Deep Kernel Learning (AISTATS'16)

2016. 08. 09.

Saehoon Kim (Ph. D. candidate)

Machine Learning Group

Outline

- **Deep Kernel Learning (AISTATS'16)**
  - A. G. Wilson (CMU), Z. Hu (CMU), R. Salakhutdinov (Univ. of Toronto) and E. P. Xing (CMU)

- Gaussian Process Kernels for Pattern Discovery and Extrapolation (ICML'13)
  - A. G. Wilson (Cambridge) and R. P. Adams (Havard)

- Kernel Interpolation for Scalable Structured Gaussian Process (KISS-GP) (ICML'15)
  - A. G. Wilson (CMU) and H. Nickisch (Phillips Research)

Deep Kernel Learning

- They transform **the inputs of a spectral mixture base kernel with a deep neural network**

- For a scalable kernel representation, they use *local kernel interpolation, inducing points, and structure exploiting* (Kronecker and Toeplitz) algebra

- On diverse applications (including a 2 million dataset), they showed the improved performance

Gaussian Process (GP)

- Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution
  - It is completely specified by its mean and covariance functions

$$
\begin{aligned}
m(\boldsymbol{x}) &= \mathbb{E}\left[f(\boldsymbol{x})\right] \\
k(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}\left[\left(f(\boldsymbol{x}) - m(\boldsymbol{x})\right)\left(f(\boldsymbol{x}) - m(\boldsymbol{x})\right)\right]
\end{aligned}
$$

  - It can be considered as **a distribution over functions**

$$
\begin{aligned}
f(\boldsymbol{x}) &\sim GP\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\right) \\
\mathbf{f} &\sim \mathcal{N}\left(\boldsymbol{m}, \boldsymbol{K}\right)
\end{aligned}
$$

$$
\mathbf{f} \triangleq \left[f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)\right]^\top \qquad \boldsymbol{m} \triangleq \left[m(\boldsymbol{x}_1), \cdots, m(\boldsymbol{x}_n)\right]^\top
$$

$$
[\boldsymbol{K}]_{ij} \triangleq k(\boldsymbol{x}_i, \boldsymbol{x}_j)
$$

C. E. Rasmussen and C. K. I. Williams. Gaussian Process for Machine Learning . Springer Verlag (2005).

Gaussian Process (GP)

- Assuming a Gaussian noise, $y(\boldsymbol{x})|f(\boldsymbol{x}) \sim \mathcal{N}\left(y(\boldsymbol{x}); f(\boldsymbol{x}), \sigma^2\right)$, the predictive distribution is given by

$$\begin{aligned} \mathbf{f}_*|\boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\theta} &\sim \mathcal{N}\left(\mathbb{E}[\mathbf{f}_*], \mathrm{cov}(\mathbf{f}_*)\right) \\ \mathbb{E}[\mathbf{f}_*] &= \boldsymbol{\mu}_* + \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})\left[\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} \\ \mathrm{cov}[\mathbf{f}_*] &= \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})\left[\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \end{aligned}$$

- The marginal log-likelihood is given by

$$\log p(\boldsymbol{y}|\boldsymbol{X}; \boldsymbol{w}, \boldsymbol{\theta}) \propto -\left[\boldsymbol{y}^\top \left(\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} + \log|\boldsymbol{K} + \sigma^2 \boldsymbol{I}||\right)\right]$$

Gaussian Process vs. Neural Network

- Bayesian NN with infinitely many hidden units converged to GP with a particular kernel (covariance) function

	Gaussian Process	Neural Network
Representation Learning	-	Possible
Uncertainty Measuring	Possible	-
Extrapolation	Possible	-
Time Complexity (Inference)	High (O(N^2))	Low
Space Complexity (Inference)	High (O(N^2))	Low

R. Neal. Bayesian Learning for Neural Networks. Springer Verlag (1996).

Deep Kernel Learning

- They show ***how to construct*** *kernels which encapsulate the expressive power of neural networks*, and **how to learn** *these kernels in a scalable GP framework*

