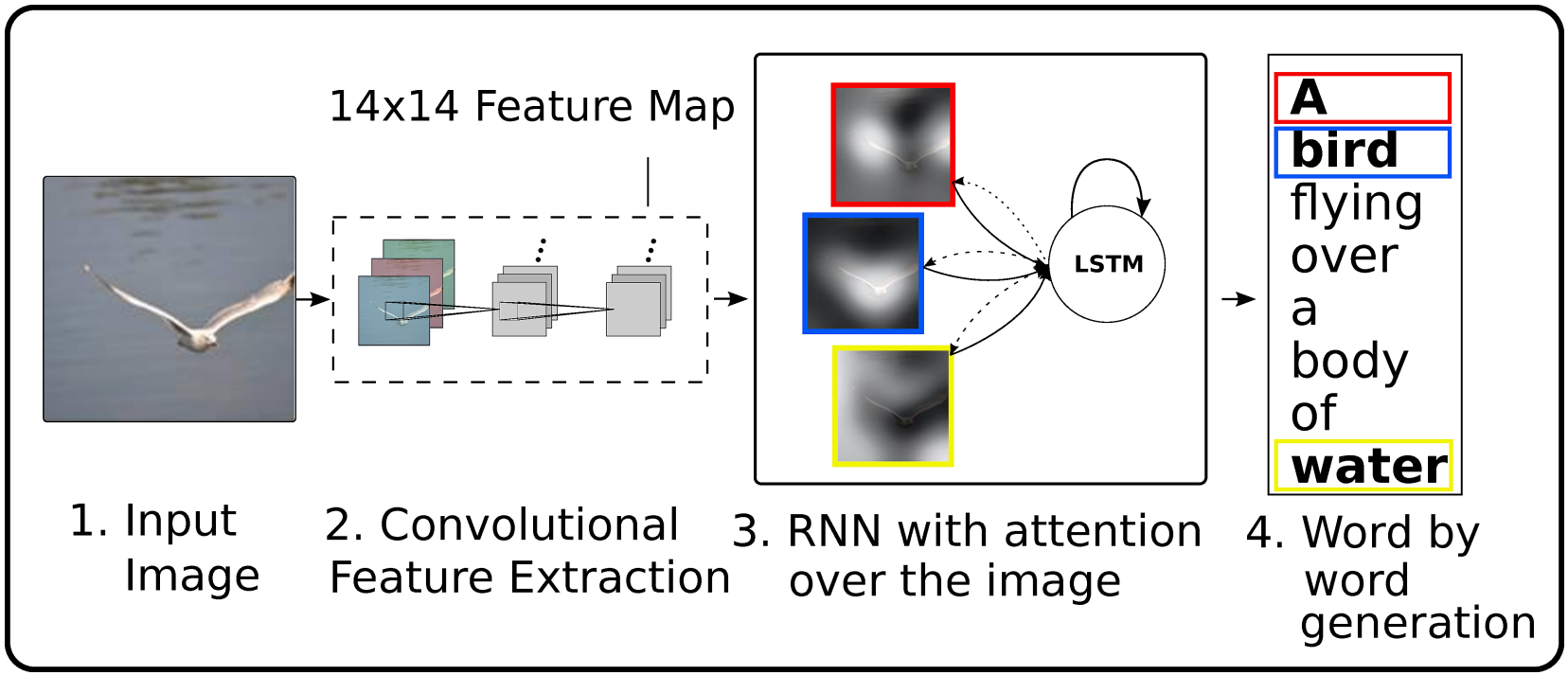


Image: `arxiv.org/abs/1502.03044`

Artificial Neural Networks - software and tools

- Use one of the many very good free software packages!
- Getting started: Keras - `https://keras.io/`
  - Simplified interface to Tensorflow and Theano (automatic differentiation frameworks)
  - Many examples: `https://github.com/fchollet/keras/tree/master/examples`
- More advanced: Tensorflow - `https://www.tensorflow.org/`
  - Use specifies computation graph in terms of tensor operations
- Features:
  - Nice Python interfaces (can be a problem for older Automatic Differentiation frameworks)
  - Sophisticated SGD variants (e.g. ADAM)
  - Activation functions and operations eg. ELU, dropout, BatchNormalisation etc etc.
  - New tricks are developed frequently. Literature moves rapidly. Good packages respond quickly.
  - Convolutional network models (advice, can implement with sparse matrix multiplication and local weight connectivity...)

Case study: Adaptive Basis Element Free Galerkin

- **Original inspiration:** Consider convergence of Series Expansion Stochastic FEM. Convergence rate depends on selected output basis.
- **Question:** How to improve the convergence rate?
- **Advice:** "Choose a better basis"

But how?

- **Idea:** Use Probabilistic Numerical Methods to derive an adaptive basis finite element scheme.
- Represent PDE solution function with an ANN.
- Need to derive a ANN loss function.
- Can derive a basis optimisation objective via information theory.
- Test case: Poisson equation.
- Source: DKE Green, PhD thesis 2017, §7, *Probabilistic analysis for computational mechanics with applications in Civil Engineering*

PDE - energy functionals

Consider PDE with energy functional

$$I(u) = \frac{1}{2}\langle Lu, u\rangle - \langle f, u\rangle$$

Can derive PDE (Euler-Lagrange equations):

$$Lu = f$$

Example for numerical case study, Poisson equation:

$$\nabla^2 u(x) = f(x)$$

Boundary conditions by Lagrange multipliers: Consider energy functional to incorporate constraints $Au = c$:

$$\mathcal{L}(u, \lambda) = I(u) - \langle \lambda, Au(x) - c(x)\rangle$$

Element Free Galerkin

Consider solutions u and v from the Sobolev function space $\mathbb{H}$.
Weak solutions: Consider error $G(u) = Lu - f$. If, for every $v \in \mathbb{H}$, the (Sobolev) inner product error functional satisfies:

$$\langle G(u), v \rangle = 0$$

then $u \in \mathbb{H}$ is said to be a *weak solution*.
Disretisation: Representation of solution function in terms of arbitrary function basis expansion:

$$u(x) = U^i \phi_i(x)$$

Substitute into energy functional, find discrete Euler-Lagrange equations to find discretised weak form solution.
Lots of standard algebra yields bordered Hessian:

$$\begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial u^2} & \frac{\partial^2 \mathcal{L}}{\partial u \partial \lambda} \\ \left(\frac{\partial^2 \mathcal{L}}{\partial u \partial \lambda}\right)^T & \frac{\partial^2 \mathcal{L}}{\partial \lambda^2} \end{bmatrix} \begin{bmatrix} U \\ \Lambda \end{bmatrix} = \begin{bmatrix} K & A^T \\ A & B \end{bmatrix} \begin{bmatrix} U \\ \Lambda \end{bmatrix} = \begin{bmatrix} F \\ C \end{bmatrix}$$

ANN Parameterisation of boundary value PDE solution

Represent solution as weighted sum of basis functions:

$$u_\theta(x) = \sum_i W_i \Phi_{i,\theta}(x)$$

where:

- x - spatial input location
- θ - notational representation of ANN weights
- $u_\theta(x)$ - ANN parameterised solution function
- W_i - basis function coefficient weights
- $\Phi_{i,\theta}(x)$ - ANN parametrised basis functions

End result summary - adaptive solution algorithm

ANN solution and basis representation (discretised weak form)

$$u_\theta(x) = \sum_i W_i \Phi_{i,\theta}(x)$$

- Two phase iterative algorithm:
  - 1: use your favourite optimisation technique to find W_i given $\Phi_{i,\theta}$
  - 2: Train the ANN using the regularised training objective below.

Expectation-Maximisation Neural Network regularised training objective

Using Laplace priors and techniques similar to sparse-coding can derive:

$$J(\theta) = -Q(\theta|\theta^i) \approx \|u_\theta(x) - W^i \Phi_\theta(x)\|_2^2 + \lambda_W \|W^{i+1}\|_2^2 + \lambda_\Phi \|\Phi_\theta(x)\|_1$$

The Expectation-Maximisation algorithm

EM gives (locally optimal) MLE and MAP estimates.

- **Goal:** Find the maximum likelihood parameters for $L(\theta; X) = P(X|\theta)$
- To simplify, introduce latent variable Z such that:

$$P(X|\theta) = \int_{z \in Z} P(X, z|\theta) dz$$

- Two stage optimsation. On iteration i:
  - Expectation step: Find auxiliary function:

$$Q(\theta|\theta^i) = \int_z P(z = Z|X, \theta^i) \log P(X, z = Z, \theta) dz$$

  - Maximisation step: find the values of θ^{i+1} to maximise the auxiliary function:

$$\theta^{i+1} = \underset{\theta}{\operatorname{argmax}} Q(\theta|\theta^i)$$

- See Bishop, 2006, *Pattern Recognition and Machine Learning*.

Sparse coding

Learning over-complete basis functions - improve generalisation performance.

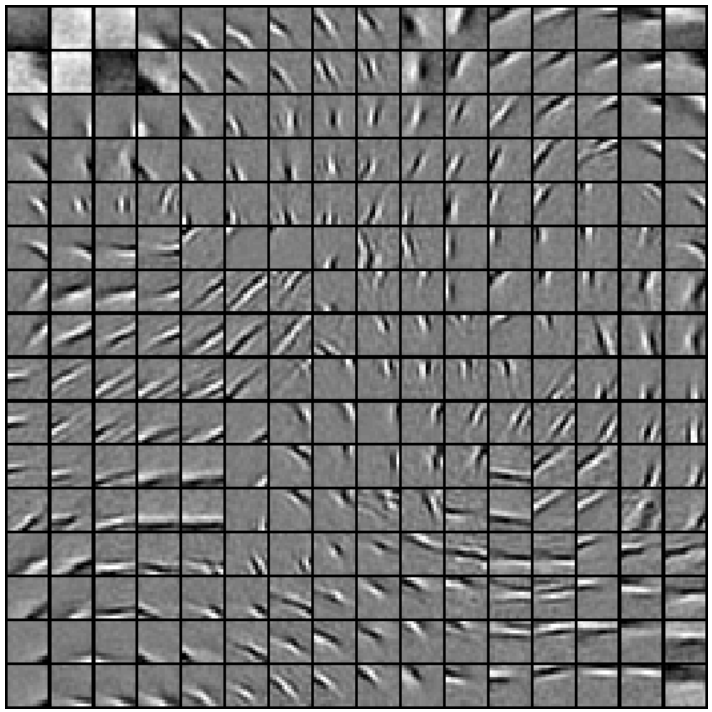


Image: `tinyurl.com/y9wymk3g`

Sparse coding

Technique for learning a 'dictionary' (over-complete basis) representations.
Consider:

$$y(x) = \sum_{i=1}^{N} W_i \phi_i(x)$$

Using KL divergence and Laplace prior can derive:

$$\underset{W,\phi}{\operatorname{argmin}} \left[\|y(x) - W\phi\|_2^2 + \lambda \|W\|_1 \right]$$

such that:

$$\|\phi\|_2^2 \leq C$$

- Two-norm feature constraint prevents failure mode where sparsity penalty, $\lambda\|W\|_1$, disappears by setting W very small and ϕ very large.
- Two step optimsation:
  - Update W given ϕ
  - Update ϕ given W