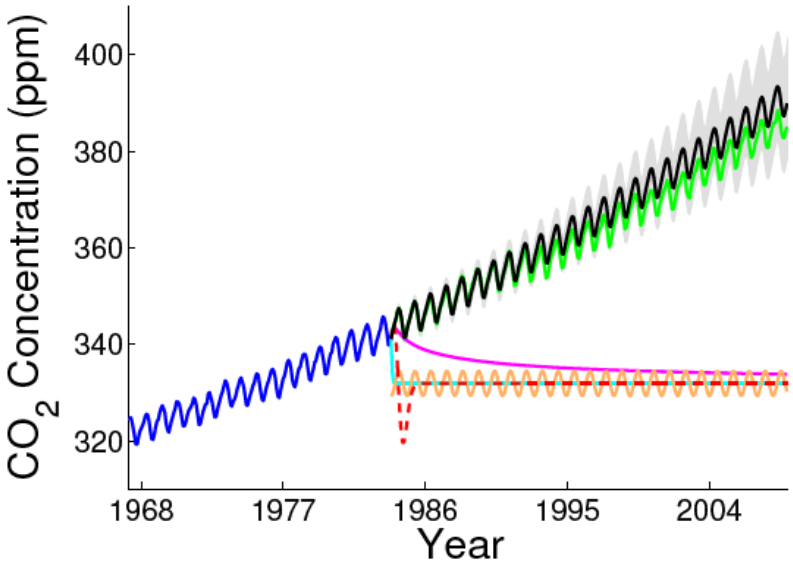
- Starting from a base kernel with hyperparameters θ, the new kernel is derived as

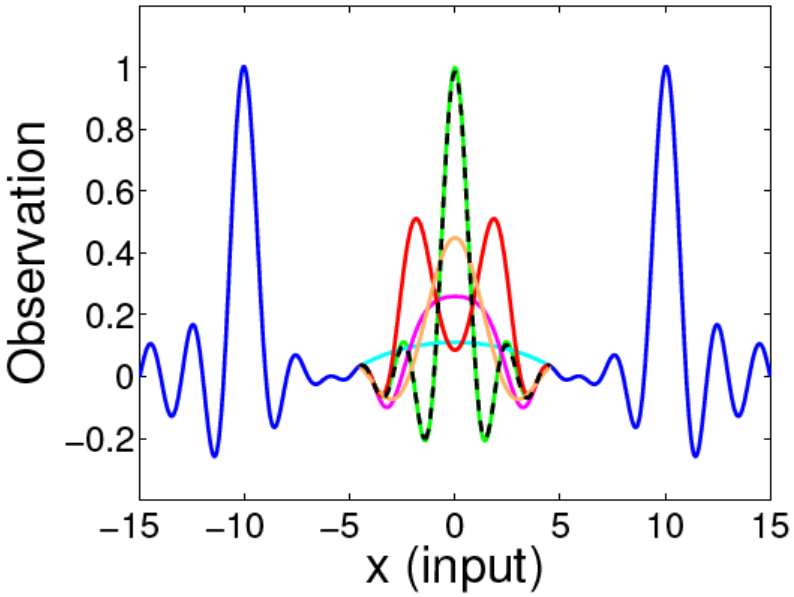
$$k(\boldsymbol{x}_i, \boldsymbol{x}_j|\boldsymbol{\theta}) \rightarrow k\left(g(\boldsymbol{x}_i; \boldsymbol{w}), g(\boldsymbol{x}_j; \boldsymbol{w})|\boldsymbol{\theta}, \boldsymbol{w}\right)$$

  - Which base kernel is used?
  - How to learn the hyperparameters?

Why we need spectral mixture base kernels?

- To discover patterns and enable extrapolation, the authors explore flexible classes of kernels that support a broad class of stationary kernels





A. G. Wilson and R. P. Adams. Gaussian Process Kernels for Pattern Discovery and Extrapolation. ICML'13.

Bochner's theorem

Theorem 1. *(Bochner) A complex-valued function k on $\mathbb{R}^d$ is the covariance function of a weakly stationary mean square continuous complex-valued random process on $\mathbb{R}^d$ if and only if it can be represented as*

$$k(\tau) = \int_{\mathbb{R}^d} \exp(2\pi i s^\top \tau) p(s) ds,$$

where $p(s)$ is a probability density function.

It is often to call $p(s)$ as the *spectral density* or *power spectrum* of k, and k and p are Fourier duals:

$$\begin{aligned} k(\tau) &= \int p(s) \exp(2\pi i s^\top \tau) ds \\ p(s) &= \int k(\tau) \exp(-2\pi i s^\top \tau) d\tau \end{aligned}$$

A. G. Wilson and R. P. Adams. Gaussian Process Kernels for Pattern Discovery and Extrapolation. ICML'13.

Limitation of SE kernel

- The spectral density of squared exponential (SE) kernel is given as

$$k_{SE}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2l^2}||\boldsymbol{x} - \boldsymbol{x}'||_2^2\right)$$

$$S_{SE}(\boldsymbol{x}, \boldsymbol{x}') = \left(2\pi l^2\right)^{d/2} \exp\left(-2\pi^2 l^2 s^2\right)$$

- SE kernels and mixtures of SE kernels **live in a very small region of the set of possible stationary kernels**
  - They correspond only to Gaussian spectral densities centered on the origin!

Spectral Mixture base kernels

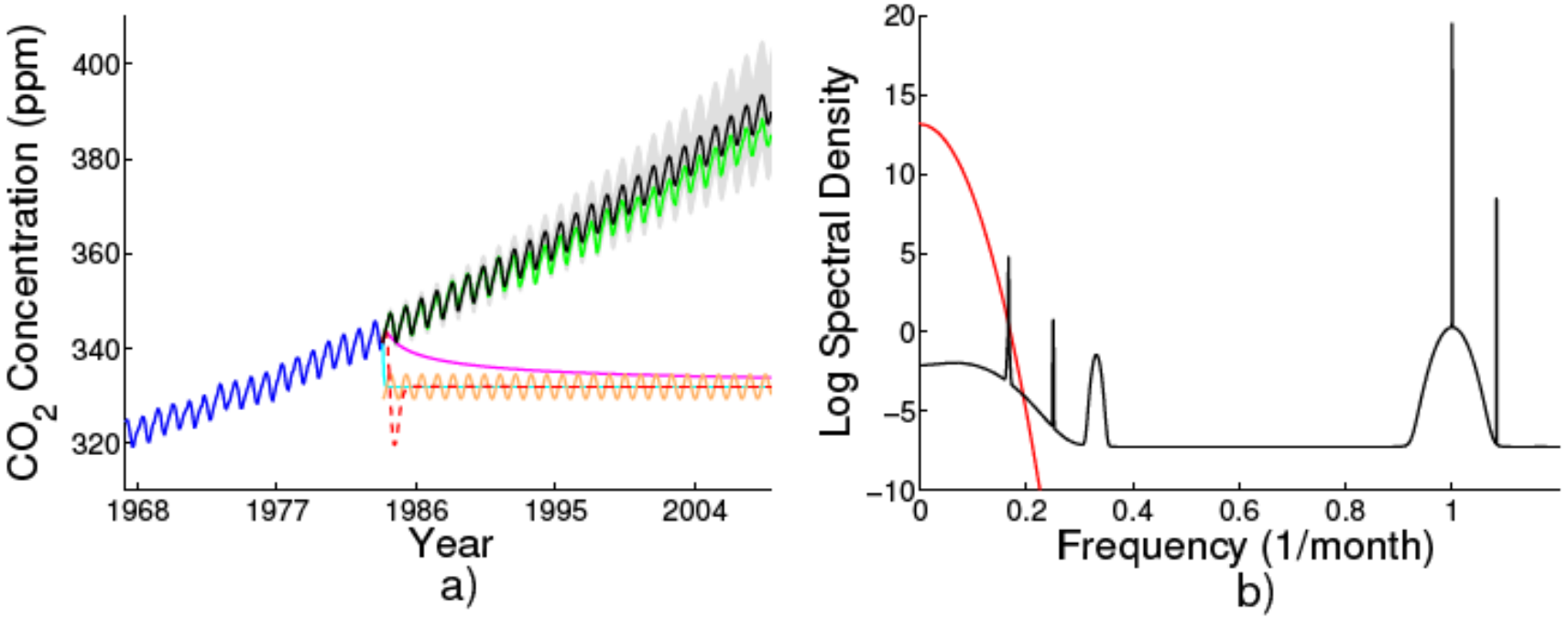
- They employ **a mixture of Gaussian spectral densities that have non-zero means**

- Consider a mixture model:

$$\begin{aligned}
\phi(s;\mu,\sigma^2) &= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}(s-\mu)^2\right) \\
p(s) &= \sum_{i=1}^{Q} w_i \phi\left(s;\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i\right) \\
k(\boldsymbol{\tau}) &= \sum_{i=1}^{Q} w_i \frac{|\boldsymbol{\Sigma}_i|^{1/2}}{(2\pi)^{d/2}}\exp\left(-\frac{1}{2}||\boldsymbol{\Sigma}_i^{1/2}(\boldsymbol{\tau})||_2^2\right)\cos\left(\boldsymbol{\tau}^\top 2\pi\boldsymbol{\mu}_i\right)
\end{aligned}$$