Probabilistic Numerical Methods

- **Treat numerical methods as inference problems**.
- Even deterministic problems have solutions can be framed as inference problems: the solution is unknown to the subjective observer.
- Overview: Hennig, Osborne, Girolami, 2015, *Probabilistic numerics and uncertainty in computations*, `doi: 10.1098/rspa.2015.0142`.
- Neat example: non-parametric derivation of trapezoidal rule from Bayes rule for Gaussian Processes (update of mean and covariance function).

Probabilistic interpretation of Element Free Galerkin

For linear PDE $Lu = f$ (Dirichlet BC's) consider mean field form of PDE solution:

$$P(u_\theta) = \frac{1}{Z} e^{-\beta\left[\int_\Omega [Lu_\theta(x) - f(x)]^2 dx + \int_{\partial\Omega} [u_\theta(x) - c(x)]^2 dx\right]}$$

$$P(u_\theta) = \frac{1}{Z} \underbrace{\prod_{i=1}^{\infty} e^{-\beta[Lu_\theta(x_i) - f(x_i)]^2}}_{\text{Main spatial term}} \underbrace{\prod_{j=1}^{\infty} e^{-\beta\left[u_\theta(x_j) - c(x_j)\right]^2}}_{\text{Boundary term}}$$

Introduce Galerkin method by considering:

$$P(u_\theta) = \int_W \int_\Phi P(u|W, \Phi) P(W|\Phi) P(\Phi)$$

Substitute $P(u|W, \Phi)$ for a Gaussian Process and $P(W|\Phi)$ for the discretised Galerkin error Gibbs measure.

Probabilistic interpretation of Element Free Galerkin

Need terms for:

$$P(u_\theta) = \int_W \int_\Phi P(u|W,\Phi)P(W|\Phi)P(\Phi)$$

Consider representation of solution function in terms of a Bayesian Linear Regression/Gaussian Process (weight space representation) of PDE solution:

$$P(u_\theta|W,\Phi_\theta,X) \sim \prod_i \mathcal{N}\left(W\Phi_\theta(x_i),\sigma^2\right)$$

Solution of energy functional $\mathcal{L}(u,\lambda) = I(u) - \langle\lambda, u-c\rangle$ yields equations with solutions $U = (W,\lambda)$ so $KU = F$. Represent solution in terms of Gibbs measure:

$$P(W|\Phi_\theta) = \frac{1}{Z}\exp(-\|KU-F\|_2^2)$$

Will be able to treat $P(\Phi)$ as a Laplace prior to simplify marginalisation integrals by introducing sparsity over basis function space.

EM for Element Free Galerkin

Treating solutions as probability measures, error in ANN encoding of solution is minimised when $D_{KL}(P(u)\|P(u|\theta)) = 0$ so:

$$D_{KL}(P(u)\|P(u|\theta)) = \int P(u) \log\left(\frac{P(u)}{P(u|\theta)}\right) du$$

$$D_{KL}(P(u)\|P(u|\theta)) = \int P(u) \log\left(P(u)\right) du - \int P(u) \log\left(P(u|\theta)\right) du$$

So then:

$$-\int P(u) \log\left(P(u|\theta)\right) \geq 0$$

So minimum $D_{KL}(P(u)\|P(u|\theta))$ when $\log P(u|\theta)$ is a maximum. This suggests EM algorithm.

EM for Element Free Galerkin

Introduce latent variable model:

$$P(u_\theta) = \int_W \int_\Phi P(u_\theta, w, \phi) dw d\phi$$

Let:

$$P(W, \Phi_\theta) = P(W, \Phi|\theta)$$

To simplify (cheat) - assume independence:

$$P(W, \Phi_\theta|u) = P(W, \Phi_\theta)$$

This leads to the auxiliary function:

$$Q(\theta|\theta^i) = \int_W \int_\Phi P(w, \phi|\theta^{i+1}) \log P(u_\theta, W, \Phi_\theta) d\phi dw$$

$$Q(\theta|\theta^i) = \int_W \int_\Phi P(w, \phi|\theta^{i+1}) \log P(u_\theta|W, \Phi_\theta) P(W|\Phi_\theta) P(\Phi_\theta) d\phi dw$$

Sparse coding simplification

Want to recover training objective for Maximisation step that is similar to sparse coding mixed $\mathcal{L}^2$ and $\mathcal{L}^1$ regularisation terms.
Goal is to find ANN minimisation objective $J(\theta)$:

$$J(\theta) = -Q(\theta|\theta^i) = \int_W \int_\Phi P(w, \phi|\theta^{i+1}) \log P(u_\theta|W, \Phi_\theta) P(W|\Phi_\theta) P(\Phi_\theta) d\phi dw$$

First, set basis functions to $\mathcal{L}^1$ regularisation by Laplace prior:

$$J(\theta) = -\int_W P(w, \hat{\phi}|\theta^{i+1}) \log P(u_\theta|W, \hat{\phi}) P(W|\hat{\phi}) P(\hat{\phi}) dw$$

Introduce the Galerkin Gibbs measure term

To further simplify the training objective, introduce the probabilistic interpretation of the Galerkin method:

$$P(W|\Phi,\theta^i) = \frac{1}{Z}\exp(-\beta\|KU-F\|_2^2)$$

By Bayesian Linear Regression, the Ordinary Least Squares solution for $KW = F$ corresponds to the Maximum Likelihood Estimate for $P(W|\Phi,\theta)$, denote by $\hat{W}$.