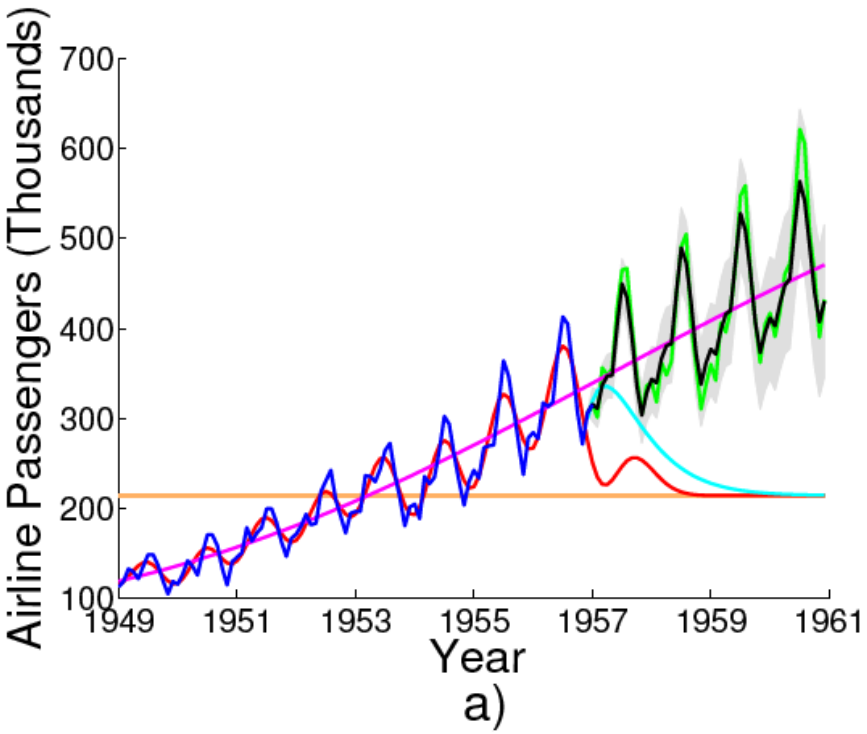
Some experimental results (1/2)

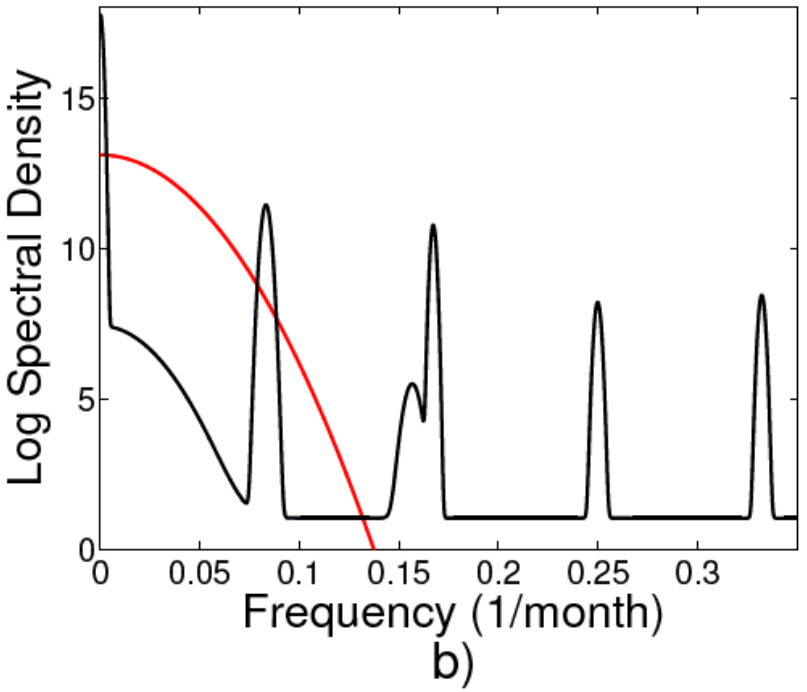
- SM kernels are tractable, allowing us to derive the closed form of marginal likelihood
- Extrapolation Atmospheric CO_2



Some experimental results (2/2)

- Airline Passenger Data

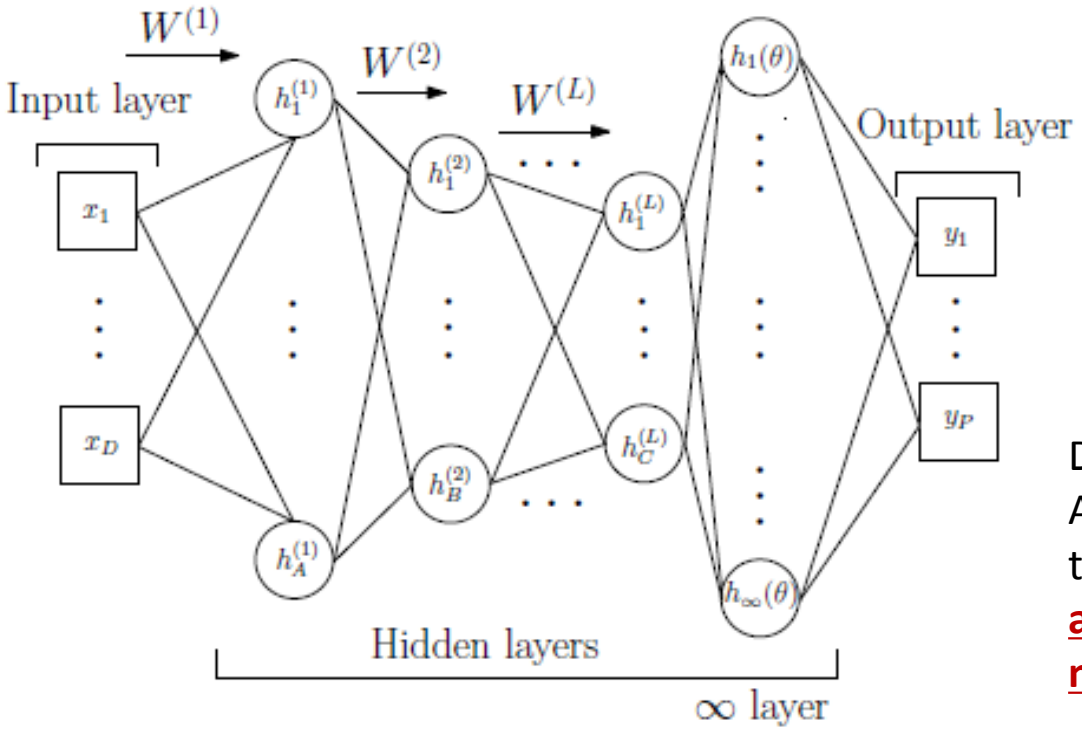




Back to deep kernel learning

- Incorporating a neural network into SM kernels

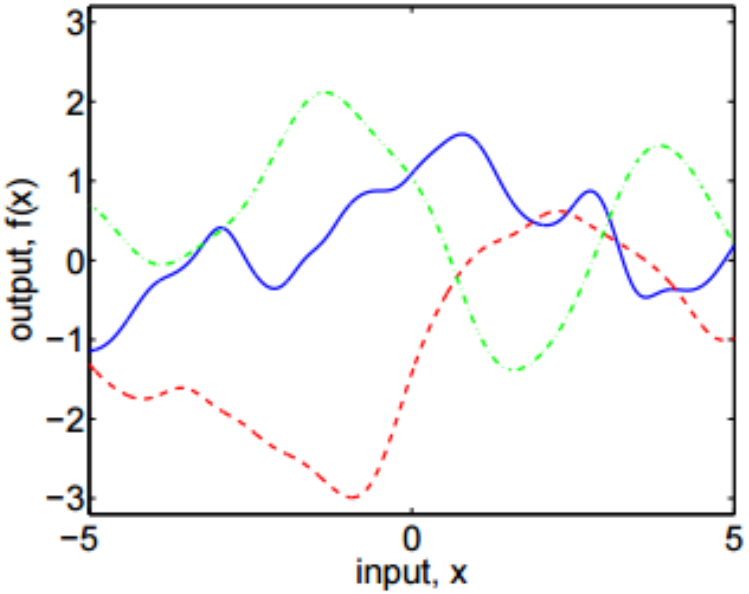
$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{Q} w_i \frac{|\boldsymbol{\Sigma}_i|^{1/2}}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}||\boldsymbol{\Sigma}_i^{1/2}(g(\boldsymbol{x};\boldsymbol{w}) - g(\boldsymbol{x}';\boldsymbol{w}))||_2^2\right) \cos\left((g(\boldsymbol{x};\boldsymbol{w}) - g(\boldsymbol{x}';\boldsymbol{w}))^\top 2\pi\boldsymbol{\mu}_i\right)$$



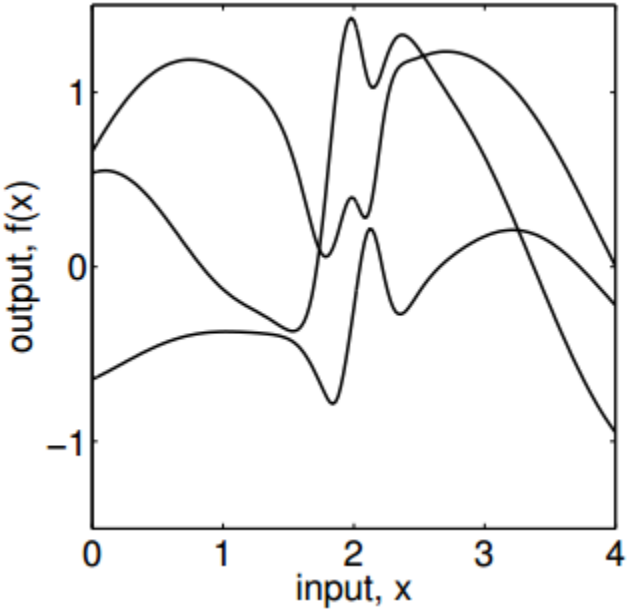
Deep kernel Learning:
A GP with a NN maps the inputs through **L hidden layers** followed by **a hidden layer with an infinite number of basis functions**

Nonstationary kernels

- Now, it is an example of nonstationary kernels..