Assume a peaked Gaussian prior on W centred at $\hat{W}$ to induce an $\mathcal{L}^2$ regularisation term for $J(\theta)$:

$$J(\theta) = -P(\hat{w},\hat{\phi}|\theta^{i+1})\log P(u_\theta|\hat{w},\hat{\phi})P(\hat{w}|\hat{\phi})P(\hat{\phi})$$
$$J(\theta) = -\log P(u_\theta|\hat{w},\hat{\phi})P(\hat{w}|\hat{\phi})P(\hat{\phi})$$

Introduce the Galerkin Gibbs measure term

Expanding to recover training objective:

$$\log P(u_\theta|\hat{w},\hat{\phi})P(\hat{w}|\hat{\phi})P(\hat{\phi})$$

Expectation-Maximisation Neural Network regularised training objective

Using Laplace priors and techniques similar to sparse-coding can derive:

$$J(\theta) = -Q(\theta|\theta^i) \approx \|u_\theta(x) - W^i\Phi_\theta(x)\|_2^2 + \lambda_W\|W^{i+1}\|_2^2 + \lambda_\Phi\|\Phi_\theta(x)\|_1$$

where:

- λ_W: Weight regularisation hyperparameter
- λ_Φ: Basis sparsity regularisation hyperparameter

- Galerkin method follows from expression of PDE solution as weight-space Gaussian process (Bayesian linear regression).
- Galerkin solution completes latent variable model with $P(W|\phi)$.

Numerical test case: 1D Poisson equation

Deterministic one-dimensional Poisson equation with fixed Dirichlet boundary conditions:

$$\begin{aligned} \nabla^2 u(x) &= f(x) && x \in \Omega \\ u(x) &= c(x) && x \in \partial\Omega \\ f(x) &= -1 \\ c(x) &= 0 \\ \Omega &= [0,1] \end{aligned}$$

Simple solution:

$$u(x) = -\frac{1}{2}x^2 + \frac{1}{2}x$$

ANN Design

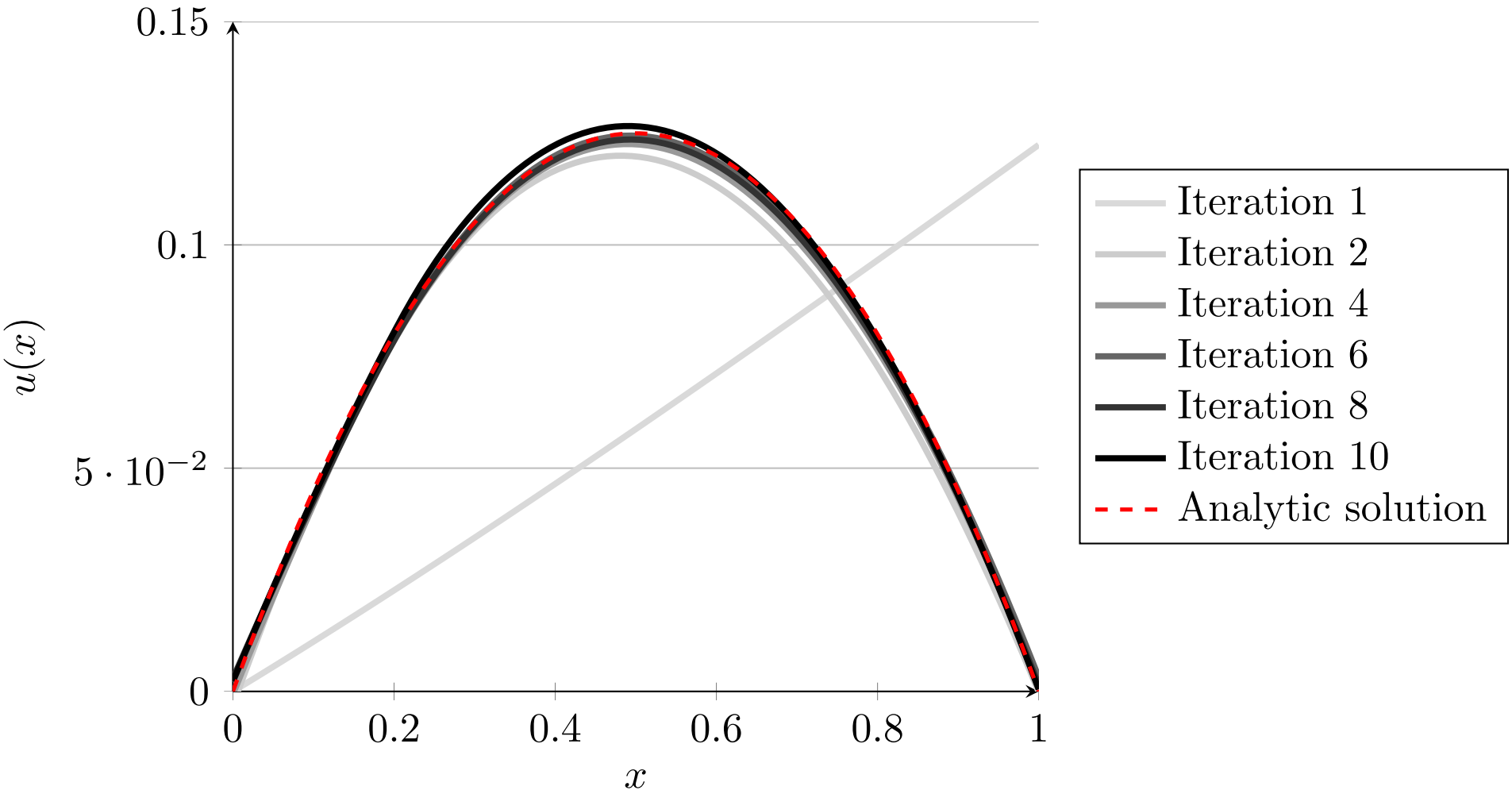
Architecture:

- Layer 0: 1 dimensional input position : $x \in \Omega$
- Layer 1: 50 ELU units
- Layer 2: 50 ELU units
- Layer 3: 50 Tanh units : $\Phi_\theta(x)$
- Layer 4: 10 Linear units : $W\Phi_\theta(x)$
- Layer 5: 1 dimensional output : $u(x) = W\Phi_\theta(x)$

Design considerations:

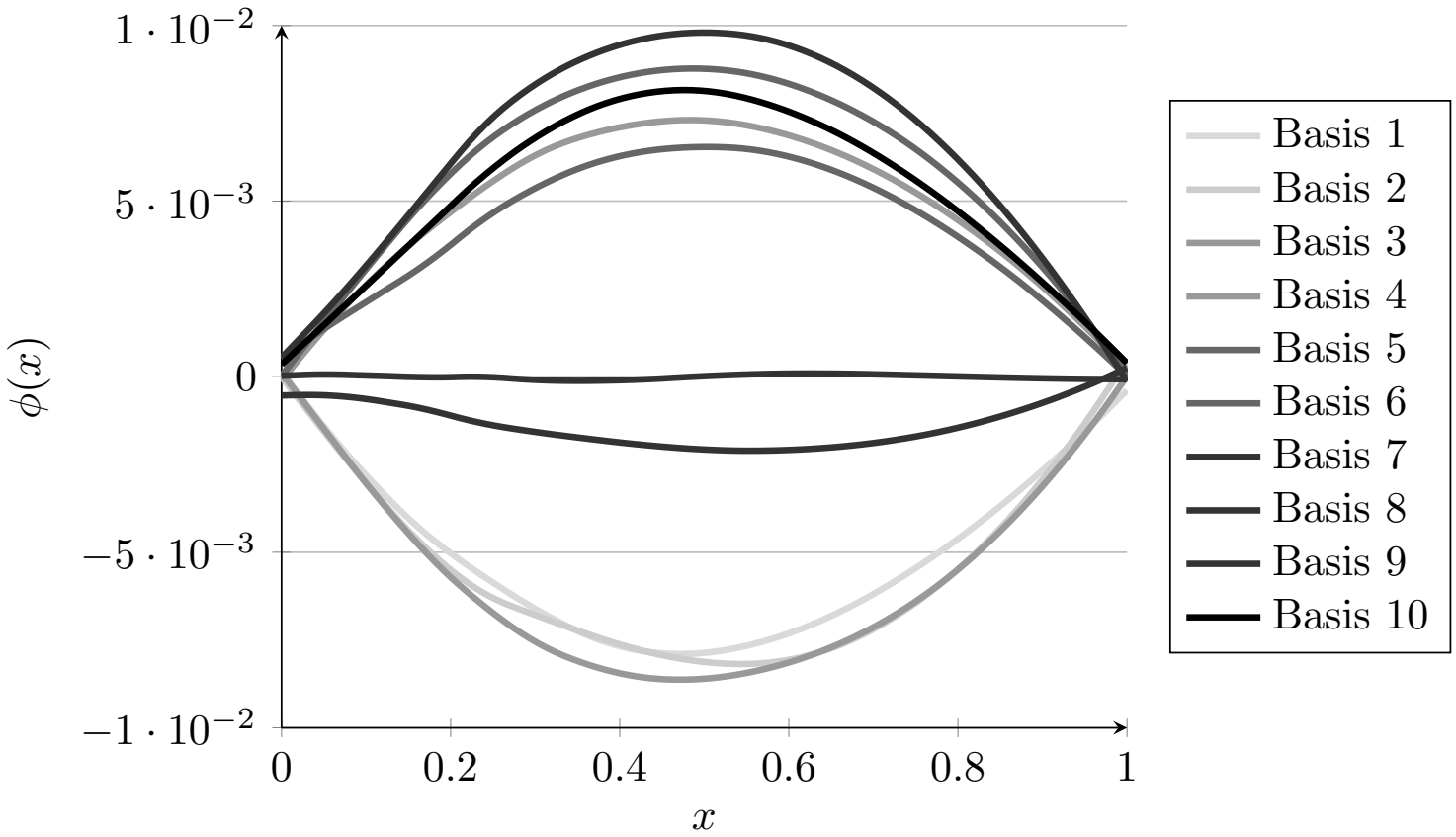
- ELU layers for good performance with depth
- Tanh Layer to squash basis functions to range $[0, 1]$
- Linear weights (no bias) to represent $KW = F$