Random functions drawn from GP with a stationary kernel



Random functions drawn from GP with a non-stationary kernel

How to learn hyperparameters?

- Maximize the marginal log-likelihood w.r.t. the kernel hyperparameter and the weights of the network
  - The weights of the network should be pre-trained!
  - Any type of network can be embedded
- A simple chain rule is used to update the parameters:

$$\log p(\boldsymbol{y}|\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}) \propto -\left[\boldsymbol{y}^\top \left(\boldsymbol{K}+\sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{y} + \log|\boldsymbol{K}+\sigma^2\boldsymbol{I}||\right)\Big]$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{K}}\frac{\partial \boldsymbol{K}}{\partial \boldsymbol{\theta}}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{K}}\frac{\partial \boldsymbol{K}}{\partial g(\boldsymbol{x};\boldsymbol{w})}\frac{\partial g(\boldsymbol{x};\boldsymbol{w})}{\partial \boldsymbol{w}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{K}} = \frac{1}{2}\left(\boldsymbol{K}^{-1}\boldsymbol{y}\boldsymbol{y}^\top\boldsymbol{K}^{-1} - \boldsymbol{K}^{-1}\right)$$

A Scalable GP framework

- It is very *time-consuming to compute the partial derivative* required at every update step:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{K}} = \frac{1}{2}\left(\boldsymbol{K}^{-1}\boldsymbol{y}\boldsymbol{y}^{\top}\boldsymbol{K}^{-1} - \boldsymbol{K}^{-1}\right)$$

- Adopt the scalable GP framework to accelerate the update
  - Exploit the existing structure of the covariance matrix to make the inversion and matrix multiplication fast
  - The approximated covariance matrix should be similar to the original covariance matrix as much as possible

Inducing Point Methods

- A sort of low-rank approximation
- With m landmark points, the kernel is approximated as

$$\boldsymbol{K} \approx \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{L})\boldsymbol{K}(\boldsymbol{L}, \boldsymbol{L})^{-1}\boldsymbol{K}(\boldsymbol{L}, \boldsymbol{X})$$

- It takes $\mathcal{O}\left(m^2 n + m^3\right)$ computations and $\mathcal{O}\left(mn + m^2\right)$ storage for hyperparameter learning
- It takes $\mathcal{O}\left(m\right)$ and $\mathcal{O}\left(m^2\right)$ for predictive mean and variance

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs, NIPS'06.

Fast Structure Exploiting Inference

- If we have inputs on a Cartesian grid, $\boldsymbol{x} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_P$,

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \prod_{p=1}^{P} k(\boldsymbol{x}_i^p, \boldsymbol{x}_j^p)$$

$$\boldsymbol{K} = \boldsymbol{K}_1 \otimes \cdots \otimes \boldsymbol{K}_p$$

- We can efficiently find the SVD of K **by separately computing SVD of each of K_i**.
  - Kernel matrix inversion (including matrix-vector multiplication) now becomes much faster!
  - A Cartesian grid can be approximated by k-means

A. Wilson et al. Fast kernel learning for multi-dimensional pattern extrapolation, NIPS'14.

Experimental Results

- Face orientation extraction
  - Predicting the orientation of a face
  - Constructing datasets of 28x28 face images by randomly rotating (-90 to + 90)

- Neural network architecture
  - 2 convolutional layers
  - 4 fully-connected layers
  - Two-dimensional outputs (that can be interpreted as the orientation of a face)

Experimental Results

- Better than CNN?