Trained solution



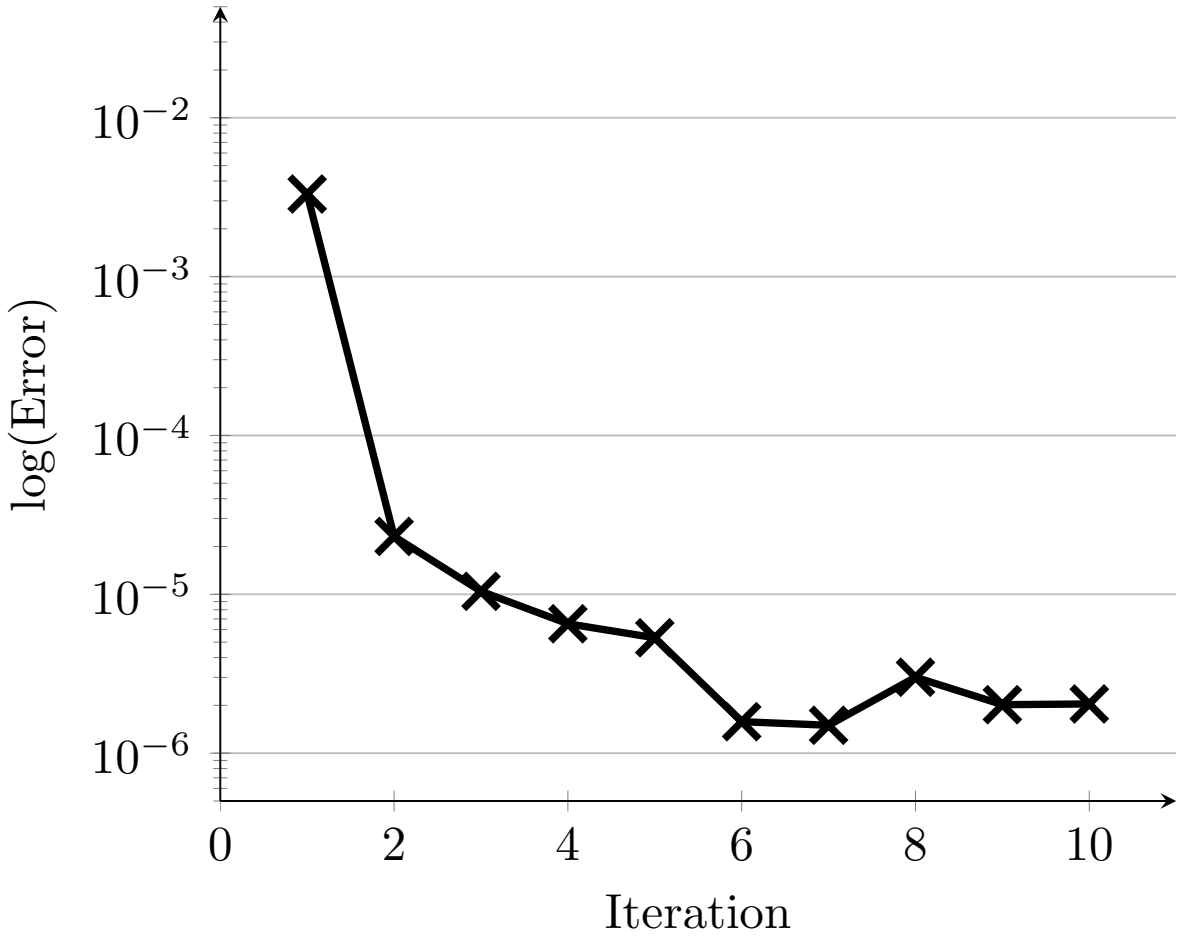
Solution converges rapidly.

Learnt basis functions



Basis function irregularities due to Monte Carlo integration used to evaluate functional inner products for weak-form projections e.g. $K_{ij} = \langle \nabla \phi_i, \nabla \phi_j \rangle$ etc. etc.

Error reduction



Rapid convergence with iteration. EFG inner products evaluated by Monte Carlo - source of error oscillation about MCS accuracy.

Conclusions

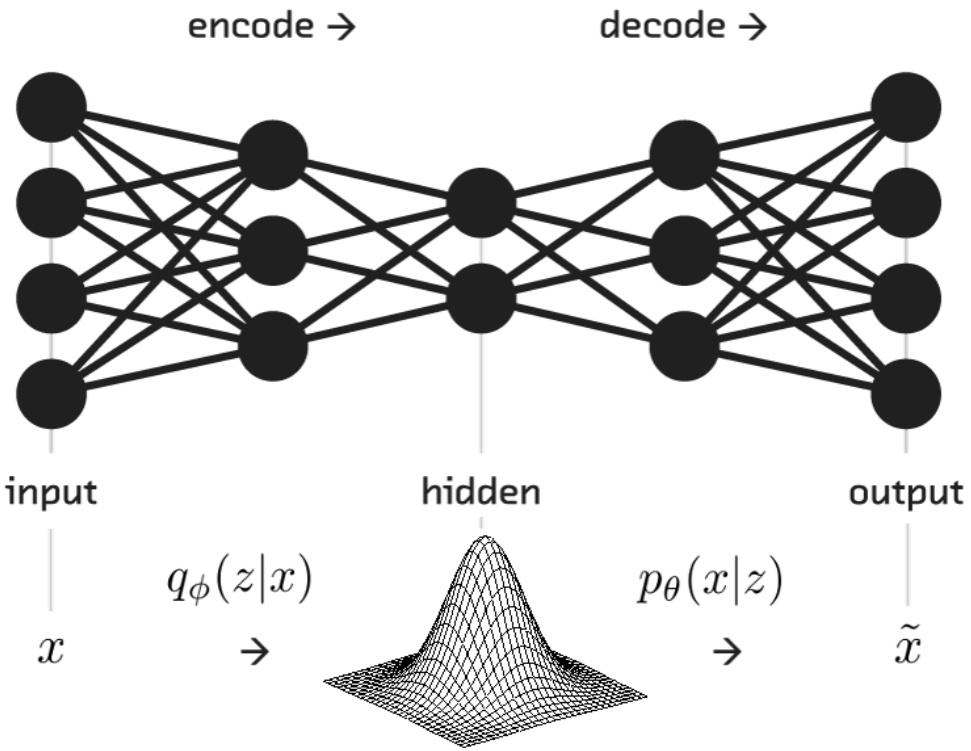
- Parametric approach combining probabilistic numerics and Galerkin methods yields Gradient Descent form of basis function adaption.
- Probabilistic interpretation - Galerkin method provides $P(W|\Phi)$ term for Gaussian process representation of $P(u)$.
- Deterministic inputs converge to single basis equal to PDE solution plus noise terms.
- More interesting basis functions would be found for probabilistic inputs. For inputs $x \in X$ and outputs $y \in Y$:

$$P(y) = \int_X P(y|x)P(x)dx$$

- Related to Probabilistic Principle Components Analysis (PPCA).
- Long term: automating numerical methods.

Future directions - Generative models - VAE

- Autoencoding Variational Bayes - `arxiv.org/abs/1312.6114`



ANN density estimation and simulation technique. Samples from latent space:

$$P(X) = \int_Z P(X|z,\theta)P(z)dz$$

Want to sample from latent space such that generated outputs match input distribution.

Image: `https://tinyurl.com/yax9f64r`

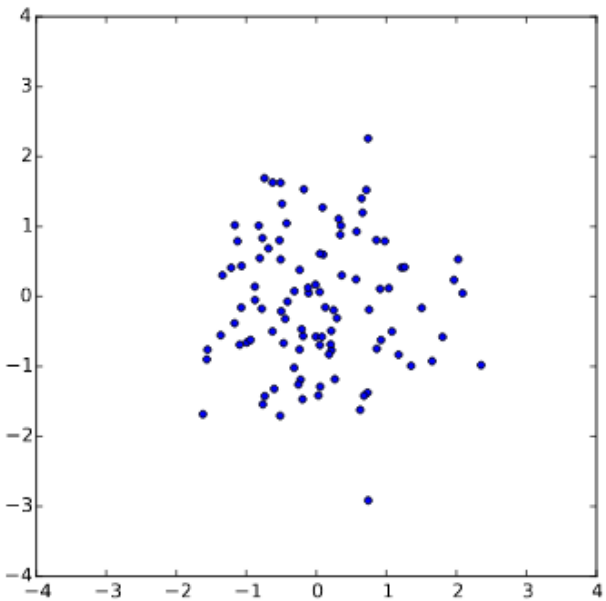
VAE - Decoder idea

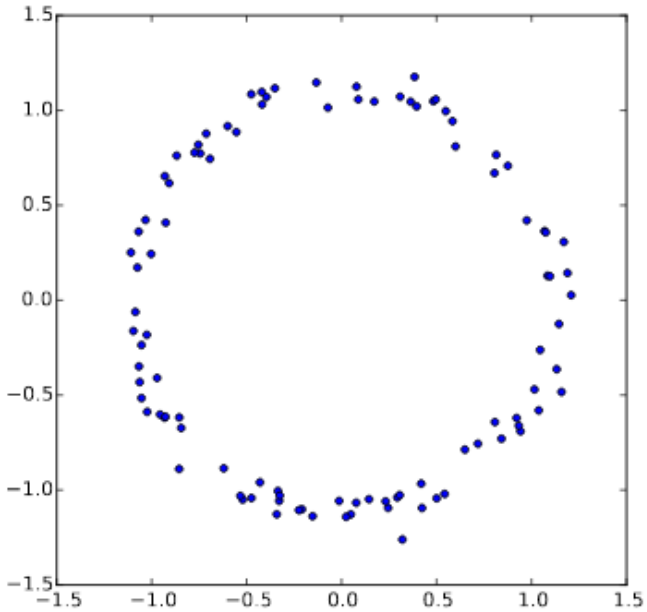
Latent space representation: Gaussian with specified mean and standard deviation.

$$z \sim \mathcal{N}(\mu, \sigma^2)$$

Decoder: Arbitrary map to new random variable

$$X = g(z) = \frac{z}{10} + \frac{z}{\|z\|}$$





Images: `arxiv.org/abs/1606.05908`

VAE - Training objective

Assume latent representation:

$$P(z) = \mathcal{N}(0, \mathbb{I})$$

Then decoder network is:

$$P(X) = \int_z P(X|z,\theta)P(z)dz$$
$$P(X|z,\theta) = \mathcal{N}(X|f(z,\theta), \sigma^2\mathbb{I})$$

Improve performance by learning encoder network:

$$P(z|X,\theta) = \mathcal{N}\left(\mu(X), \sigma(X)\right)$$

Regularisation: Force encoder network to match $P(z) = \mathcal{N}(0, \mathbb{I})$
Training objective:

$$J(\theta) = \underbrace{\mathbb{E}_{z\sim Q}\left[\log P(X|z)\right]}_{\text{reconstruction error}} - \underbrace{\mathcal{D}\left[Q(z|X)\|P(z)\right]}_{\text{regulariser}}$$

Future directions - Generative models

- Key paper: `arxiv.org/abs/1406.2661`
- Generator and discriminator networks are in a zero-sum game, solution converges to a Nash equilibrium.