Datasets	GP	DBN+GP	CNN	CNN+GP	DKL
Olivetti	16.33	6.42	6.34	6.42	**6.07**
MNIST	1.25	1.03	0.59	0.56	**0.53**

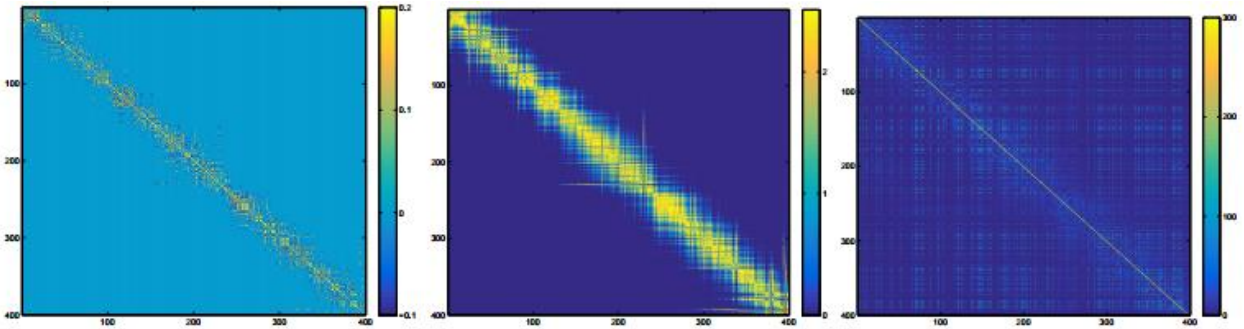


Figure 5: **Left**: The induced covariance matrix using DKL-SM kernel on a set of test cases, where the test samples are ordered by the *orientations* of the input faces. **Middle**: The respective covariance matrix using DKL-RBF kernel. **Right**: The respective covariance matrix using regular RBF kernel. The models are trained with $n = 12,000$, and $Q = 4$ for the SM base kernel.

Fooling images for CNN

- CNNs easily produce very high-confidence of unrecognizable images
- CNNs typically return many “fooling images” into one-of-classes with very high confidence (99.9%)
- It appears because it is a discriminative model?

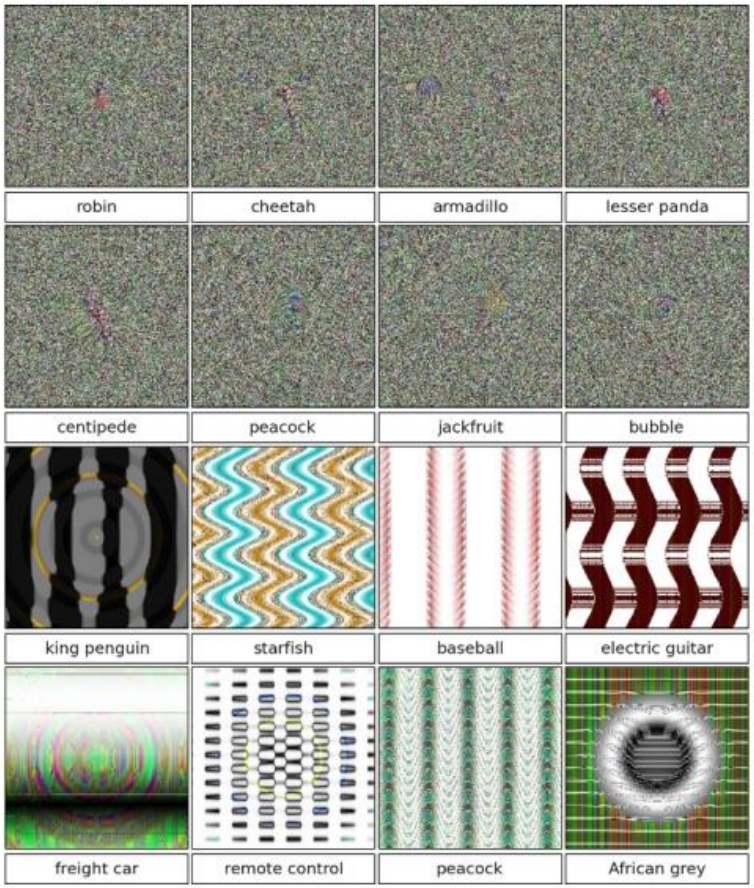


Figure 1. Evolved images that are unrecognizable to humans, but that state-of-the-art DNNs trained on ImageNet believe with $\geq$ 99.6% certainty to be a familiar object. This result highlights differences between how DNNs and humans recognize objects. Images are either directly (*top*) or indirectly (*bottom*) encoded.