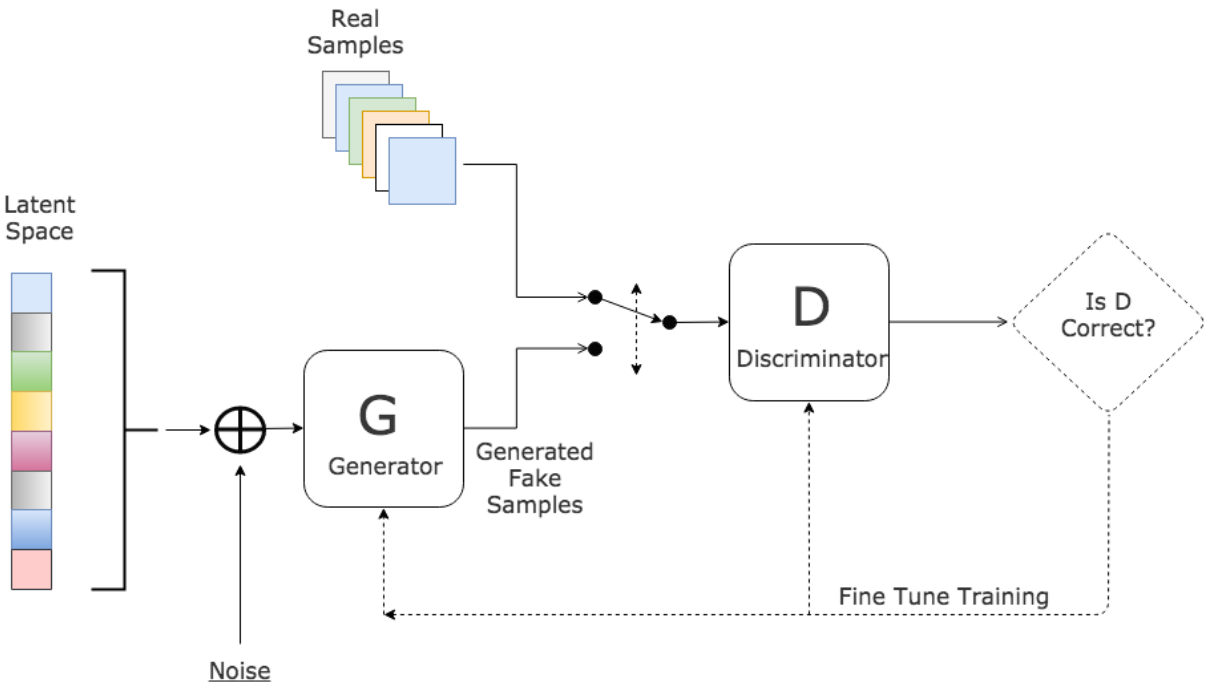


Image: `https://tinyurl.com/ycf3rthw`

Future directions - Generative models

- All sorts of cool applications
- Nice example - image synthesis from text (image source): `arxiv.org/abs/1612.03242v1`

Thank you!

Sparse coding - probabilistic interpretation

- See: Olshausen, Field, 1996, *Natural image statistics and efficient coding*, doi: 10.10188/0954-898x/7/2/014
- See: Stanford UFLDL - tinyurl.com/y7n6alv5

Technique for learning 'dictionary' (over-complete basis) representations:

$$y(x) = \sum_{i=1}^{N} W_i \phi_i(x) + \text{Gaussian noise}$$

$$P(y(x)|W,\theta) = \frac{1}{Z} \exp\left(- \frac{(y(x) - \sum_{i=1}^{N} W_i \phi_i(x))^2}{2\sigma^2} \right)$$

Find basis such that model $P(y|\theta)$ is as close as possible to input data distribution, $P^*(y)$:

$$D_{KL}(P^*(y) \| P(y|\theta)) = \int P^*(y) \log \left(\frac{P^*(y)}{P(y|\theta)} \right)$$

Minimising $D_{KL}(P^*(y) \| P(y|\theta))$ equivalent to maximising $\log P(y|\theta)$.

Sparse coding - probabilistic interpretation

Assume independence prior on W so $P(W) = \prod_i P(W_i)$
Want:

$$P(y(x)|\phi) = \int P(y(x)|W, \phi)P(W)dw$$

Consider Laplace priors to force peaked distribution on W:

$$P(w_i) \propto \exp(-\beta|w_i|) \text{ for large } \beta$$

then, let max $W = W^*$:

$$P(y(x)|\phi) = \int P(y(x)|W, \phi)P(W)dw = P(y(x)|\phi, W^*)$$

Then, our goal is to find:

$$\phi^* = \underset{\phi}{\text{argmax}} \left(P(y(x)|\phi, W^*)\right)$$

Sparse coding - probabilistic interpretation

Define $P(y(x), W|\phi)$ in terms of (Gaussian process) energy function:

$$H(y(x), W|\phi) = -\log P(y(x)|\phi, W)P(W)$$

$$H(y(x), W|\phi) = \sum_{j=1}^{m} \|y(x^j) - \sum_{i=1}^{k} w_i^j \phi_i\|_2^2 + \lambda \|W\|_1$$

Two-norm constraint prevents optimisation failure mode where weights scale down and basis features scale up.
Minimum energy maximises likelihood:

$$J(\theta) = H(y(x), W|\phi) = \sum_{j=1}^{m} \|y(x^j) - \sum_{i=1}^{k} w_i^j \phi_i\|_2^2 + \lambda \|W\|_1$$