A. Nguyen et al. Deep neural networks are easily fooled: high confidence predictions for unrecognizable images, CVPR’15.

Fooling image examples

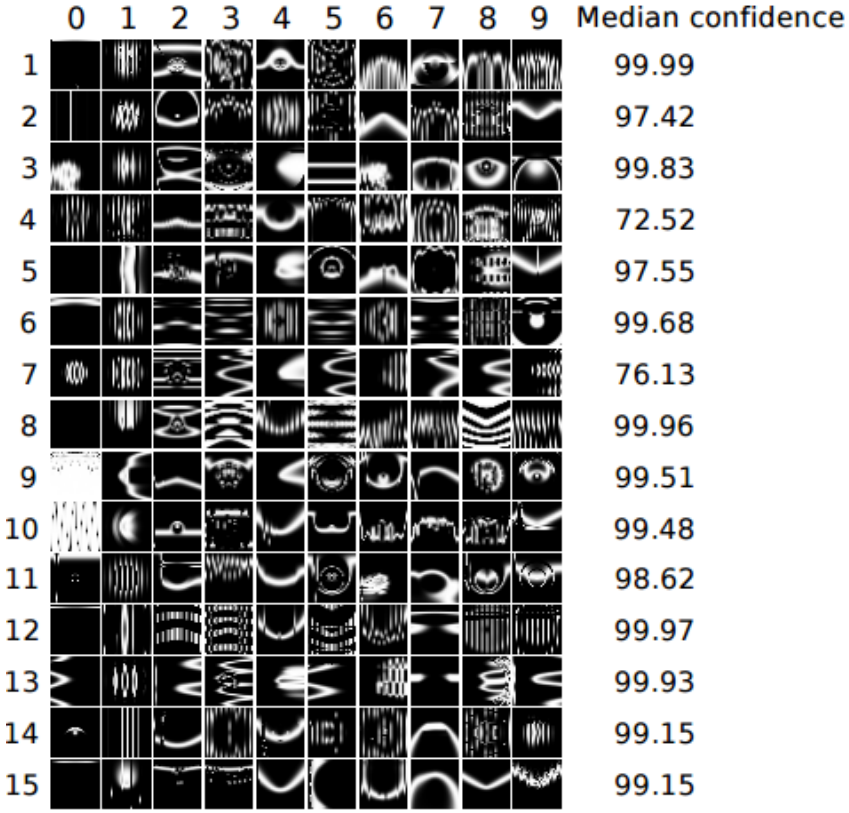


Figure 11. Training MNIST DNN_i with images that fooled MNIST DNN_1 through DNN_{i-1} does not prevent evolution from finding new fooling images for DNN_i. Columns are digits. Rows are DNN_i for $i = 1...15$. Each row shows the 10 final, evolved images from one randomly selected run (of 30) per iteration. Medians are taken from images from all 30 runs.

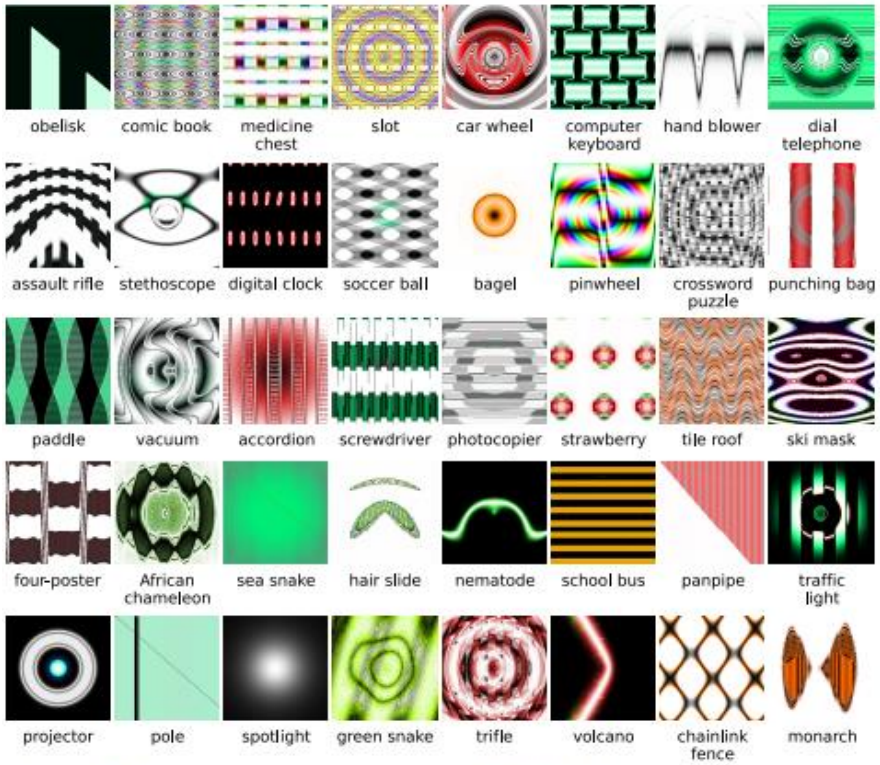


Figure 8. Evolving images to match DNN classes produces a tremendous diversity of images. Shown are images selected to showcase diversity from 5 evolutionary runs. The diversity suggests that the images are non-random, but that instead evolutions producing discriminative features of each target class. The mean DNN confidence scores for these images is 99.12%.